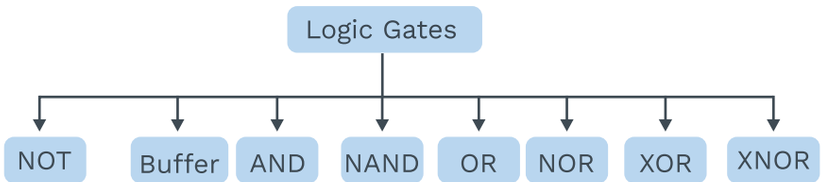
1 Boolean Algebra

1.1 LOGIC GATES

Definition

It is the basic building block to construct any digital circuit.



Note:

We can implement any Boolean function using the above gates.

What are 1 & 0 in Digital Logic?

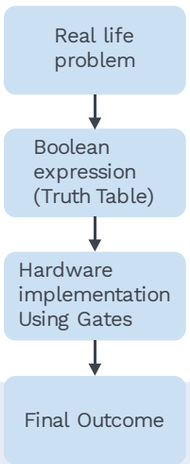
1: True	0: False
1: high	0: low
1: Yes	0: NO

Table 1.1

Note:

Truth table gives value of output for all possible input combination.

How are real-world problems solved using a digital logic system?



Positive and negative logic:

Definition

In positive logic, 1 represents “high”, and 0 represents “low”.
In negative logic, 1 represents “low”, and 0 represents “high”.

Buffer:
Symbolic representation

y = i [output = input]

Fig. 1.1 Symbolic Representation of Buffer

Truth table	
i	y
0	0
1	1

Table 1.2 Truth Table of Buffer

Note:

A buffer increases the strength of the signal so that it travels for a longer distance

Waveform

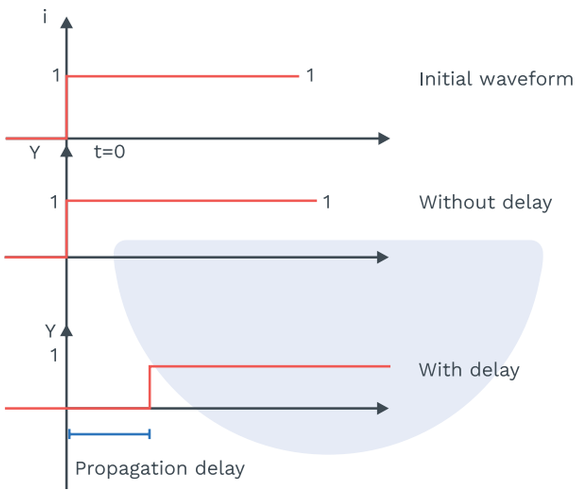


Fig. 1.2 Waveform of Buffer

Propagation Delay is the time taken by a signal to go from input to output of a logic Gate. When the buffer has no delay, it follows the input simply. When the buffer has a propagation delay, then after it senses the input at t = 0, it produces the output after propagation Delay.

Note:

In order to show output with propagation delay we shift output right by t_{pd}.
$t_{pd} \rightarrow$ propagation delay

Inverter:

Definition

Inverter/NOT gate complements the logic.

Symbolic representation

Fig. 1.3 Symbolic Representation of Inverter

Binary logic:

Sample Space, $S = \{0, 1\}$

$0^C = 1$

$1^C = 0$

Grey Matter Alert!

NOT gate and Inverter are same and are used to complement a logic.

Truth table	
i	$y = i^c$
0	1
1	0

Table 1.3 Truth Table of Inverter

Waveform:

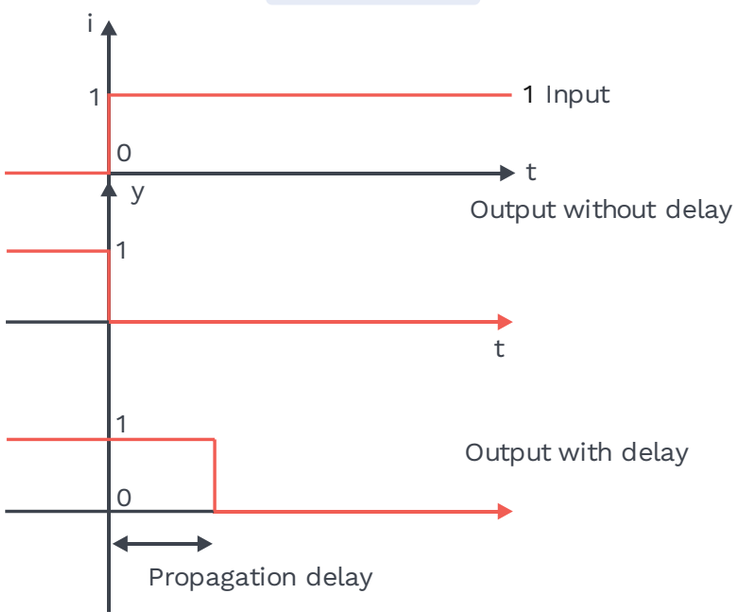


Fig. 1.4 Waveform of NOT Gate

When the NOT Gate has propagation delay, after it senses the input at t=0, it complements the input after the propagation delay.

Cascading of Inverters:

Definition

Whenever o/p of one inverter act as an input to another inverter.

Two inverters in cascade:

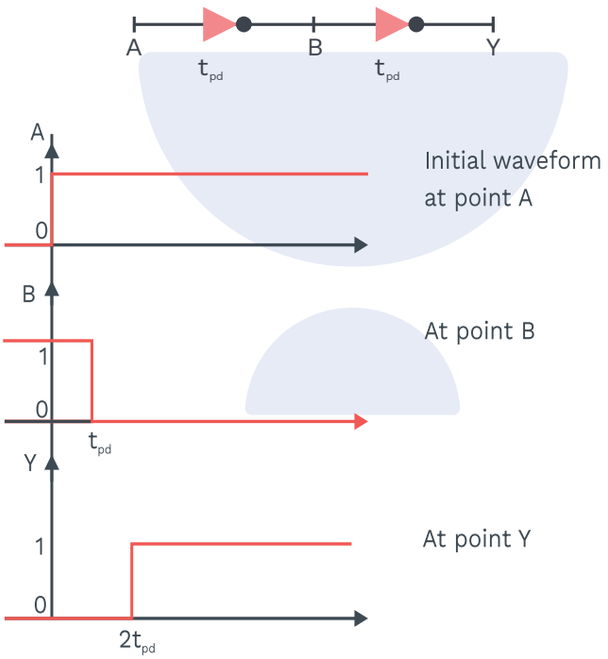


Fig. 1.5 Waveform of two Inverters in Cascade

Note:

Two inverters in cascade behave like a buffer with delay = $2t_{pd}$

Three inverters in cascade:

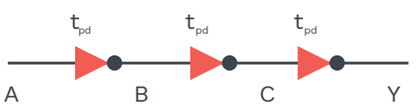


Fig. 1.6 Symbolic Representation of Three Inverters in Cascade

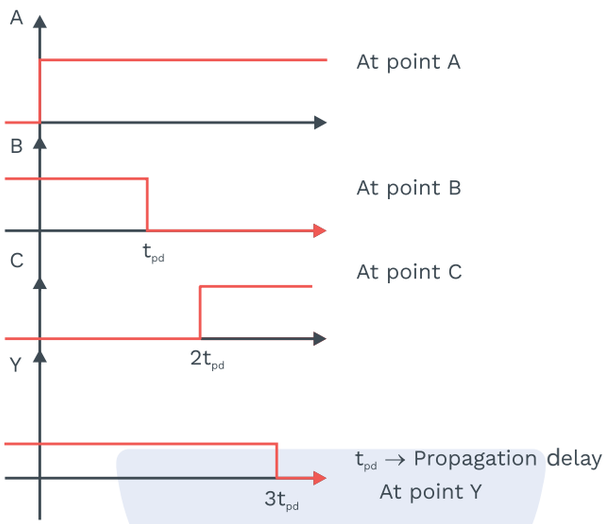


Fig. 1.7 Waveform of Three Inverters in Cascade

Note:

Three inverters in cascade behave like a buffer with delay $3t_{pd}$

NOT gate with feedback:

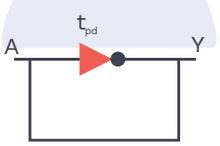


Fig. 1.8 Symbolic Representation of NOT Gate with Feedback

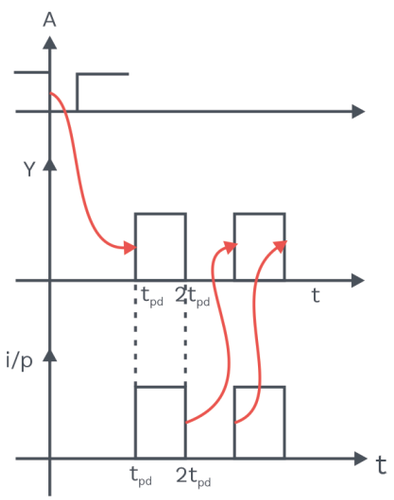


Fig. 1.9 Waveform of NOT Gate with Feedback

- Apply trigger input at 'A', which means input is applied for a short duration
- O/p signal is again feedback to the i/p, Arrow in the timing diagram represents input-output effect.
- Whenever input changes, the output will change after t_{pd}.

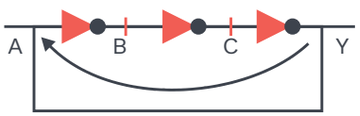


Fig. 1.10 Symbolic Representation of Three Inverters in Cascade with Feedback

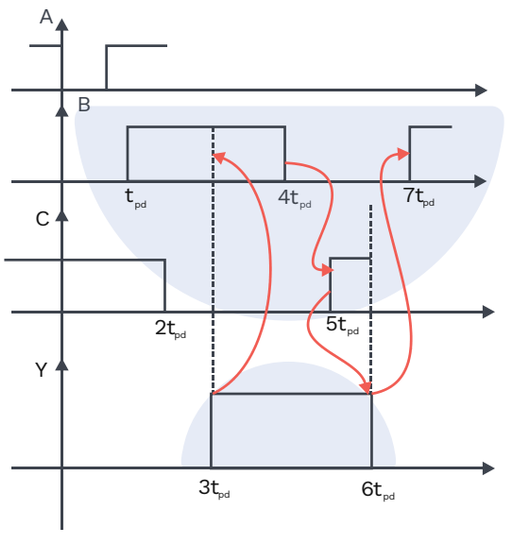
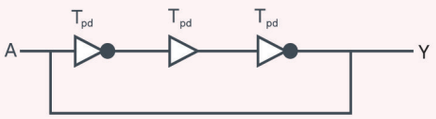


Fig. 1.11 Waveform of Three Inverters in Cascade with Feedback

Note:

Period of square wave (r) = 6tpd
= 2 × 3 × tpd
= 2n tpd
n = number of NOT gates in cascade (odd)

Rack Your Brain



What is the clock period of the output waveform Y?

Note:

Buffer does not impact shape of the Output but provides delay

AND gate:

Definition

Both inputs must be true for the output to be true.

Symbolic representation:

A, B are input, Y is the output

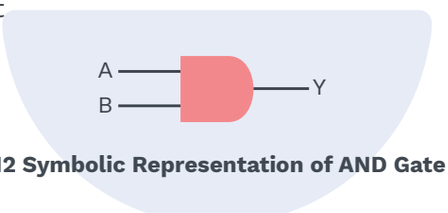


Fig. 1.12 Symbolic Representation of AND Gate

Truth table		
A	B	Y
0	0	0
0	1	0
1	0	0
1	0	0
1	1	1

Table 1.4 Truth Table of NAND Gate

O/p = High when both inputs are high, and if any input = low, then output = low

a) Commutative law $\Rightarrow$ A.B = B.A

b) Associative law $\Rightarrow$ (AB).C= A.(BC)

(Fan-in=2) (Fan-in=3)

Fig. 1.13 Symbolic Representation of NAND Gate with 3 i/ps

Note:

Fan-in defines maximum number of digital inputs that a single logic gate can accept.

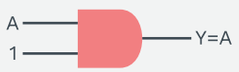
Truth table			
A	B	C	Y = ABC
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

Table 1.5 Truth Table of NAND Gates with 3 i/ps

Grey Matter Alert!

Enable and Disable AND Gate

The input A is not passed to output
∴ it is disable



The input A is passed to output
→ it acts as a buffer
→ it is enabled

Floating input:

- One of the i/p lines is not connected anywhere.

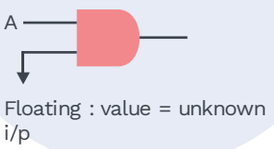


Fig. 1.14 AND Gate with Floating Input

OR gate:

Definition

At least one of the input is High/True for output to be High/True.

Symbolic representation:

Fig. 1.15 Symbolic Representation of OR Gate

Truth table		
A	B	Y = A + B
0	0	0
0	1	1

1	0	1
1	1	1

Table 1.6 Truth Table of OR Gate

1) **Commutative law:**
   A + B = B + A
   - OR Gate follows commutative law.

2) **Associative law:**
   (A+B) + C = A + (B + C)

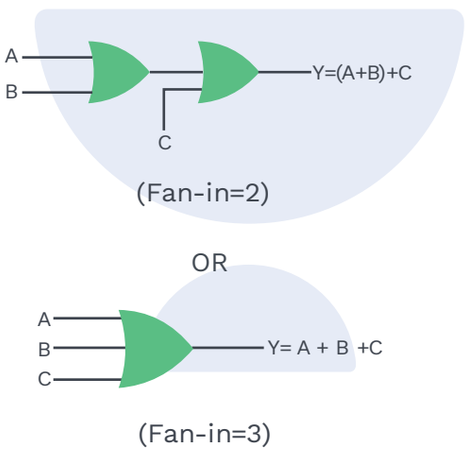


Fig. 1.16 Truth Table of OR Gate

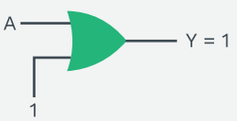
Truth table			
A	B	C	Y = A + B + C
0	0	0	0
0	0	1	1
0	1	0	1

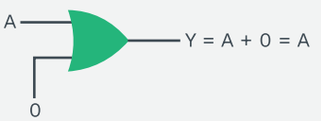
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.7 Truth Table of OR Gate with 3 i/ps

Grey Matter Alert!

Enable & Disable OR gate





→ o/p does not depend on i/p
→ disabled OR gate

→ behaves as a buffer
→ enabled OR gate

Floating i/p:

- Floating i/p is not connected to any value & its value is undetermined.

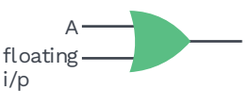


Fig. 1.17 OR Gate with Floating Input

NAND gate:

Definition

It is a combination of invertor and AND gate
NAND = AND output is noted

Symbolic representation:

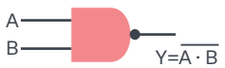


Fig. 1.18 Symbolic Representation of NAND Gate

Truth table		
A	B	$Y = \overline{A \cdot B}$
0	0	1
0	1	1
1	0	1
1	1	0

Table 1.8 Truth Table for NAND Gate

Commutative law:

NAND gate follows commutative law.

$\overline{AB} = \overline{BA}$

Associative law:

NAND gate does not follow associative law.

$\overline{\overline{AB} \cdot C} \neq \overline{A \cdot \overline{BC}}$

NOR gate:

Definition

A HIGH output results if both the inputs to the gate are LOW(0); if one or both input is HIGH(1), a LOW output results.

Symbolic representation:

Fig. 1.19 Symbolic Representation of NOR Gate

Truth table		
A	B	$X = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

Table 1.9 Truth Table for NOR Gate

Note:

- NOR gate output is HIGH if both i/ps are low else o/p is LOW
- A NOR gate behaves the same as an AND gate with inverted inputs.
- A NOR gate behaves the same as an OR gate with inverted outputs.

Commutative law

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

NOR gate is commutative.

Associative law

$$\overline{\overline{(A + B)} + C} \neq \overline{A + \overline{(B + C)}}$$

$\therefore$ NOT associative.

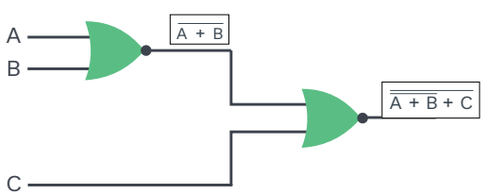
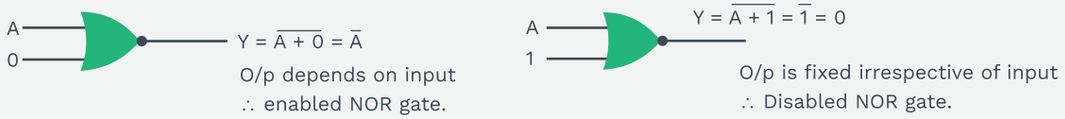


Fig. 1.20

Grey Matter Alert!

Enable & Disable NOR gate



Floating input:

- Floating input can be made 0 or connected to existing i/p

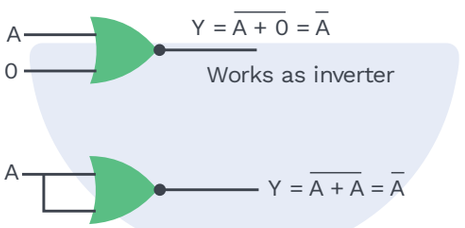


Fig. 1.21

Note:

1) $\overline{A + 0} = \bar{A}$
2) $\overline{A + 1} = 0$
3) $\overline{A + A} = \bar{A}$
4) $\overline{A + \bar{A}} = 0$

XOR gate

Definition

XOR gate is a digital logic gate that gives a true output when the number of true or HIGH(1) input is odd.

Symbolic representation:

Fig. 1.22 Symbolic Representation of XOR Gate

Truth table		
A	B	$X = A \oplus B$
0	0	0
0	1	1
1	0	1
1	1	0

Table 1.10 Truth Table for XOR Gate

Note:

XOR gate is also called inequality detector

Logic expression:

$X = A\overline{B} + \overline{A}B \xrightarrow{\text{implementation}}$

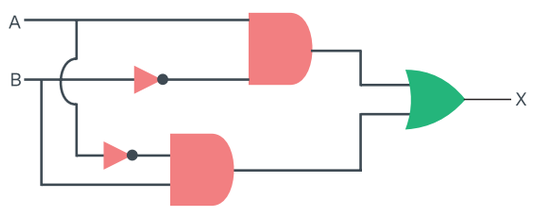


Fig. 1.24 Implementation of XOR Gate Using Basic Gates

Commutative law: $A \oplus B = B \oplus A$
Associative law: $A \oplus (B \oplus C) = (A \oplus B) \oplus C$

Properties of XOR gate:

$Y = A \oplus B \oplus C$

o/p = 1 if an odd number of 1's are present at the input

Truth table			
A	B	C	$Y = A \oplus B \oplus C$
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 1.11 Truth Table for XOR Gate with 3 i/ps

1) $A \oplus 0 = A\overline{0} + \overline{A}0 = A$(buffer)

Fig. 1.25

2) $A \oplus 1 = A\overline{1} + \overline{A}1 = \overline{A}$ (inverter)

Fig. 1.26

3) $A \oplus A = 0$

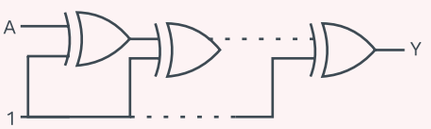
4) $A \oplus \overline{A} = 1$

5) if $A \oplus B = C$

Then $A \oplus C = B$
$B \oplus C = A$

Rack Your Brain

Consider a logic circuit with 30 XOR Gates connected as shown below. Find Y



XNOR Gate

Definition

The XNOR gate is a digital logic gate whose function is the logical complement of the Exclusive OR gate.

Symbolic representation:

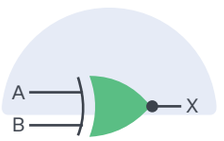


Fig. 1.27 Symbolic Representation of XNOR Gate

Note:

XNOR gate is called equality detector

Truth table		
A	B	$X = A \odot B$
0	0	1
0	1	0

1	0	0
1	1	1

Table 1.12 Truth Table for XNOR Gate

Logic expression:

$X = AB + \overline{A}\,\overline{B}$

Commutative law:

$A \odot B = B \odot A$

Associative law:

$(A \odot B) \odot C = A \odot (B \odot C)$

Properties:

1) $A \odot 0 = \overline{A}\,\overline{0} + A0$

$= \overline{A} \cdot 1$

$= \overline{A}$ (inverter)

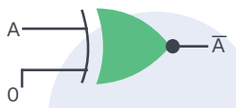


Fig. 1.28 XNOR Gate as Inverter

2) $A \odot 1 = A1 + \overline{A}\,\overline{1}$

$= A + A \cdot 0$

$= A$(Buffer)

Fig. 1.29 XNOR Gate as Buffer

3) $A \odot A = 1$

4) $A \odot \overline{A} = 0$

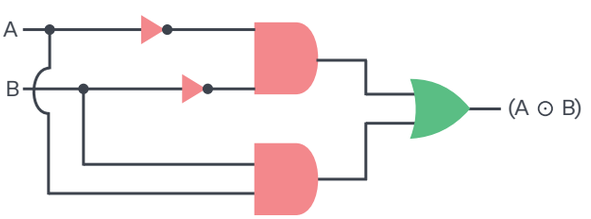


Fig. 1.30 Implementation of XNOR Gate Using Basic Gates

Truth table				
A	B	C	$(A \odot B)$	$Y = A \odot B \odot C$
0	0	0	1	0
0	0	1	1	1
0	1	0	0	1
0	1	1	0	0
1	0	0	0	1
1	0	1	0	0
1	1	0	1	0
1	1	1	1	1

Table 1.13 Truth Table for XNOR Gates

Note:

- For odd Number of i/p, XOR = XNOR
- For even Number of i/p, XNOR = $\overline{XOR}$

1.2 BOOLEAN LAWS

AND law	OR Law	Inversion law
1. $A \cdot 0 = 0$	1. $A + 0 = A$	1. $\overline{\overline{A}} = A$
2. $A \cdot 1 = A$	2. $A + 1 = 1$	
3. $A \cdot A = A$	3. $A + A = A$	
4. $A \cdot \overline{A} = 0$	4. $A + \overline{A} = 1$	

Table 1.14 Laws of Boolean Algebra

Implication and inhibition gate:

Implication gate:

$$Y = \overline{\overline{A} \cdot B} = \overline{\overline{A}} + \overline{B} = A + \overline{B}$$

if B = 0, Y = 1

B = 1, Y = A i.e if IMPLY = 1

Y = A

Or If 'B' is true, then A appears in the output.

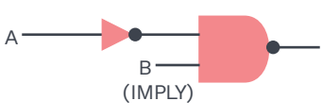


Fig. 1.31 Implication Gate

Inhibition gate:

$$Y = \overline{\overline{A} + B} = A \cdot \overline{B}$$

If B = 1, Y = 0 i.e A does not reach o/p

B = 0, Y = A.

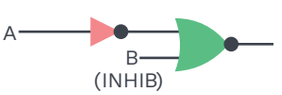


Fig. 1.32 Inhibition Gate

Theorems and properties of Boolean algebra:

1) Idempotent law
- $x.x = x$
- $x + x = x$

2) Associative law
- $(x + y) + z = x + (y + z)$
- $(x.y) . z = x . (y . z)$

3) Complement law
- $x + \overline{x} = 1$
- $x . \overline{x} = 0$

4) Commutative law
- $x + y = y + x$
- $x . y = y . x$

5) Distributive law
- $x . (y + z) = x . y + x . z$
- $x + y . z = (x + y) . (x + z)$

6) Identity law
- $x + 1 = 1$
- $x + 0 = x$
- $x . 0 = 0$
- $x . 1 = x$

Demorgan's first theorem:

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

Truth table			
A	B	$\overline{AB}$	$\overline{A} + \overline{B}$
0	0	1	1

0	1	1	1
1	0	1	1
1	1	0	0

Table 1.15 Truth Table - Demorgan's Law

Fig. 1.33 Implementation - Demorgan's Law

NAND gate = Bubbled i/p OR gate

Demorgan's second theorem:

$$\overline{A + B} = \overline{A} \cdot \overline{B}$$

Truth table			
A	B	$\overline{A + B}$	$\overline{A} \cdot \overline{B}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

Table 1.16

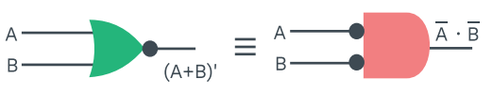


Fig. 1.34

NOR gate = Bubbled i/p AND gate

Transposition theorem:

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

Proof

RHS

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

$= AA' + AB + A'C + BC$

$= AB + A'C + BC$

$= AB + A'C + BC(A + A')$

$= AB + ABC + A'C + A'BC$

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

$= AB + A'C = LHS$

Consensus theorem/ redundancy theorem:

The Redundancy theorem is used to simplify Boolean expressions. It is also known as consensus theorem.

$$AB + A'C + BC = AB + A'C$$

The consensus of the terms AB and A'C are BC. The conjunctive dual of the above equation is:

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

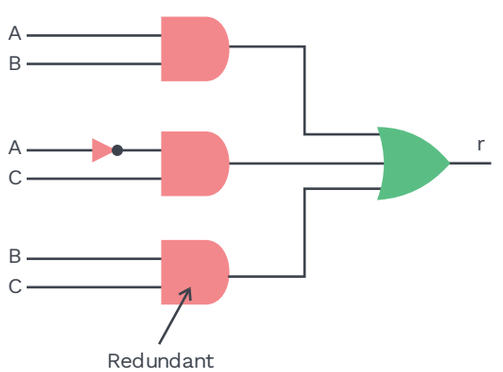


Fig. 1.35

Example $\Rightarrow \overline{B}C + \overline{A}C + B\overline{A}$

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

Conditions for applying the redundancy theorem are:

1) Three variables must be present in the Boolean expression.

2) Each variable is repeated twice.

3) One variable in the expression must be present in both complemented and uncomplemented forms.

After applying this theorem, we can only take those terms which contain the complemented variable.

Proof

Y = AB + A'C + BC

Y = AB + A'C + BC(1)

Y = AB + A'C + BC (A + A')

Y = AB + A'C + ABC + A'BC

Y = AB (1 + C) + A'C (1 + B)

Y = AB + A'C (Proved)

Redundant literal rule:

A + A'B = A + B [Literal $\bar{A}$ is redundant & can be removed]

Similarly,

A (A' + B) = AB

Proof

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

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

$\Rightarrow$ A + B (1) = A + B

Duality:

Definition

This theorem states that the dual of the Boolean function is obtained by interchanging logical AND operator with logical OR operator and 0's with 1's & vice-versa.

Following Boolean equations are dual to each other:

Group 1		Group 2
x + 0 = x	↔	x . 1 = x
x + 1 = 1	↔	x . 0 = 0
x + x = x	↔	x . x = x
x + x' = 1	↔	x . x' = 0
x + y = y + x	↔	x . y = y . x
x + (y+z) = (x + y)+z	↔	x . (yz) = (x . y) . z
x(y+z) = xy + xz	↔	x + yz = (x + y) (x + z)

Complementary theorem:
For obtaining complement expression,

1) Change each OR sign by AND sign and vice-versa.
2) Complement any 0 or 1 appearing in the expression.
3) Complement the individual literals.

Example $\Rightarrow$ Complement of $A(B+C) = A'+(B'\cdot C') = (A'+B')(A'+C')$

1.3 BOOLEAN FUNCTION

Definition

A Boolean function is described by an algebraic expression consisting of binary variables, the constants 0 and 1 and logical symbols '+' and ' . '. For a given set of input, a Boolean function can have value of '0' or '1'

A Boolean function can be represented in canonical form:
i) POS (product of sum)
ii) SOP (sum of product)

Example
f = a + bc

Truth table			
a	b	c	f

0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.17 Truth Table of 'f'

Rack Your Brain

How many m-ary functions are possible with 'n' K-ary variables?

SOLVED EXAMPLES

Q1 **f (p, q, r) = p'q + pqr + r', Represent it in canonical SOP form**

Sol: f (p, q, r) = p' q + pqr + r'
= p'q (r + r') + pqr + r' ' (p + p') (q + q')
= p'qr + p'qr' + pqr + pqr' + p'qr' + p'q'r' + pq'r'
= p'qr + pqr + pqr' + pq'r' + p'q'r' + p'qr'

Functional properties:

- The canonical sum of product or product of sum form of a switching function is UNIQUE.
- The switching function $f_1(x_1,.....x_n)$ and $f_2(x_1,x_2.....x_n)$ are said to be logically equivalent if both functions have same value for each and every combination of $x_1, x_2..... x_n$.
- Two switching functions are equivalent, if their canonical SOP and canonical POS expressions are equal.

Note:

For a switching function, there can be many expressions which will define the same function, but when we convert them to their corresponding SOP and POS form, it is UNIQUE.

Neutral function:

Number of minterms = Number of maxterms

Example

For 2 variable Boolean function number of neutral functions possible = ${}^4C_2$

A	B	f_1	f_2
0	0	0	1
0	1	0	1
1	0	1	0
1	1	1	0

Table 1.20 Neutral Functions with 2 Variables

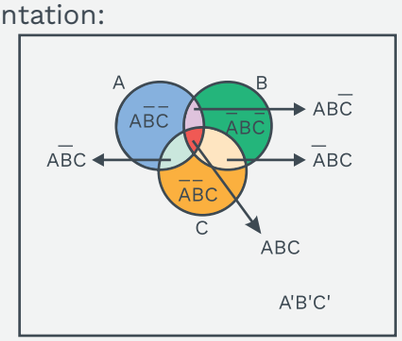
For 'n' variable function:

Number of neutral function $= {}^{2^n}C_{\frac{2^n}{2}}$

$= {}^{2^n}C_{2^{n-1}}$

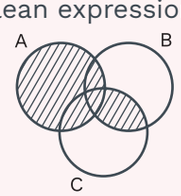
Grey Matter Alert!

Venn diagram representation:



Rack Your Brain

Find the Boolean expression



Simplifications of Boolean functions:

1) Simplification using algebraic functions:

SOLVED EXAMPLES

Q2 **F(X,Y,Z)=(X+Y)(X+Z)**

Sol: F(X,Y,Z) = (X+Y)(X+Z)

= X.X + X.Z + Y.X + Y.Z

= X + X.Z + X.Y + Y.Z

= X (1 + Z) + X.Y + Y.Z

= X + X.Y + Y.Z

= X (1 + Y) + Y.Z
= X + Y.Z (minimized form)

2) Simplification using K-map: (Discussed later)

Complement of Boolean function:

We can find the complement of a function using given 2 rules in Demorgan's law:

1) Change the OR Gates with AND Gates or change the AND Gates with OR Gates

2) Change each literal of the function with its complement.

Example F = pq' + p'q

F' = (p' + q) · (p + q')

= p'q' + pq [∵ p·p' = 0]

1.4 STANDARD SOP AND POS FORMS

The main advantage of standard forms is to minimize the number of inputs to logic gates. Sometimes there will be a reduction in the total number of logic Gates required to represent a Boolean function.

1) Standard SOP form

2) Standard POS form

Note:

- Canonical form is unsimplified
- Standard form is simplified

Standard sum of product (SOP):

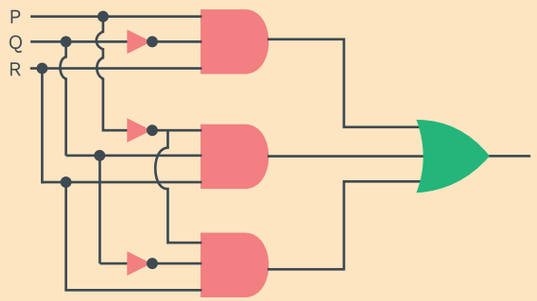
Definition

Standard SOP form means standard sum of products form. In this form, each product term need not contain all literals. So product terms may/may not be minterms.

∴ Standard SOP form is the simplified form of canonical SOP form.

SOLVED EXAMPLES

Q3 **Implement the function given below using SOP and POS forms:**



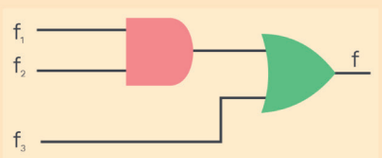
Sol:

$F = P\bar{Q}R + \bar{P}QR + \bar{P}\bar{Q}R$

$= \sum m\ (1, 3, 5)$

$= \pi M\ (0, 2, 4, 6, 7)$

Q4 **Consider the three 4 variable functions f_1, f_2, f_3.**
Given,f_1, f_3 and f in canonical SOP (m decimal) for the circuit
$f_1 = \Sigma m\ (8, 6, 7, 4, 5)$
$f_3 = \Sigma m\ (1, 6, 15)$
$f = \Sigma m\ (6, 1, 8, 15)$

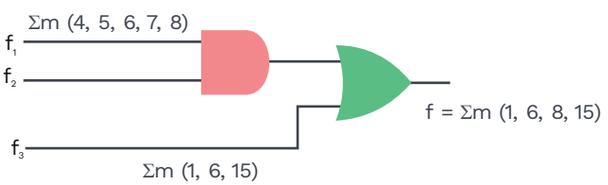


Then f_2 is -

1) $\Sigma m\ (4, 6)$ **2)** $\Sigma m\ (4, 8)$

3) $\Sigma m\ (6, 8)$ **4)** $\Sigma m\ (4, 6, 8)$

Sol: AND: Intersection of minterms
i.e. common minterm between 2 function
OR: Union of minterms



$f = (f_1 \wedge f_2) \vee f_3$
f_2 must be $\sum m\ (6, 8)$
Option (3)

1.5 CANONICAL FORM

Minterms and maxterms:

Minterms:

Definition

A minterm of n variables is a product of variables in which each variable appears exactly once in true or complemented form.
Each minterm = 1 for only one combination of values of the variables.
= 0, otherwise

A	B	C	Minterm
0	0	0	$\bar{A}\bar{B}\bar{C}$ m_0
0	0	1	$\bar{A}\bar{B}C$ m_1
0	1	0	$\bar{A}B\bar{C}$ m_2
0	1	1	$\bar{A}BC$ m_3
1	0	0	$A\bar{B}\bar{C}$ m_4
1	0	1	$A\bar{B}C$ m_5
1	1	0	$AB\bar{C}$ m_6
1	1	1	ABC m_7

Table 1.21 Minterms of 3 variables

Note:

For a particular combination of input values, if any input variable = 0, take complement else use as it is & take product of all such variables.

Maxterms:

Definition

A maxterm of n variables is sum of variables in which each variable appears exactly once in true or complemented form. Each maxterm = 0 for only one combination of values of the variables.

= 1, otherwise

Note:

For a particular combination of input values, if any input variable = 1, take complement else use as it is and take sum of all such variables.

Canonical SOP and POS forms:

Definition

If there are 'n' input variables, then there will be 2^n possible combinations with zeros and ones. So value of each output variable depends on combination of input variable, So each output variable will have “1” for some combination of input variables and 'O' for some other combination of input variables.
Therefore each output variable can be expressed in following ways:

- Canonical SOP form ⟶ minterms: product of variables
- Canonical POS form ⟶ maxterm: sum of variables

Canonical SOP

Canonical SOP form means the canonical sum of product form. In this form, each product term contains all variables or literals. Hence canonical SOP is called the sum of minterms form.

Example

Input			Output
x	y	z	F
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1 m_3
1	0	0	0
1	0	1	1 m_5
1	1	0	1 m_6
1	1	1	1 m_7

Table 1.22 Table Showing Minterms

$f = m_3 + m_5 + m_6 + m_7$
$f = \sum m\ (3, 5, 6, 7)$

The Boolean function of the output f is x'yz + xy'z + xyz' + xyz

Canonical POS:

Definition

Canonical POS form means canonical product of sum from. In this form, each sum term contains all literals. So these Sum terms are the Maxterms. First, identify the Max terms for which the output variable is zero and then perform logical AND of those MAX-terms to get Boolean expression function corresponding to that output variable.

Input			Output
p	q	r	f
0	0	0	0 M_0
0	0	1	0 M_1
0	1	0	0 M_2
0	1	1	1
1	0	0	0 M_4
1	0	1	1
1	1	0	1
1	1	1	1

Table 1.23 Table Showing Maxterms

$f = M_0 . M_1 . M_2 . M_4$

$f = \pi M (0, 1, 2, 4)$

$f = (p + q + r) (p + q + r') (p + q' + r) (p' + q + r)$

Conversion between canonical forms:

Let's look at an example:

$F = ABC + AB\bar{C} + A\bar{B}C + \bar{A}BC$

In terms of minterms the functions can be written as:

$F = m_3 + m_5 + m_6 + m_7 = \sum m\ (3, 5, 6, 7)$ Sum of minterms

In terms of maxterms the function can be written as:

$F = (A + B + C)(A + B + \bar{C})(A + \bar{B} + C)(\bar{A} + B + C)$

$= \pi\ M\ (0, 1, 2, 4)$ ← Consider those maxterms which are not included in minterms

Product of Maxterms.

Conversion In Standard Form:

SOLVED EXAMPLES

Q5 **Convert the following Boolean function into standard SOP form:**
f = A' B C + A B' C + A B C' + A B C

Sol:

f = A' B C + A B' C + A B C' + A B C
= A' B C + A B' C + AB
= B [A + A' C] + A B' C
= B [A + C] + A B' C
= A B + B C + A B' C
= A [B + B' C] + B C
= A (B + C) + B C
= A B + A C + B C

Q6 **Convert the following Boolean function into standard POS form:**
f = (X + Y + Z) (X + Y + Z') (X + Y' + Z) (X' + Y + Z)

Sol:

f = (X + Y + Z) · (X + Y + Z') (X + Y' + Z) (X' + Y + Z)
f = (X + Y + Z) (X + Y + Z') (X + Y + Z) (X + Y' + Z) (X + Y + Z) (X' + Y + Z)
f = (X + Y + Z Z') (X + Z + Y Y') (Y + Z + X X')
f = (X + Y) (X + Z) (Y + Z)
using (A+B) · (A+C)= A + BC]

1.6 MINIMIZATION OF BOOLEAN FUNCTION

Logic minimization – using K-map:

Definition

K-Map is a graphical method which consists of 2^n cells for 'n' variables. The adjacent cells are differed by a single bit position.

Note:

K-map method is most suitable for minimizing the Boolean Function of 2 variables to 5 variable.

Grouping of K-map variables:

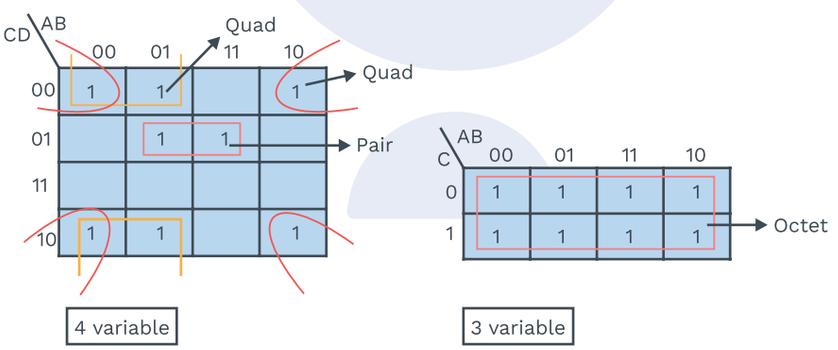


Fig. 1.36 Grouping of K-map Variables

- Groups can overlap.
- The number of literals in a group must be equal to the power of 2, such as 1,2,4,8 etc.
- Groups can wrap around. The squares at the corners (which are at the end of a column or row) should be considered as adjacent squares.
- The grouping of K-map variables can be done in different ways, So obtained simplified equations need not be UNIQUE always.
- The boolean equation must be in canonical form (SOP/POS) to draw the K–map.
- If we group 2^K cells together, then K variables are eliminated.

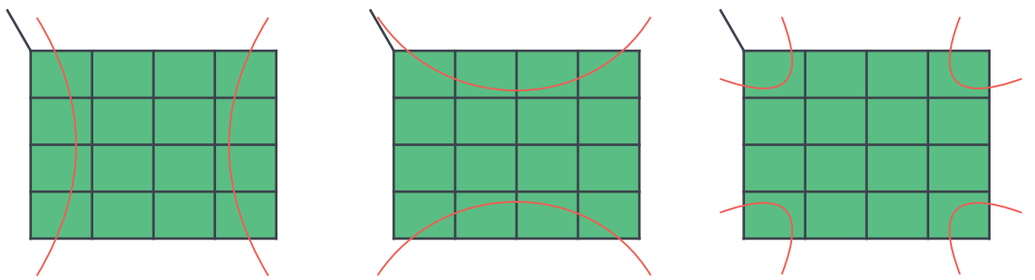


Fig. 1 Fig. 2 Fig. 3

Fig. 1.37 Wrapping Around K-map

2) Variable K-maps:

The possible minterms with 2 variables (A and B) are AB, AB', A'B, A'B'

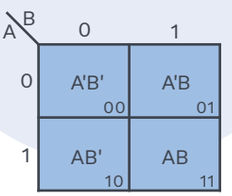


Fig. 1.38

Example

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

Table 1.24

- For SOP, consider minterms, where F = 1.

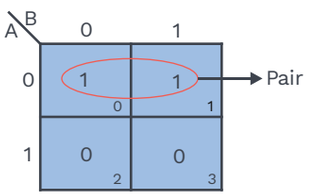


Fig. 1.39

In the pair, 'B' is changing from 0 to 1.

∴ $\boxed{F = \overline{A}}$

- For POS consider maxterms, where F = 0

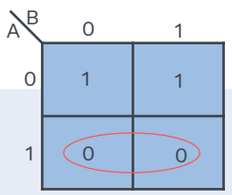


Fig. 1.40

In the pair, 'B' is changing from 0 to 1.

∴ $\boxed{F = \overline{A}}$ (maxterm)

3-Variable K-maps: 8 Possible minterms are:

A	B	C	O/P function
0	0	0	A'B'C'
0	0	1	A'B'C
0	1	0	A'BC'
0	1	1	A'BC
1	0	0	AB'C'
1	0	1	AB'C
1	1	0	ABC'
1	1	1	ABC

Table 1.25

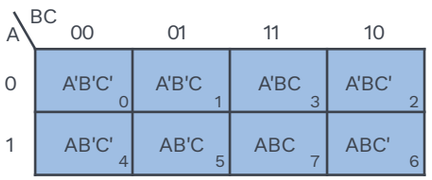


Fig. 1.41

Example

F = ∑m (5, 1, 0, 2, 4)

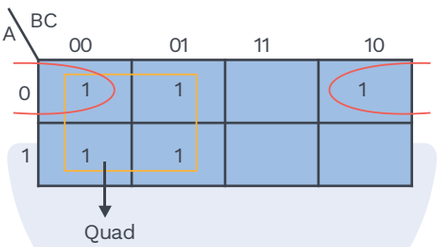


Fig. 1.42

$\boxed{F = \overline{B} + \overline{A}\overline{C}}$

Example

F = π M (3, 6, 7)

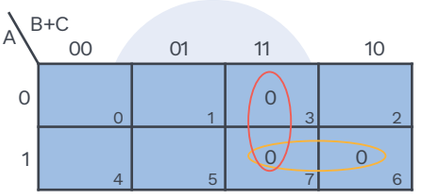


Fig. 1.43

$\boxed{F = (\overline{B} + \overline{C})(\overline{B} + \overline{A})}$

4 variable K-map:

F (y, z, w, x) = ∑m (4, 1, 5, 7, 13, 9, 14, 15)

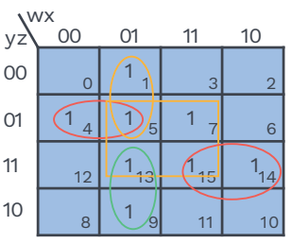


Fig. 1.44

F = w'y'z + w'xy' + w'xy + wyz + xz (14 literals)

But this is not a minimized expression; we need to do the grouping carefully so that the biggest group ("octet" then "Quad" then "pair") gets the first priority.

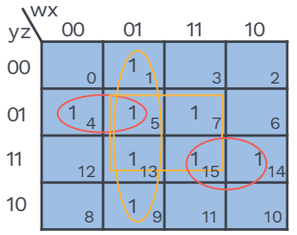


Fig. 1.45 4-Variable K-Map

f = w'y'z + wyz + w'x + xz (10 literals) (minimum)

1.1.2 Implicants, prime implicants and essential prime implicants:

Definition

If 'f' covers 'g' then 'g' is said to imply 'f'. This is denoted by g → f ['g' is a implicant of 'f']

Ex. = f (x, y, z) = xy + z

x	y	z	$f_1 = xy$	$f_2 = z$	f = xy + z
0	0	0	0	0	0
0	0	1	0	1	1
0	1	0	0	0	0
0	1	1	0	1	1
1	0	0	0	0	0
1	0	1	0	1	1

1	1	0	1	0	1
1	1	1	1	1	1

Table 1.26 Truth Table of 'f'

$\because$ f is covering both f_1 and f_2
if f_1 or f_2 is 1, f is 1
$\therefore$ f_1 and f_2 are implicants of f.
$f_1 \rightarrow f$
$f_2 \rightarrow f$

Note:

In an SOP expression, each product term is implicant of the original function.

Prime implicant:

Definition

An implicant 'P' of a function 'f' is said to be prime implicant if:
i) P is any product term (i.e. subcube)
ii) Subcube P can not be part of any bigger subcubes completely

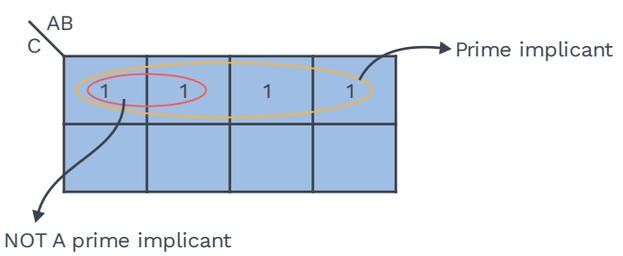


Fig. 1.46 3-Variable K-Map

($\because$ it is a part of the bigger subcube completely)

Essential prime implicants:

Definition

A Prime implicant 'P' of a function 'f' is said to be an essential prime implicant if it covers at least one minterm of 'f', which is not covered by any other prime implicants.

Example

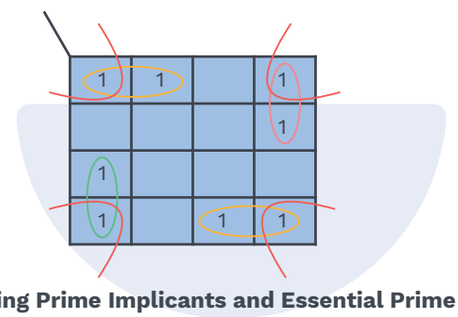


Fig. 1.47 K-Map Showing Prime Implicants and Essential Prime Implicants

5 Prime implicants
4 Essential prime implicants

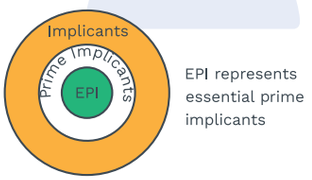


Fig. 1.48

Rack Your Brain

AB \ CD	00	01	11	10
00	1	1		1
01		1	1	1
11			1	1
10	1			1

Find number of prime implicants and essential prime implicants.

Note:

In cyclic K map

- Every minterms is covered by atleast 2 prime implicants.
- All prime implicants are of the same size.

Procedure for obtaining minimal SOP:

1) Determine all essential prime implicants and include them in the minimal SOP.

2) Remove all those prime implicants which are covered by essential prime implicants.

3) If the set determined in step 1 covers all the minterms of 'f' then it is a unique minimal expression; otherwise, select the additional prime implicants so that the function 'f' is covered completely and the total number and size of prime implicants added are minimal.

Note:

Prime implicants will be chosen such that literals in the term are minimum.

Example

F(yzwx) = ∑m (4, 3, 5, 7, 13, 9, 14, 15)

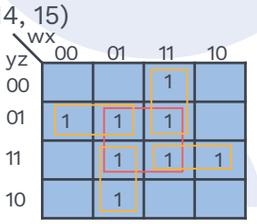


Fig. 1.48

5 Prime implicants

4 Essential prime implicants

f = w'y'z + wxy' + wyz + w'xy

Prime implicant chart:

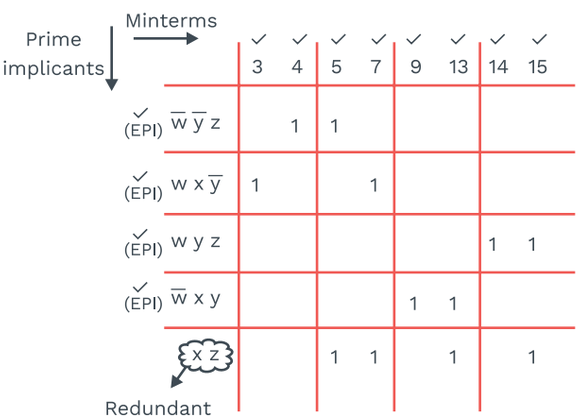


Fig. 1.49 Prime Implicant Chart

- All essential prime implicants are included in SOP (Find essential prime implicants from k-map).
- Include the prime implicants in the minimal expression which is covering maximum minterms.
- All minterms must be covered.
- 'xz' is redundant prime implicant since after including all the essential Prime implicants, all of the minterms get covered.

Note:

Redundant Prime Implicants are never included in minimal SOP.

Don't care conditions:

Definition

- A function is said to be completely specified if it is given 0 or 1 for every combination of i/p variables.
  There are some function which are not completely specified.
- Combination for which value of function is not fully specified are called don't care combinations.
- Since each don't care combination represent two values {0, 1}, an incompletely specified function containing K don't care combinations, corresponds to a class of 2^k distinct function.

Example

a	b	f	f_1	f_2	f_3	f_4
0	0	1	1	1	1	1
0	1	1	1	1	1	1
0	0	ø	0	0	1	1
1	1	ø	0	1	0	1

Table 1.27 Table Showing Don't Cares

ø ← don't care

Note:

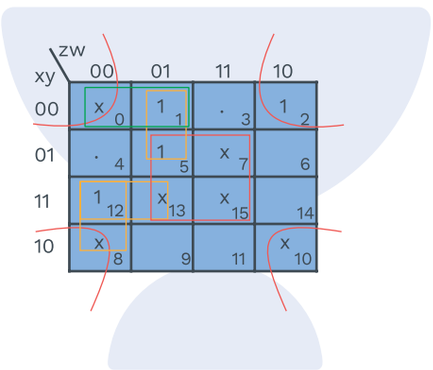
We can assign '0' or '1' to a don't care $\emptyset$ in such a way to increase/decrease the size of a prime implicant.

Understanding through examples:

SOLVED EXAMPLES

Q7 **F = S m (2, 1, 5, 12) + S d (0, 7, 8, 10, 13, 15)**
Find minimized Boolean expression

Sol:



x ← don't care
6 prime implicant

Note:

Here, EPIs do not include all the minterms so, minimal SOP is not UNIQUE here.

2 Possibilities exist:

1) 2 QUAD, 2 Pair: $F = yw + \bar{y}\bar{w} + \bar{x}\,\bar{y}\,\bar{z} + x\bar{z}\bar{w}$ (10 literals) (NOT Minimal)

2) 1 QUAD, 2 Pair: $F = \bar{y}\bar{w} + \bar{x}\bar{z}w + x\bar{z}\bar{w}$ (8 literals) (Minimal)

Let's look at how to find the number of minimal expressions.

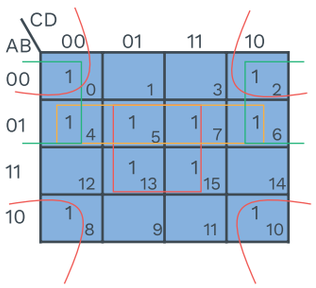


Fig. 1.50

Minterms →	0 ✓	2 ✓	4	5 ✓	6	7 ✓	8 ✓	10 ✓	13 ✓	15 ✓
PI. ↓										
$\bar{B}\,\bar{D}$ (EPI)	1	1					1	1		
B D (EPI)				1		1			1	1
A' B			1	1	1	1				
A' D'	1	1	1		1					

Both these prime-implicants are covering the left over minterms (m_4, m_6) we can include either of them in the final expression

Fig. 1.51 Prime Implicant Chart

Final SOP ~ $\boxed{\bar{B}\bar{D} + BD + \bar{A}B}$

OR

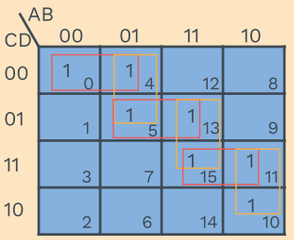
$\boxed{\bar{B}\bar{D} + BD + \bar{A}\bar{D}}$

Note:

Both minimal SOP and minimal POS will be independent of the same number of variables if there are no don't cares.

SOLVED EXAMPLES

Q8 **Find number of Minimal SOP in the following K-map**



Sol: Prime implicant chart

		0 ✓	4 ✓	5	13	15	11 ✓	10 ✓
(EPI)	P = $\bar{A}\,\bar{C}\,\bar{D}$	1	1					
	Q = $\bar{A}\,B\,\bar{C}$			1				
	R = $B\,\bar{C}\,D$			1	1			
	S = $A\,B\,D$				1		1	
	T = $A\,C\,D$						1	1
(EPI)	V = $A\,\bar{B}\,C$						1	1

Let's reduce the prime implicant chart (include all the EPIs in the Minimal SOP and removing the all minterms associated with the EPIs)

	5	13	15
Q	1		
R	1	1	
S		1	1
T			1

Minimal SOP → P + V + R + S
OR
P + V + R + T
OR
P + V + Q + S
} 3 Minimal SOP

Variable entrant map: (VEM)

Definition

A variable entrant map (VEM) is a K-map in which the size of the map is reduced by removing one or more variables from the specification of the map cell locations

$f(x, y, z) = \sum (1, 2, 3, 5, 6)$

X	Y	Z	f	
0	0	0	0	} z
0	0	1	1	
0	1	0	1	} 1
0	1	1	1	
1	0	0	0	} z
1	0	1	1	
1	1	0	1	} $\bar{z}$
1	1	1	0	

VEM →

X	Y	f
0	0	z
0	1	1
1	0	z
1	1	$\bar{z}$

Table 1.28 Table Showing VEM

Minimization procedure for VEM:

- Set all the variables in the cell as '0' and obtain the SOP expression.
- Make one variable in the cell as '1' and obtain SOP by making earlier minterms as don't cares ($\not{0}$) and multiply the obtained SOP expression with the concerned variable.
- Repeat step 2 until all the variables in the cell are covered.
- SOP of VEM is obtained by performing 'OR' operation with previously obtained SOP expression.

SOLVED EXAMPLES

Q9 **Find the minimized Boolean expression for the following VEM**

Z \ XY	00	01	11	10
0	D	1	D'	D'
1	D	1	0	Ø

Sol:

Step 1

Z \ XY	00	01	11	10
0	0	1	0	0
1	0	1	0	Ø

SOP = X'Y

Step 2

For D :

Z \ XY	00	01	11	10
0	1	Ø	0	0
1	1	Ø	0	Ø

SOP = X'

For D' :

Z \ XY	00	01	11	10
0	0	Ø	1	1
1	0	Ø	0	Ø

SOP = XZ'

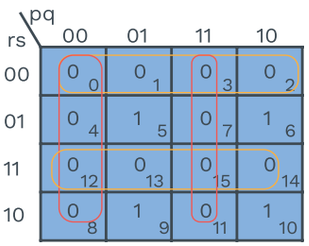
Step 3

$\boxed{X'Y + X'D + XZ'D'}$

Product of Sum Simplification

Q10 **f (p,q,r,s) = π M (0,1,2,3,4,7,8,11,12,13,14,15) Find Minimal**

Sol:



4 Essential Prime Implicants

Minimal POS = $(r + s)(p + q)(\overline{r} + \overline{s})(\overline{p} + \overline{q})$ ← [8 literals]

Note:

Minimal expression = All essential prime implicants
+
Prime implicants needed to cover remaining minterms, if any.

Previous Years' Question

The literal count of a Boolean expression is the sum of the number of times each literal appears in the expression. For example, the literal count of (xy + xz') is 4. What are the minimum possible literal counts of the product-of-sum and sum-ofproduct representations respectively of the function given by the following Karnaugh map? Here, X denotes "don't care"

xy \ zw	00	01	11	10
00	X	1	0	1
01	0	1	X	0
11	1	X	X	0
10	X	0	0	X

1) (11, 9)
2) (9, 13)
3) (9, 10)
4) (11, 11)

Sol: 3) **(GATE-2003)**

1.7 NAND AND NOR

Universal gates:

Definition

- A universal gate can implement any Boolean function without using gates of any other type. NAND and NOR are the universal gates.
- In practice, this is advantageous since NAND and NOR gates are economical and easier to fabricate and are the basic gates used in all ic digital logic families.
- In fact, AND gate is implemented as a NAND gate followed by an inverter. OR gate is implemented as a NOR gate followed by an inverter.

1.7 IMPLEMENTATION OF BOOLEAN FUNCTIONS WITH NAND GATES

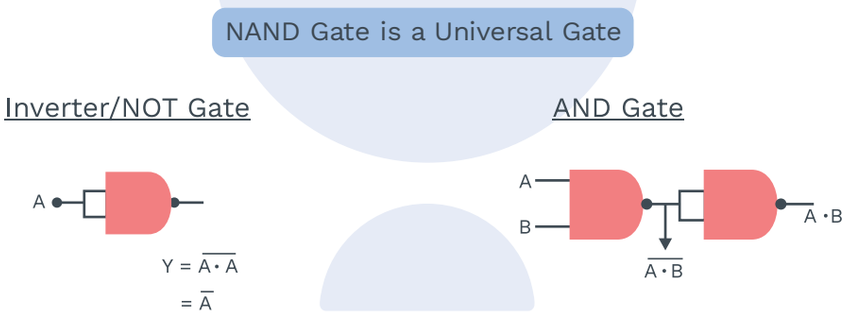


Fig. 1.50 Implementation Using NAND Gate

OR gate:

$Y = A + B = \overline{\overline{A} \cdot \overline{B}}$ [From Demorgan's law]

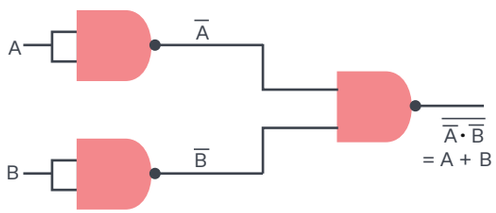


Fig. 1.51 Implementation Using NAND Gate

NOR gate:

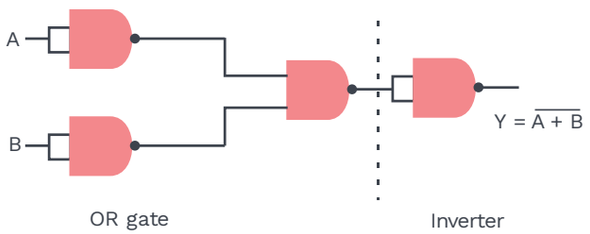


Fig. 1.52 Implementation Using NAND Gate

X-OR:

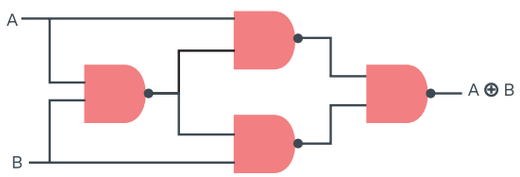


Fig. 1.53 Implementation of X-OR Gate using NAND Gate

4 NAND gates [minimum] required

XNOR:

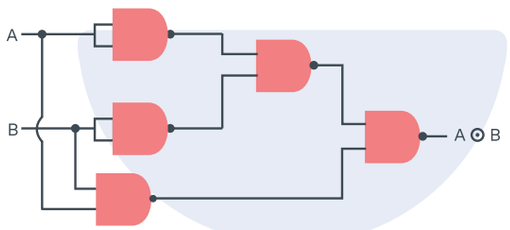


Fig. 1.54 Implementation of X-NOR Gate using NAND Gate

5 NAND Gates (minimum)

Implementation of Boolean functions with NOR gates:

NOR gate is a universal gate

Inverter/Not gate

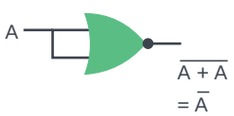


Fig. 1.55 Implementation of NOT Gate Using NOR Gate

AND gate

$y = A \cdot B = \overline{\overline{A} + \overline{B}}$ [By Demorgan's law]

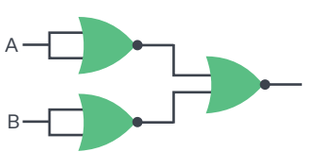


Fig. 1.56 Implementation of AND Gate Using NOR Gate

OR gate

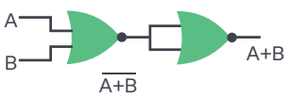


Fig. 1.57 Implementation of OR Gate Using NOR Gate

NAND gate

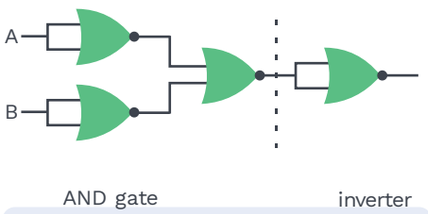


Fig. 1.58 Implementation of NAND Gate Using NOR Gate

X-OR

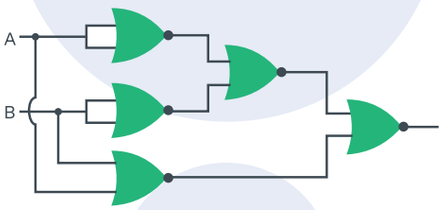


Fig. 1.59 Implementation of XOR Gate using NOR Gate

XOR = XNOR + Inverter
5 NOR gates [minimum]

XNOR gate

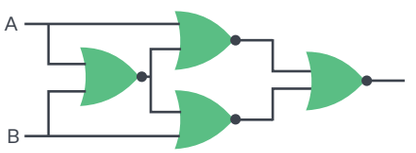


Fig. 1.60 Implementation of XNOR Gate Using NOR Gate

4 NOR gates [minimum] required

Note:

	Min NAND gates	Min NOR gates
X-OR	4	5
X-NOR	5	4

- Bubbled i/p OR gate = NAND gate
- Bubbled i/p AND gate = NOR gate.

Example $Y = AB + CD$

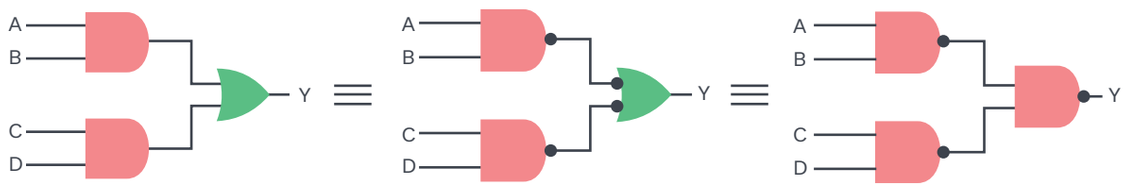


Fig. 1.61 Boolean Logic Circuit Implementing Function 'Y'

Example $Y = (A + B).(C + D)$

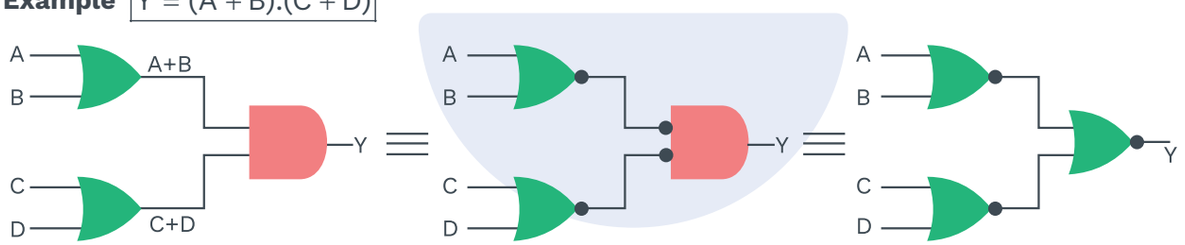


Fig. 1.62 Boolean Logic Circuit Implementing Function 'Y'

Note:

If the bubble is placed one after the other, it cancels each other's effect

SWITCH REPRESENTATION

Switch representation of the AND function:

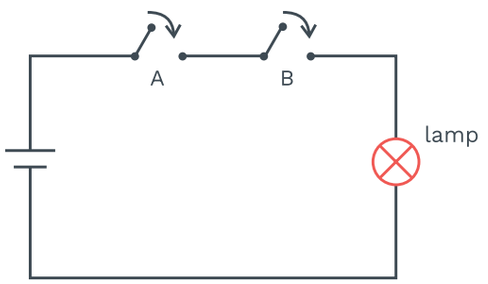


Fig. 1.63 Circuit Diagram Representing AND Function

Switch A – open = "0" closed = "1"

Switch B – open = "0" closed = "1"

Both switches must be ON for the lamp to be ON, if A = high, B = high, Y = high.

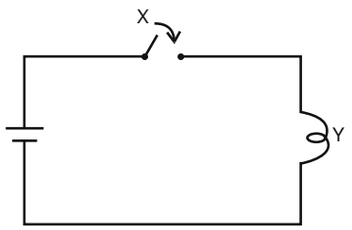


Fig. 1.64 Circuit Diagram Representing Buffer

Y = X, switch X is ON, lamp turns ON (Buffer)

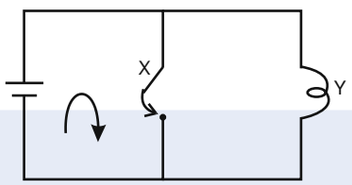
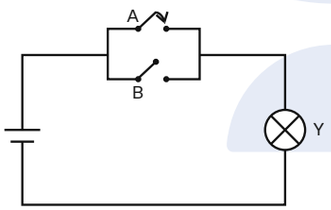


Fig. 1.65 Circuit Diagram Representing NOT Function

Switch X is ON, lamp is OFF

$Y = \overline{X}$ (Inverter)



If A is ON, lamp ON
if B is ON, lamp ON

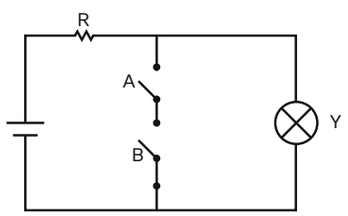
Y = A + B (OR)

Fig. 1.66 Circuit Diagram Representing OR Function

Rack Your Brain

A bulb that can be turned on by 2 switches (2-way switch) implements which gate?

Switch representation of the NAND function:



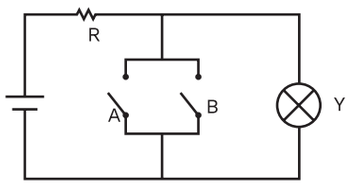
Lamp ON = "1"
Lamp OFF = "0"

Fig. 1.67 Circuit Diagram Representing NAND Function

If A & B are high, then the bulb gets shorted, lamp = OFF

$Y = \overline{AB}$

Switch representation of the NOR function:



If A = ON, lamp = OFF
B = ON, lamp = OFF

$Y = \overline{A + B}$

Fig. 1.68 Circuit Diagram Representing NOR Function

Switch representation of XOR function:

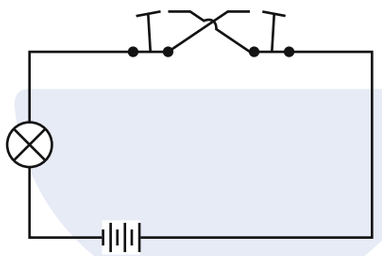


Fig. 1.69 Circuit Diagram Representing XOR Function

If both are up or both are down, the current can not flow
If one is up and another down, then current can flow.
∴ Current flows for unequal i/p
⇒ XOR gate

Switch representation of the XNOR function:

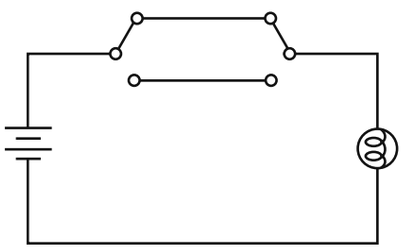


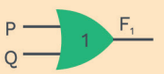
Fig. 1.70 Circuit Diagram Representing XNOR Function

If both switches are up, current flows
If both switches are down, current flows
∴ equality detector : XNOR gate

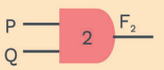
SOLVED EXAMPLES

Q11 **A universal gate can implement any Boolean function by connecting the sufficient number of them appropriately.**

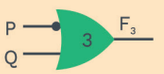
$F_1 = P + Q$



$F_2 = P \cdot Q$



$F_3 = \overline{P} + Q$



Which of the following is true?

1) Gate 1 is universal gate **2) Gate 2 is the universal gate**

3) Gate 3 is universal gate **4) None**

Sol: We already know, NOR and NAND universal gates.

$F_3 = \overline{P} + Q$

if $Q = 0$

$F_3 = \overline{P}$ [complement is generated]

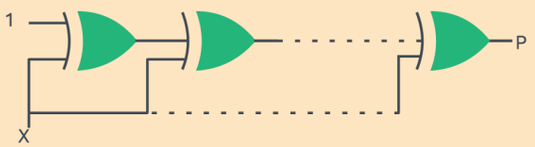
$F_3(P, Q) = \overline{P} + Q$ [given]

$F_3(\overline{P}, Q) = P + Q$ [OR is generated]

"OR" followed by complement is NOR, which is universal gate.

Ans. – Option (3)

Q12 **Below digital circuit consists of 20 XOR gates in cascade.**



output P is equal to:

1) 0 **2) $\overline{1}$**

3) X **4) $\overline{X}$**

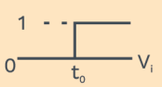
Sol: $X \oplus 1 = \overline{X}$

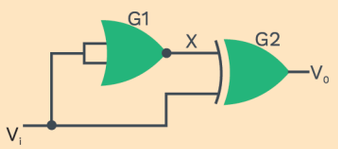
$\overline{X} \oplus X = 1$

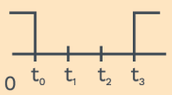
After 2 XOR Gate, o/p is 1, So after 20 XOR gates also, the output is going to be 1.

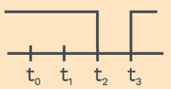
$\boxed{P = 1}$ $\boxed{\text{Ans.} - (2)}$

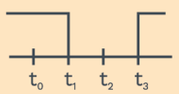
Q13 **The gates G_1 and G_2 have propagation delay 10 nsec and 20 nsec respectively. If input V_i makes an abrupt change from logic 0 to 1 at $t = t_0$ then o/p wave form V_0 is:**





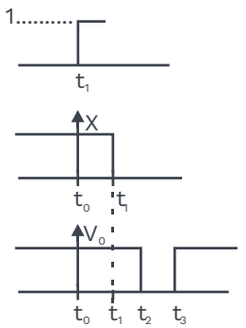
1) 

2) 

3) 

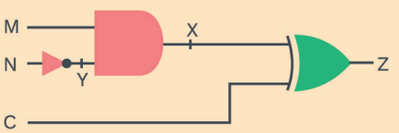
4) None

Sol:

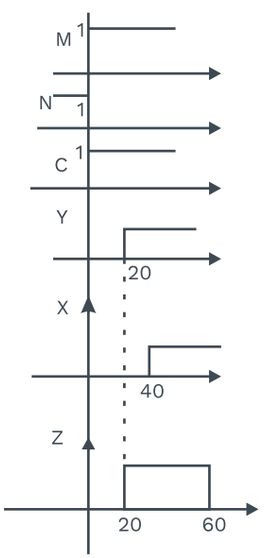


option 2)

Q14 **All logic gates shown in the figure below have a propagation delay of 20 ns.**
Let M = C = 0 and N = 1 until t = 0.
At t = 0, all the inputs flip and remain in that state.
For t > 0, output is high (1) for a duration (in ns) of _____.



Sol:



Ans. = 40

Chapter Summary

- AND - $A.B$
  OR - $A + B$
  NOT - $\overline{A}$
  NAND - $\overline{A.B}$
  NOR - $\overline{A + B}$
  XOR - $A\overline{B} + B\overline{A}$
  XNOR - $AB + \overline{A}\,\overline{B}$
- Associative law for XOR – $(A \oplus B) \oplus C = A \oplus (B \oplus C)$
  Commutative law for XOR – $A \oplus B = B \oplus A$
- AND law : $A.0 = 0$, $A.1 = A$
  OR law : $A + 0 = A$, $A + 1 = 1$
- Inversion law : $\overline{\overline{A}} = A$
- $A \oplus B \oplus C = A \odot B \odot C$
  $A \oplus B = \overline{A \odot B}$
- Demorgan's law:
  i) $\overline{A + B} = \overline{A}.\overline{B}$
  ii) $\overline{A.B} = \overline{A} + \overline{B}$
- Transposition law : $AB + A'C = (A + C)(A' + B)$
- Consensus theorem : $A'B + AC + BC = A'B + AC$
- Duality theorem: $A + BC \xrightarrow{\text{dual}} A(B + C)$
- Complementary theorem: $A(B+C) \xrightarrow{\text{comp.}} A' + (B'.C')$
- Standard SOP : each product term need not contain all literals.
- Canonical SOP : each product term must contain all literals.
- Minterm : Product of all variables where each variable appears only once either in true of complemented form
- Maxterm : Sum of variables, in which all variables appear once either in true or complemented form.
- Minimization of Boolean Expression through : K-map (2, 3, 4 variable maps)
- NAND and NOR are universal gates.
- Don't care conditions: It represents class of distinct functions.

2 Combinational Circuit

2.1 INTRODUCTION

Introduction to logic design:

Combinational circuits are those whose current input decides the current output. These circuits process the operations which can be fully logically specified with the help of Boolean functions. These circuits are formed of logic gates and input/output variables.

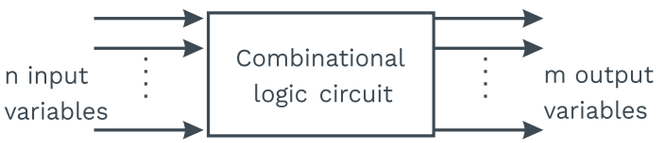
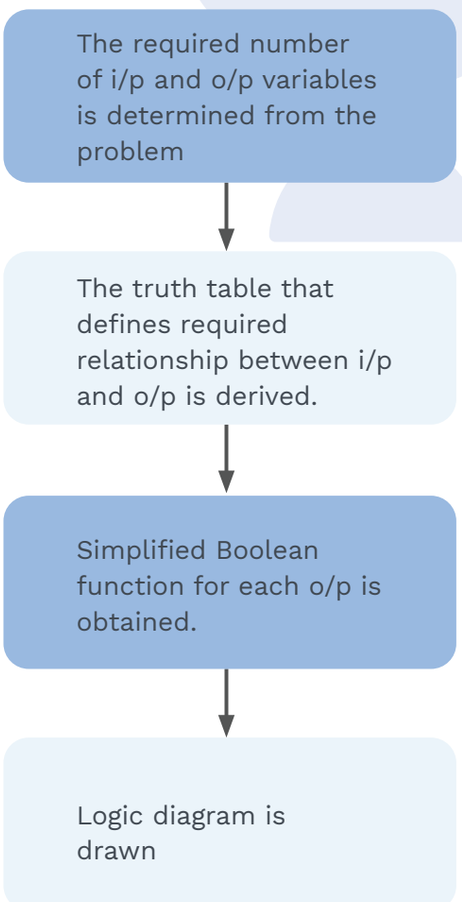


Fig. 2.1 Block Diagram of Combinational Circuit

A block diagram of a combinational circuit is shown in figure 2.1

Design procedure:



2.2 ADDER

Half adder:

Note:

- OR-AND realisation is equivalent to NOR-NOR realisation. (both of 2 levels)
  f(A, B, C, D) = (A + B) (C + D)

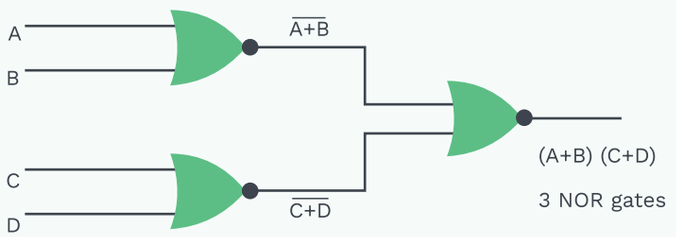


Fig. 2.2 NOR-NOR Realisation

Note:

- AND-OR realisation is equivalent to NAND-NAND realisation. f(A, B, C, D) = AB + CD

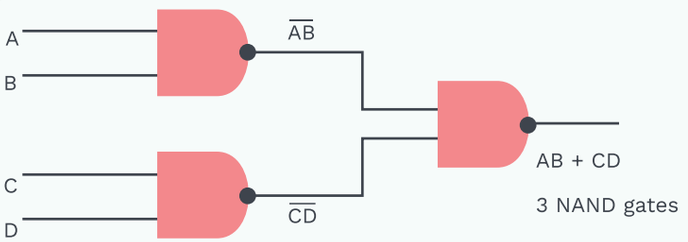


Fig. 2.3 NANDc-NAND Realisation

A half adder adds the 2 inputs and produces sum and carry bits.

Truth table			
A	**B**	**Sum**	**Carry**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table 2.1 Truth Table for Half Adder

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

Carry = AB

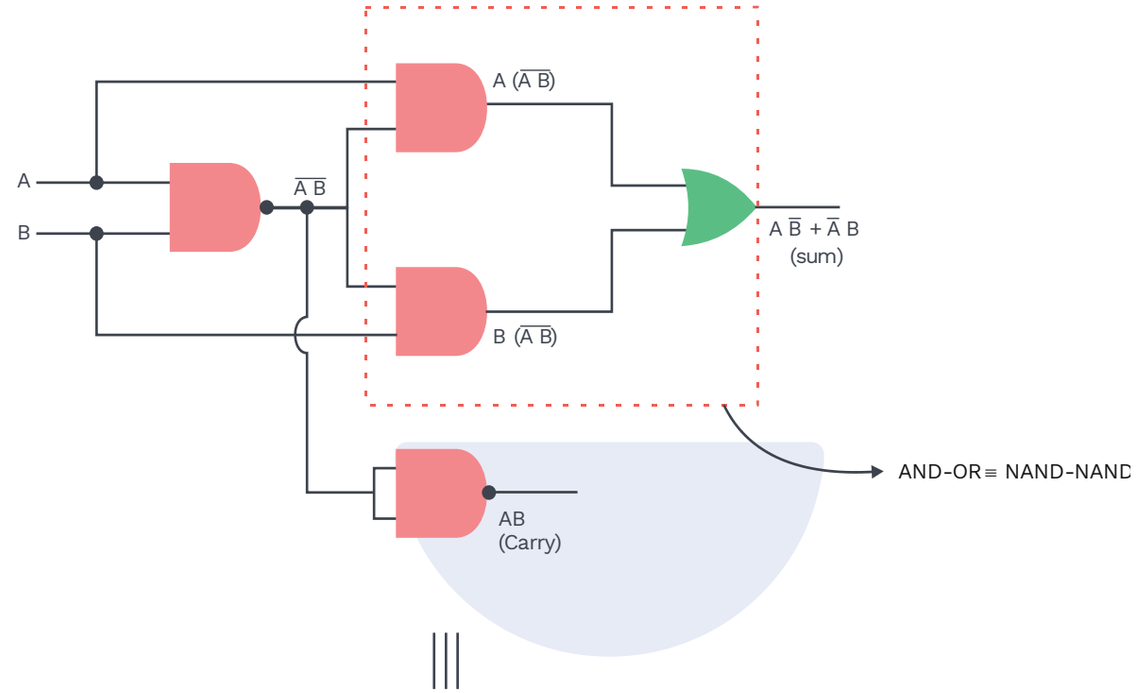


Fig. 2.4 Half Adder using AND-OR Realization

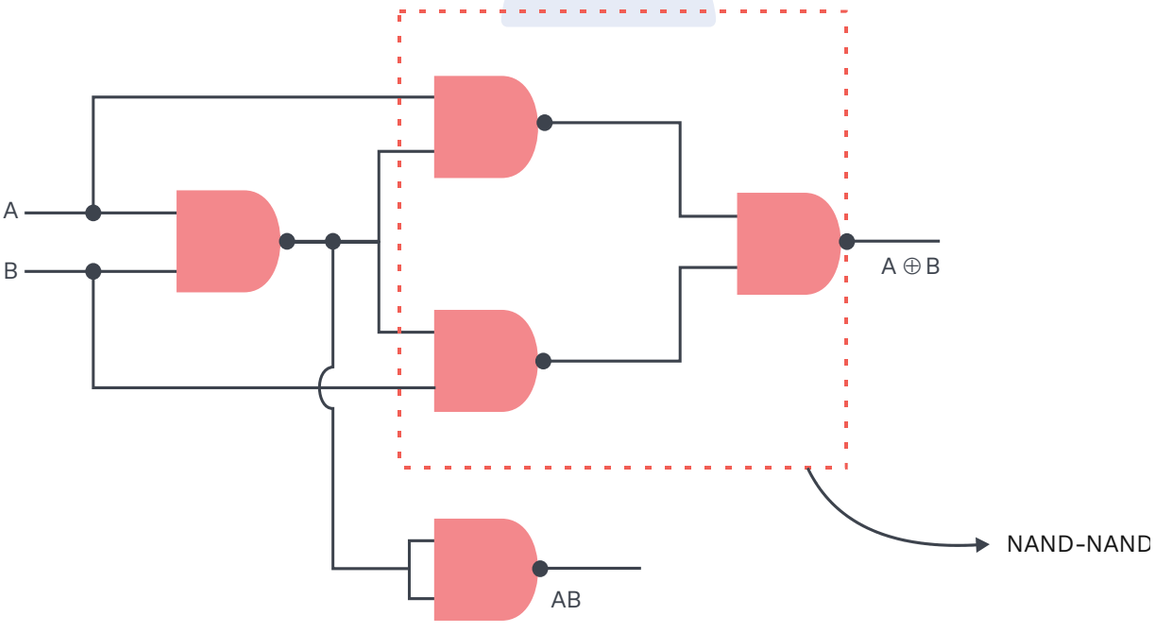


Fig. 2.5 Half Adder using NAND-NAND Realization

Half adder using NAND gate and NOR gate:

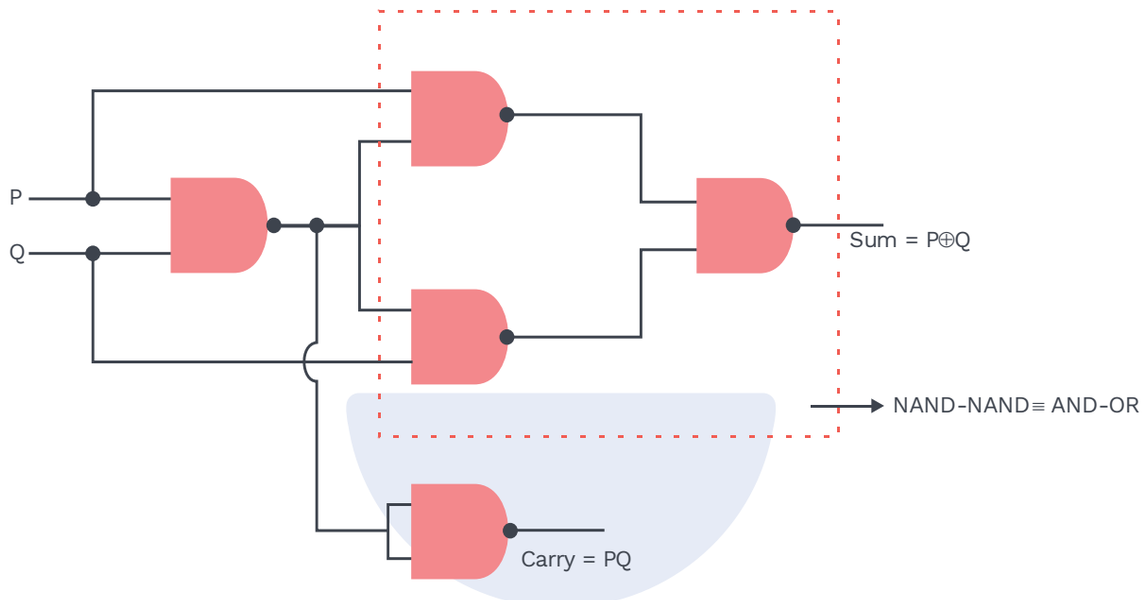


Fig. 2.6 Half Adder Circuit Implemented using only NAND Gate

Figure 2.6 represents a half adder circuit, implemented using only the NAND gate.

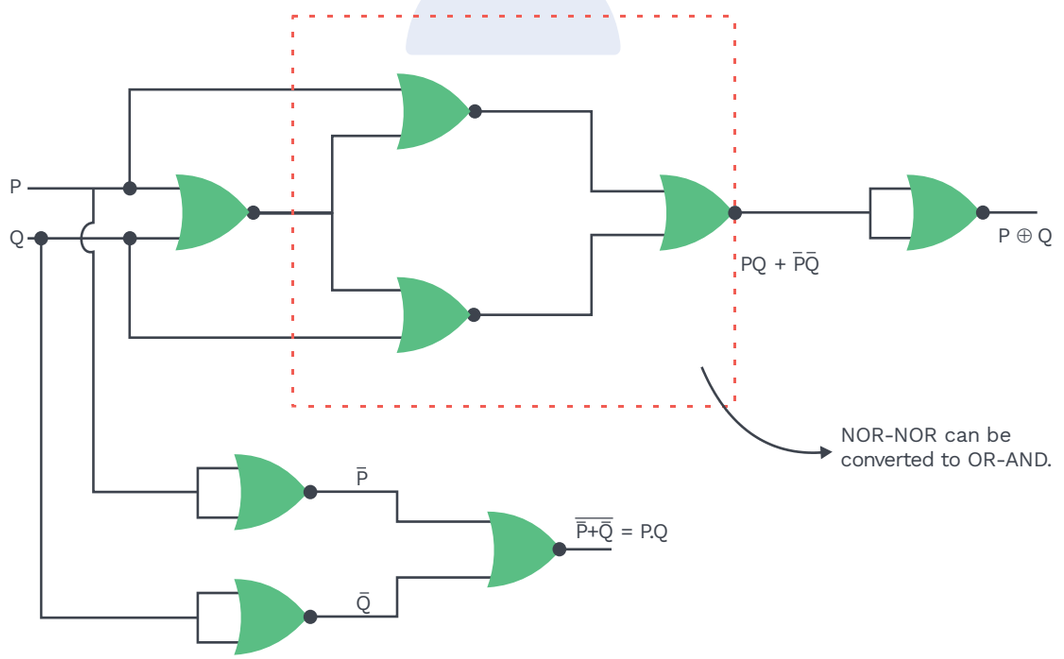


Fig. 2.7 Half Adder Circuit Implemented using only NOR Gate

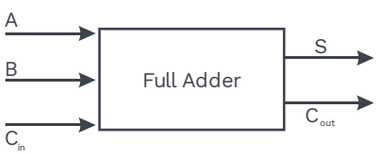
Figure 2.7 represents a half adder circuit, implemented using only NOR gate.

Full adder:

The full adder adds 3 bits and outputs sum bit S and carry bit, C_{out}.

Truth table				
Inputs			Sum	Carry
A	B	C_{in}	S	Cout
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

Table 2.2



For sum:

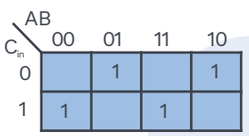


Fig. 2.8

It is a neutral function since the number of maxterms = the number of minterms.

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

For carry:

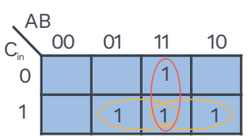


Fig. 2.9

$$C = AB + BC_{in} + AC_{in}$$

OR

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

We have got the Boolean expression for sum and carry, now we will implement the Boolean expression using logic gates.

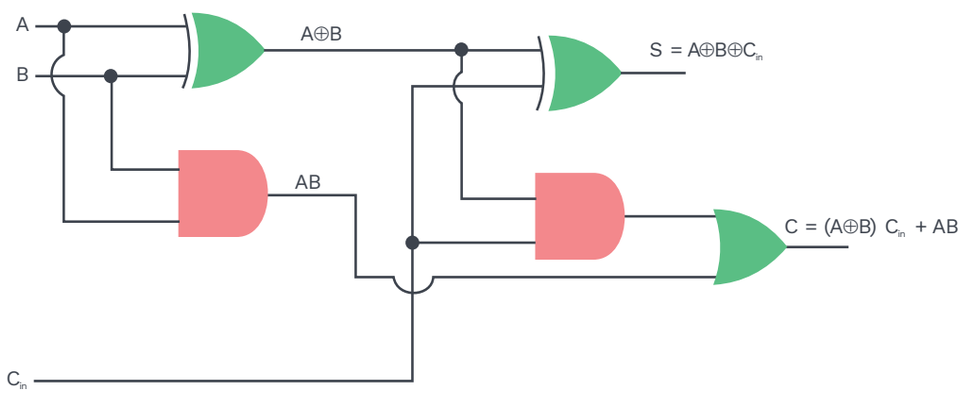


Fig. 2.10 Full Adder using 2 Half Adders and OR Gate

Parallel adder:

Definition

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

Figure 2.11 shows the connection between four full–adder (FA) circuits to provide a 4 bit parallel adder.

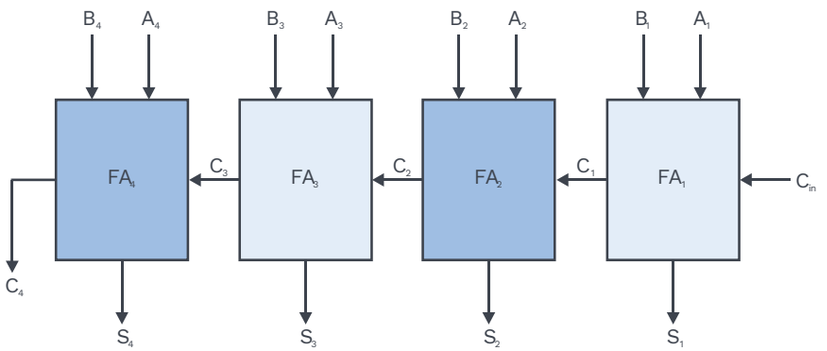
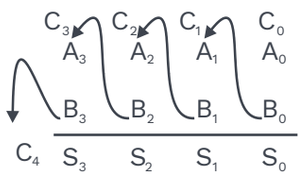


Fig. 2.11 Logic Diagram of a Binary Parallel Adder

The parallel adder in which carry-out of each full adder is the carry-in to the next most significant adder is called ripple carry adder. The greater the number of bits that a ripple carry adder must add, the greater is the time required to perform addition.

Carry look ahead adder:



$C_0 \leftarrow$ initial carry
$C_4 \leftarrow$ final carry

$$\boxed{C_1 = C_0(A_0 \oplus B_0) + A_0B_0}$$

↳(C_1 is generated if $\boxed{A_0 = 1, B_0 = 1}$

OR

$$\boxed{\begin{array}{c} A_0 = 1\ B_0 = 0 \\ \& \\ C_0 = 1 \end{array}}$$

OR

$$\boxed{\begin{array}{c} A_0 = 0\ B_0 = 1 \\ \& \\ C_0 = 1 \end{array}}\ )$$

$$\boxed{C_2 = A_1B_1 + C_1(A_1 \oplus B_1)} \quad \ldots(2)$$

$$\boxed{C_3 = A_2B_2 + C_2(A_2 \oplus B_2)} \quad \ldots(3)$$

$$\boxed{C_4 = A_3B_3 + C_3(A_3 \oplus B_3)} \quad \ldots(4)$$

In all 4 equations, each carry is dependent on the previous carry. If we perform back-substitution, put the value of C_1 in C_2 then C_2 will be dependent on C_0, which is available at the beginning.
We can continue with this process.
In this way, we can generate carry at each stage independently.

Let

$G_i = A_i B_i$ i = stage number

G = generator

$P_i = A_i \oplus B_i$ (it determines whether carry in stage i should be propagated to stage i + 1)

Rewriting the equations of carry in terms of G_i and P_i:

$C_1 = C_0P_0 + G_0$ ...(1)

$C_2 = C_1P_1 + G_1$ ...(2)

$C_3 = C_2P_2 + G_2$ ...(3)

$C_4 = C_3P_3 + G_3$ ...(4)

After substitution:

$C_1 = C_0P_0 + G_0$ ...(1)

$C_2 = C_0P_0P_1 + G_0P_1 + G_1$ ...(2)

$C_3 = (C_0P_0P_1 + G_0P_1 + G_1)\ P_2 + G_2$ ...(3)

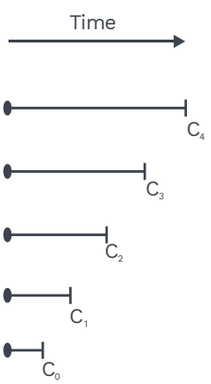
$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$ ...(4)

Note:

Carry is depandent G_i, P_i, C_0

Time complexity of carry lookahead adder:

(if we do not have gates of required fan-in)



All the carry will be generated parallelly, but there will be a lag due to the presence of more SOP (sum of product) terms as we move from C_1 to C_4

$$C_4 = C_0P_0P_1P_2P_3 + G_0P_1P_2P_3 + G_1P_2P_3 + G_2P_3 + G_3$$

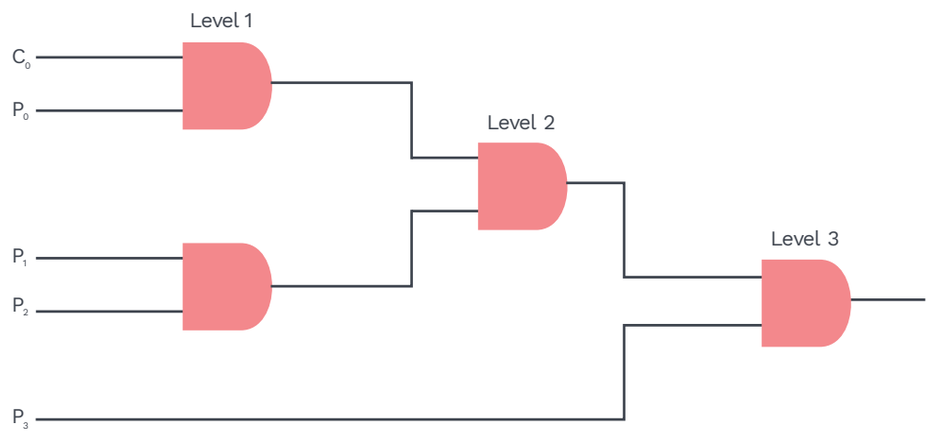


Fig. 2.12 Realization of Longest Term in Final Carry with Basic Gates of FAN-in 2

$T_{AND} \rightarrow$ Propagation delay of AND gate

$$\boxed{\text{Time complexity} = 3 \times T_{AND}}$$

Generalisation:

If there are "n+1" terms and we have AND gate of fan-in 'k', then $\left(\frac{n+1}{k^i}\right)$ is the number of gates required at i^{th} level.

$$(n+1) \rightarrow \left(\frac{n+1}{k}\right) \rightarrow \left(\frac{n+1}{k^2}\right) \rightarrow \ldots\ldots\ldots\ldots \frac{n+1}{k^i}$$

$\frac{n+1}{k^i} = 1$ $\quad i \leftarrow$ number of level.

$\Rightarrow n + 1 = k^i$

$$\boxed{\log_k (n+1) = i}$$

Time complexity $- \Theta(\log_k (n+1) \times T_{AND})$

Previous Years' Question

In a look-ahead carry generator, the carry generate function G_i and the carry propagate function P_i for inputs A_i and B_i are given by:
$P_i = A_i \oplus B_i$ and $G_i = A_iB_i$
The expressions for the sum bit C_{i+1} and the carry bit City of the look-ahead carry adder are given by:
$S_i = P_i \oplus C_i$ and $C_{i+1} = G_i + P_i C_i$, where C_0 is the input carry.
Consider a two-level logic implementation of the look-ahead carry generator. Assume that all P, and G are available for the carry generator circuit and that the AND and OR gates needed to implement the look-ahead carry generator for a 4-bit adder with S_3, S_2, S_1, S_0 and C_4 as its outputs are respectively:

1) 6, 3
2) 10, 4
3) 6, 4
4) 10, 5

Sol: 2) **(CS-2007)**

BCD adder:

Let's briefly review the process of BCD addition

1) For each digit in a decimal number, add its 4 bits BCD group.
2) If the addition result comes to be a number whose sum is less than 9, then that sum is in proper bcd form. No further updations are required.
3) But if the result of addition comes to be greater than 9, then simply add 0110 to the result to get the proper BCD. This will produce a carry to be added to the next decimal position.

Definition

A BCD adder circuit must be able to operate in accordance with the above steps. In other words, the circuit must be able to do the following
1) Add two 4 bit BCD code groups, using straight binary addition.
2) Determine if the sum of this addition is greater than '9' (1001), if it is then add 0110 (decimal 6) to this sum and generate carry for next decimal position.

SOLVED EXAMPLES

Q1 **Perform the following decimal addition in 8421 code.**
a) 25+13 **b) 679.6+536.8**

Sol: **a)**

```
  25   ⇒  0010  0101   (25 in BCD)
+ 13   ⇒  0001  0011   (13 in BCD)
------     ----------
  38       0011  1000
```

No carry, no illegal code.

b)

```
 679.6  ⇒ 0110  0111  1001  . 0110
+536.8  ⇒ 0101  0011  0110  . 1000
-------    ---------------------
 1216.4    1011  1010  1111  . 1110   ← All are illegal codes
           0110  0110  0110  . 0110
           ---------------------
          (1)0001 0000 0101  . 0100
           +1   +1   +1  +1
           ---------------------
           0001 0010 0001 0110 . 0100
             1    2    1    6    4
```

Sol: 1216.4

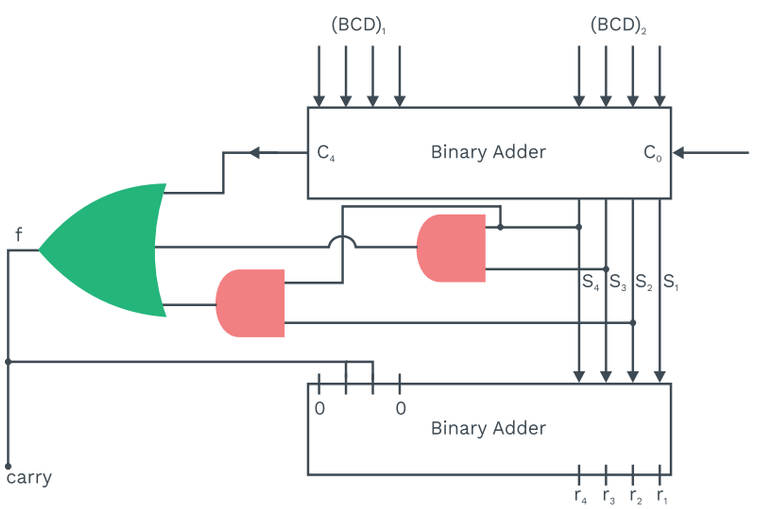


Fig. 2.13 Logic Diagram of BCD Adder

2's Complement adder:

For representing negative numbers and in the process of subtraction best way is to represent the numbers in the 2's compliment form. Both the addition and subtraction operation of signed numbers can be done only by using the addition operation if we use 2's compliment form to represent negative numbers.

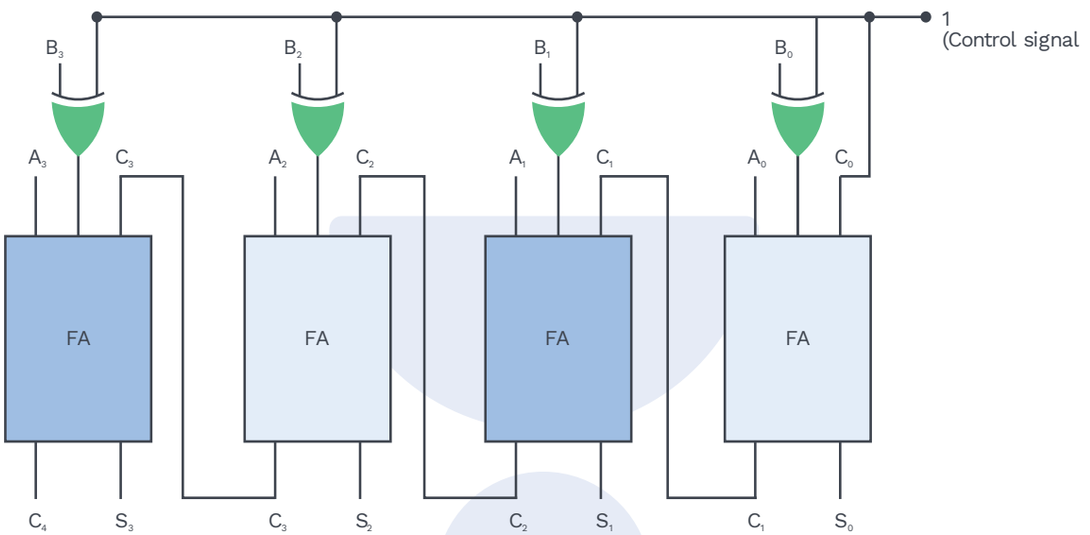


Fig. 2.14 2's Compliment Adder

Figure 2.14 shows a complete circuit that performs both addition and subtraction in 2's compliment. When the control signal is '1', the output of all the XOR gates will be $\overline{B}$ (complement of B), adding '1' to which will give 2's complement of B. So we connect the initial carry (C_0) to the control signal.

The circuit performs → A + 2's compliment of B = A - B

subtraction

Similarly, when the control signal is 0, the circuit behaves like a parallel adder.

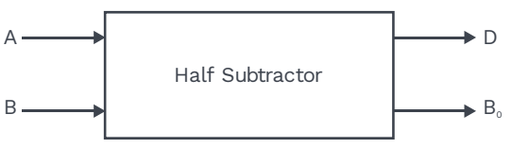
2.3 SUBTRACTOR

Half subtractor:

A half subtractor is a combinational circuit that performs the subtraction of one bit with the other and gives the result as a difference. There is an output to specify if a '1' has been borrowed. It subtracts the LSB of the subtrahend from the LSB of the minuend.

Input		Output	
A	B	D	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

a) Truth table



b) Block diagram

The output borrow B_0 is '0' as long as A ≥ B. It is 1 for A = 0 and B = 1.
The logical expression of difference and Borrow is as follows:

$$D = A\bar{B} + \bar{A}B = A \oplus B$$

$$B_0 = \bar{A}B$$

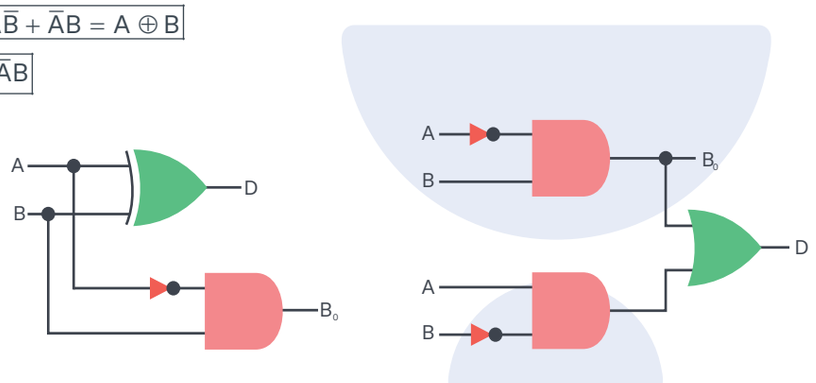


Fig. 2.15 Logic Diagrams of a Half Subtractor

Half subtractor Using NAND gate AND NOR gate:

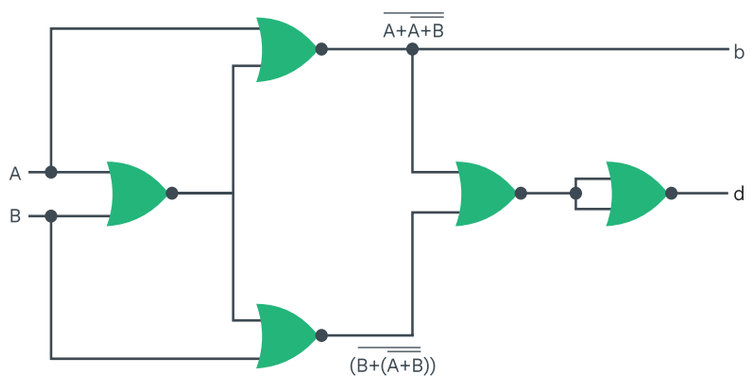


Fig. 2.16 Half-Subtractor using 2-Input NOR Gates

Logic diagram of half-subtractor using only 2-input NOR gates

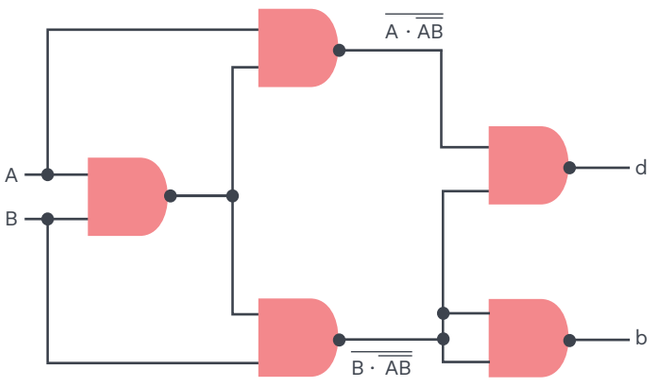


Fig. 2.17 half-Subtractor using 2-Input NAND Gates

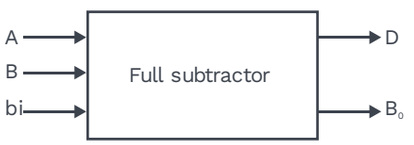
Logic diagram of a half-subtractor using only 2-input NAND gates

Full subtractor:

The half subtractor can perform only LSB subtraction. If, in result borrow is present, then this borrow is forwarded to the next higher bits.

Input			Difference	Borrow
A	**B**	**b_i**	**D**	**B_0**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

a) Truth table



b) Block diagram

From the truth table, a circuit that will give the correct difference and bits in response to every possible combination of A, B and bi is given as:

$$D = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + A\bar{B}\bar{b_i} + ABb_i$$

$$= b_i(AB + \bar{A}\bar{B}) + \bar{b_i}(A\bar{B} + \bar{A}B)$$

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

$$\boxed{D = A \oplus B \oplus b_i}$$

$$B_0 = \bar{A}\bar{B}b_i + \bar{A}B\bar{b_i} + \bar{A}Bb_i + ABb_i = \bar{A}B(b_i + \bar{b_i}) + (AB + \bar{A}\bar{B})b_i$$

$$\boxed{B_0 = \bar{A}B + (\overline{A \oplus B})b_i}$$

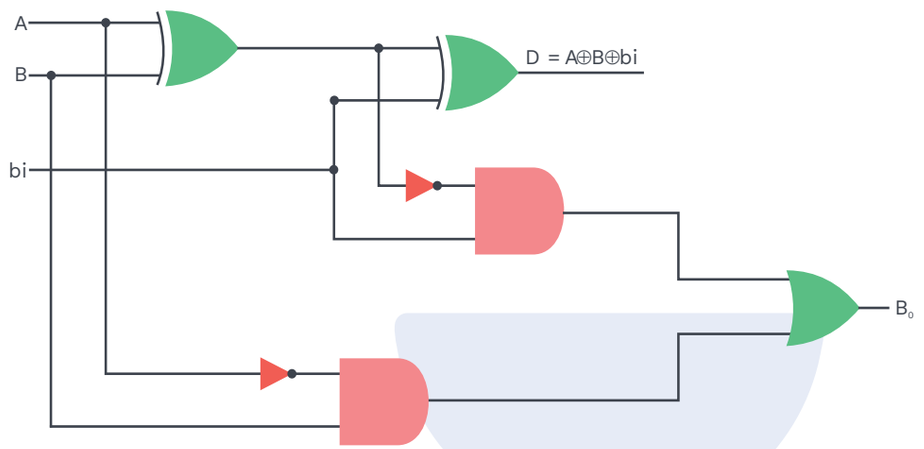


Fig. 2.18 Logic Diagram of a Full-Subtractor

Full subtractor using NAND Gate AND-NOR gate:

$D = A \oplus B \oplus \overline{b_i}$

$B_0 = \bar{A}B + (A \oplus B)b_i$

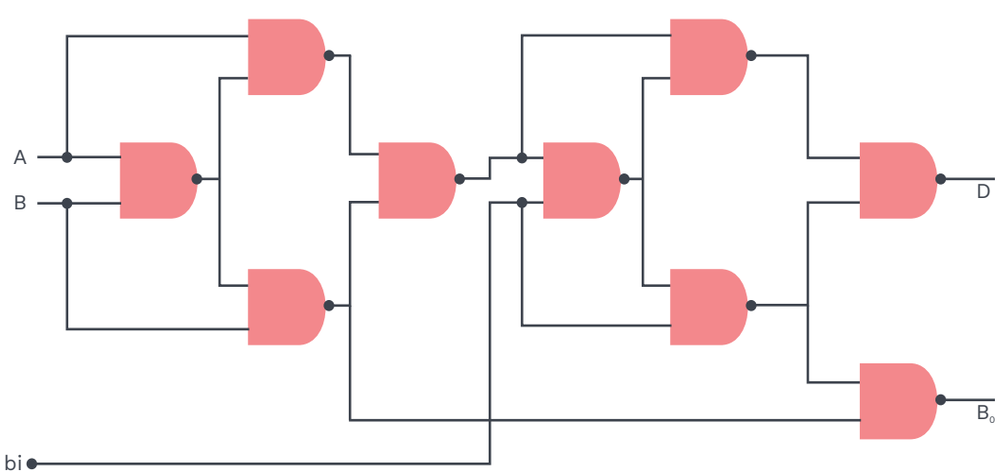


Fig. 2.19 Logic Diagram of a Full Subtractor using 2 Input NAND Gates.

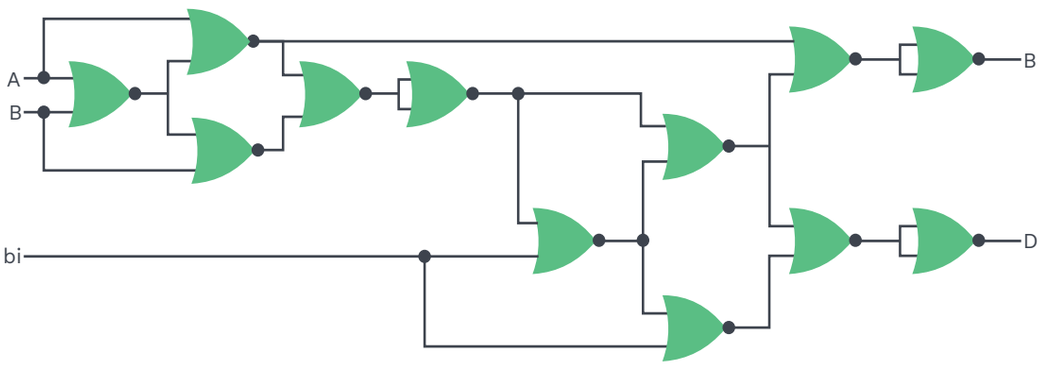


Fig. 2.20 Logic Diagram of a Full Subtractor using only 2-Input NOR Gates.

Parallel subtractor:

Complementation helps in carrying out subtraction in an easy manner. The subtraction A-B can be performed with the help of 2's complement of B and adding it to A. The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits.

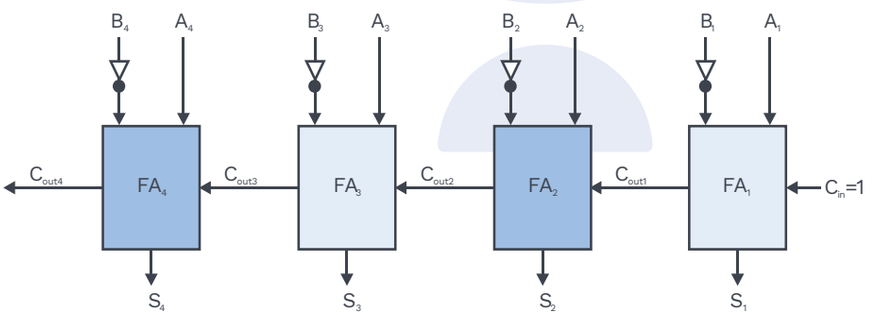


Fig. 2.21 Logic Diagram of a 4 Bit Parallel Subtractor

2.4 ENCODER

Introduction:

A digital encoder is a device whose inputs are decimal digits and whose output is the coded representation of those inputs.

Figure 2.21 shows a block diagram of an encoder with M inputs and N outputs. Here inputs are active high.

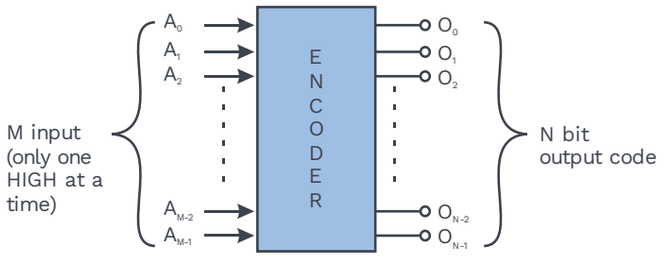


Fig. 2.22 Block Diagram of an Encoder

Octal to binary encoder:

An octal to a binary encoder (8 lines to 3 line encoder) accepts 8 input lines and produces a 3 bit output code corresponding to the particular input. From the truth table, we observe that A_2 is 1 if any of the digits D_4 or D_5 or D_6 or D_7 is 1.

Therefore,

$A_2 = D_4 + D_5 + D_6 + D_7$

Similarly,

$A_1 = D_2 + D_3 + D_6 + D_7$

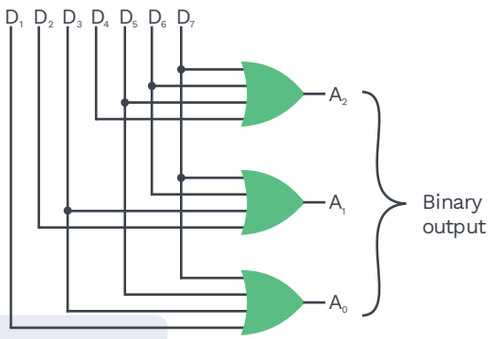
$A_0 = D_1 + D_3 + D_5 + D_7$

Note:

D_0 is not present in any of the expression, So D_0 is considered as "don't care".

Octal digits		Binary		
		A_2	A_1	A_0
D_0	0	0	0	0
D_1	1	0	0	1
D_2	2	0	1	0
D_3	3	0	1	1
D_4	4	1	0	0
D_5	5	1	0	1
D_6	6	1	1	0
D_7	7	1	1	1

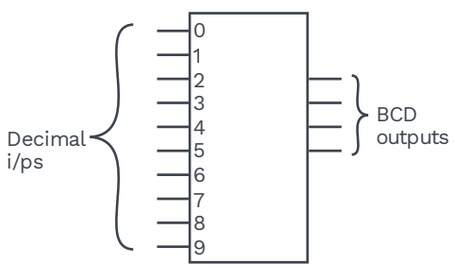
a) Truth table



b) Logic Diagram

Decimal to BCD encoder:

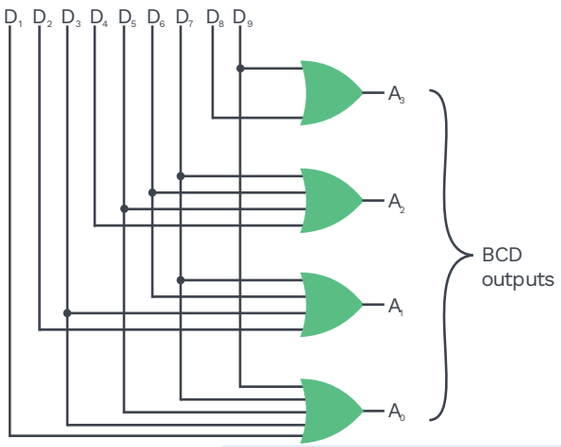
This type of encoder has 10 inputs – one for each decimal digit and 4 outputs that correspond to the BCD code.



a) Logic symbol

Decimal inputs		Binary			
		A_3	A_2	A_1	A_0
D_0	0	0	0	0	0
D_1	1	0	0	0	1
D_2	2	0	0	1	0
D_3	3	0	0	1	1
D_4	4	0	1	0	0
D_5	5	0	1	0	1
D_6	6	0	1	1	0
D_7	7	0	1	1	1
D_8	8	1	0	0	0
D_9	9	1	0	0	1

b) Truth table



c) Logic Design

Hexadecimal to binary encoder:

In hexadecimal number system, radix = 16

Range – 0 to F (15)

Total available numbers in base 16 = 16 = $2^m \Rightarrow m = \log_2 16 = \log_2 2^4 = 4$

$\therefore$ 4 o/p bits are required.

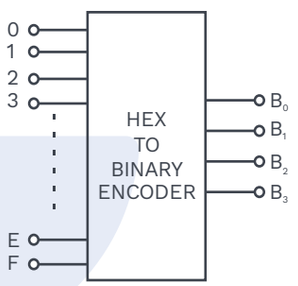
From Fig. 2.17

$B_0 = 1 + 3 + 5 + 7 + 9 + B + D + F$

$B_1 = 2 + 3 + 6 + 7 + A + B + E + F$

Similarly, we can find out B_2 & B_3 from the truth table.

I/p	O/p			
Hex	B_3	B_2	B_1	B_0
0	0	0	0	0
1	0	0	0	1
2	0	0	1	0
3	0	0	1	1
4	0	1	0	0
5	0	1	0	1
6	0	1	1	0
7	0	1	1	1
8	1	0	0	0
9	1	0	0	1
A	1	0	1	0
B	1	0	1	1
C	1	1	0	0
D	1	1	0	1
E	1	1	1	0
F	1	1	1	1



Priority encoder:

The encoders we discussed so far will operate correctly, provided that only one decimal input is high at any given time. In some practical systems, two or more decimal inputs may be HIGH at the same time.

A priority encoder is a logic circuit that responds to just one input in accordance with some priority system, among all those that may be simultaneously high.

The most common priority system is based on the relative magnitude of the inputs; whichever decimal input is largest is the one that is encoded.

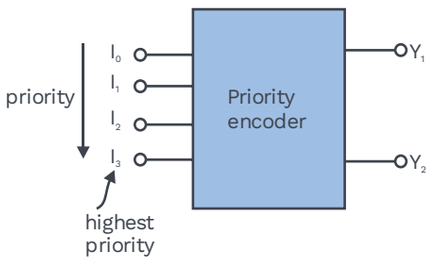


Fig. 2.23 Priority Encoder

Let's look at the truth table:

I_3	I_2	I_1	I_0	Y_1	Y_2
0	0	0	0	X	X
0	0	0	1	0	0
0	0	1	X	0	1
0	1	X	X	1	0
1	X	X	X	1	1

Table 2.3

X → don't care

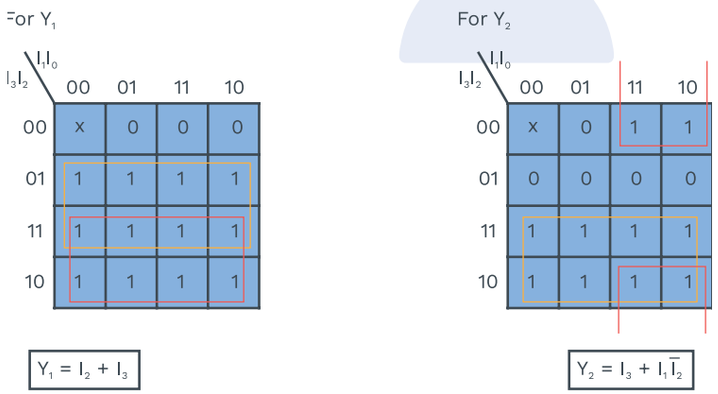


Fig. 2.24

2.5 DECODER

Introduction:

A decoder is a logic circuit that converts an N-bit binary input into M output lines such that only one output line is activated for each of the possible input combinations.

Figure 2.24 shows the general decoder diagram. Since each of the N inputs can be a 0 or 1, there are 2^N possible input combinations. For each of these input combinations, there are only one of the 'M' outputs that will be active (HIGH), all other outputs will remain inactive (LOW).

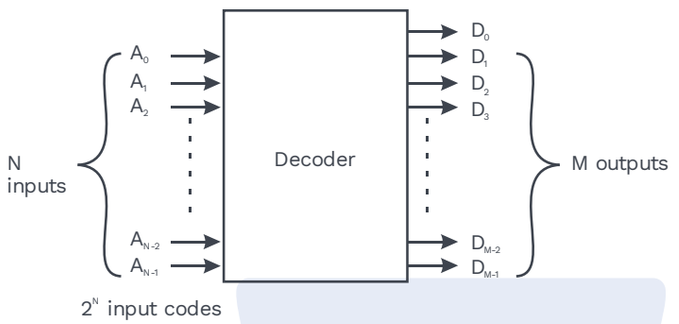


Fig. 2.25 General Block Diagram of Decoder

BCD to 7 segment decoder:

This type of decoder accepts BCD code and provides output to lighten seven segment display devices to produce a decimal read out.

Each segment is made up of a material that emits eight when the current is passed through it.

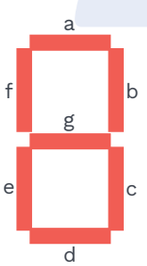


Fig. 2.26 Letters used to Designate the Segment

Decimal digit	8-4-2-1 BCD				Seven Segment Code						
	A3	A2	A1	A0	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	1	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	0	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	1	0	1	1

Table 2.4 Functional Table

We can find a simplified Boolean expression for each of the segments from the functional table.

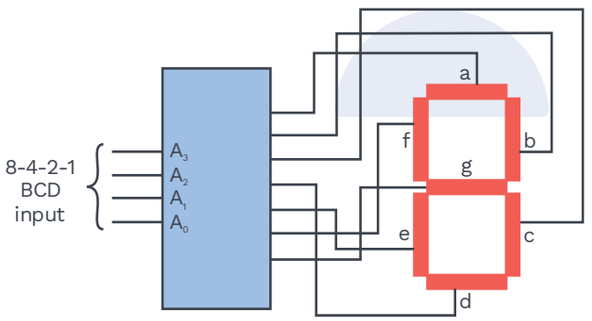
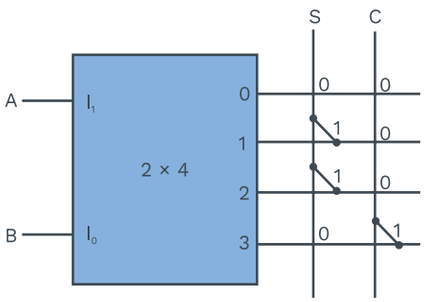


Fig. 2.26 Logic Circuit

Logic construction of ROM:

- ROM can be used to realize combinational functions by storing appropriate values at locations.
- Every ROM is expressed in terms of the ROM matrix and decoder.
- ROM matrix contains a set of links and connections. The lines entering the matrix and leaving are called connections and intersections between rows and columns are called links.



a)

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

b)

Fig. 2.27 Logic Diagram and Truth Table of ROM

Implementing function using only decoder:

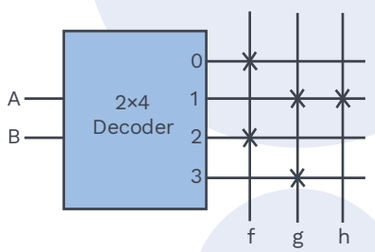


Fig. 2.28

Note:

The number of vertical lines gives the number of functions.

$f = \sum (0, 2)$
$g = \sum (1, 3)$
$h = \sum (1)$

Implementing function using decoder (ROM) and multiplexer:

Consider the function, $F = \sum (0, 1, 5, 7)$

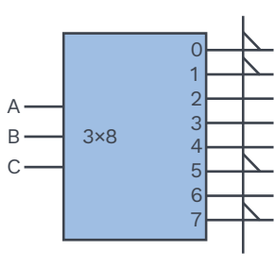


Fig. 2.29 Implementation of 'F' Using ROM

The above decoder (ROM) has → 8 + 1 connections
→ 1 Function
→ 8 × 1 = 8 Links
We can represent the same function using (2 × 4) decode & 2 × 1 MUX.

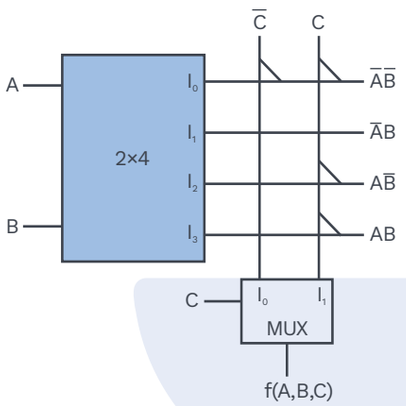


Fig. 2.30 Implementing Function Using ROM and Multiplexer

$F(A,B,C) = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{B}C + A\overline{B}C + ABC$

$\Rightarrow$ A = 0 B = 0 C = 0
$F = \overline{A}\overline{B}\overline{C}$

$\Rightarrow$ A = 0 B = 0 C = 1
$F = \overline{A}\overline{B}C$

The above combinational circuit has → (4+2) connection
→ 4 × 2 = 8 links
For a 10 variables function and for a given set of decoder and MUX, let's find how many 'connections' and 'Links' exist.

Decoder	MUX	Connections	Links
10	0	1024+1	1024
5	5	32+32=64	$2^5 \times 2^5 = 1024$
4	6	16+64=80	$2^4 \times 2^6 = 1024$
3	7	8+128=136	1024

Table 2.5

Note:

Beyond a certain limit number of connections starts increasing.

Rack Your Brain

Will always adding decoder and mux decrease number of connections?

Decoder with enable:

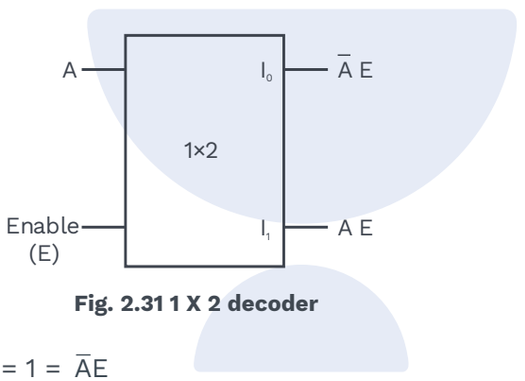
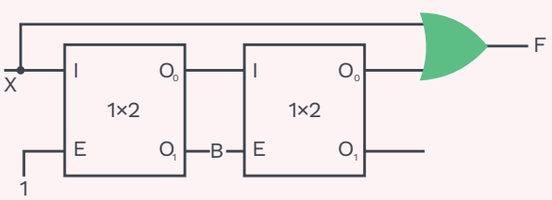


Fig. 2.31 1 X 2 decoder

E = 1 and A = 0 then $I_0 = 1 = \bar{A}E$

E = 1 and A = 1 then $I_1 = 1 = AE$

Rack Your Brain

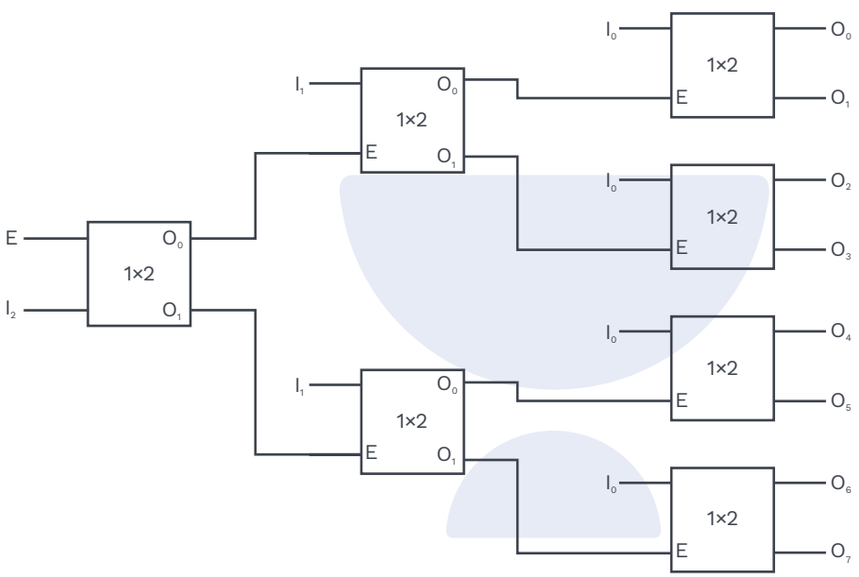


Find the Boolean expression for F.

SOLVED EXAMPLES

Q1 **Construct 3×8 decoder using 1×2 decoder.**

Sol:



Expansion of decoder in general

Q2 **Construct m×2^m decoder using n×2^n**

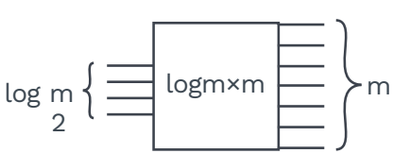
Sol: **n×2^n decoder**

1 device → "n" lines

1 line → $\frac{1}{n}$ device (not feasible, just for the sake of calculation)

Level 1

Let's say,



$m \text{ lines} \rightarrow \frac{m}{n} \text{ devices}$

Level 2

$\left(\frac{m}{n}\right) \text{lines} \rightarrow \left(\frac{1}{n} \times \frac{m}{n}\right) \text{devices}$

Level 3

$\frac{m}{n^2} \text{lines} \rightarrow \left(\frac{m}{n^3}\right) \text{devices}$

$\vdots \qquad \vdots$

Level K $\qquad \vdots$

$$\frac{m}{n^k} \approx 1$$

$$\frac{m}{n^k} \leq 1$$

$$\boxed{k \geq \left\lceil \left(\frac{\log_2 m}{\log_2 n} \right) \right\rceil}$$

Note:

The total number of decoders required for 'k' levels:

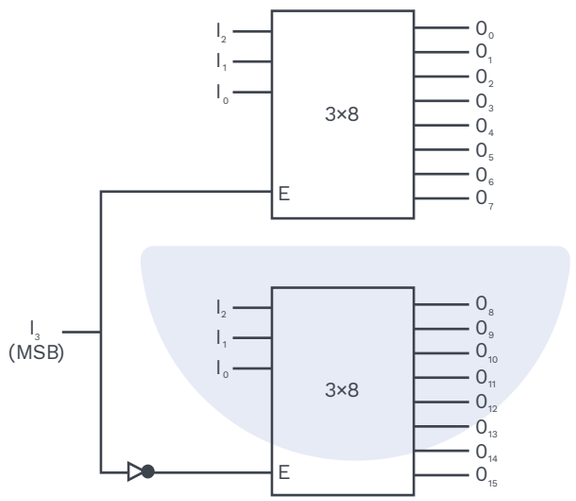
$$\left(\sum_{k=1}^{\frac{\log m}{\log n}} \frac{m}{n^k} \right)$$

Rack Your Brain

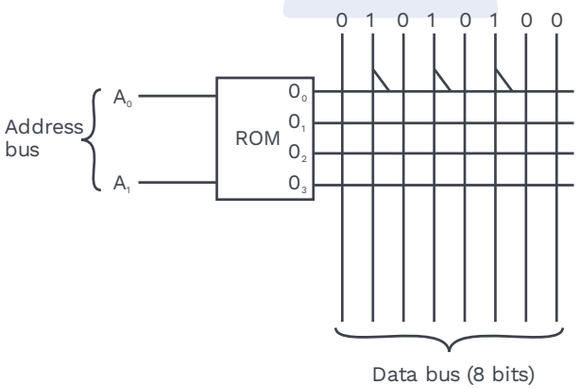
How many levels are required to construct a 7×128 decoder using a 1×2 decoder? Also, find the total number of decoders required.

Q3 **Using 2 (3×8) decoder construct 4×16 decoder.**

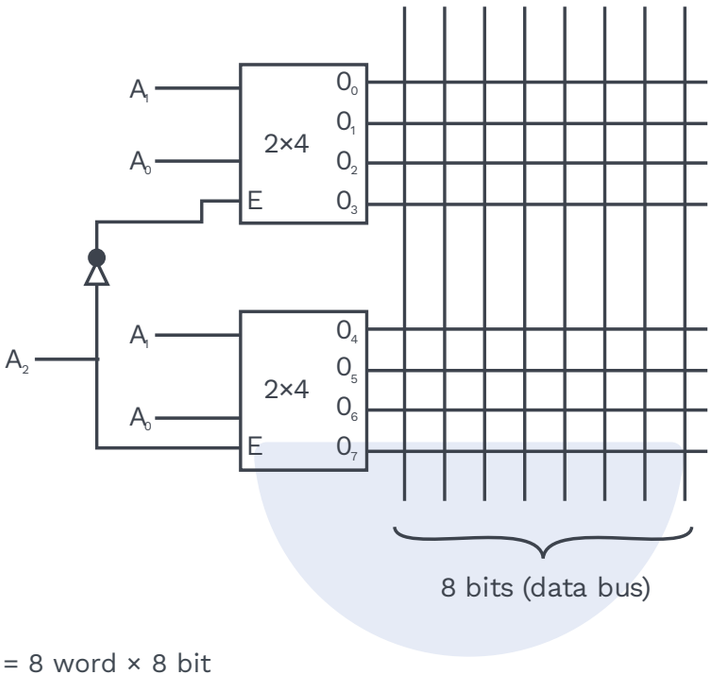
Sol:



Address expansion of ROM:

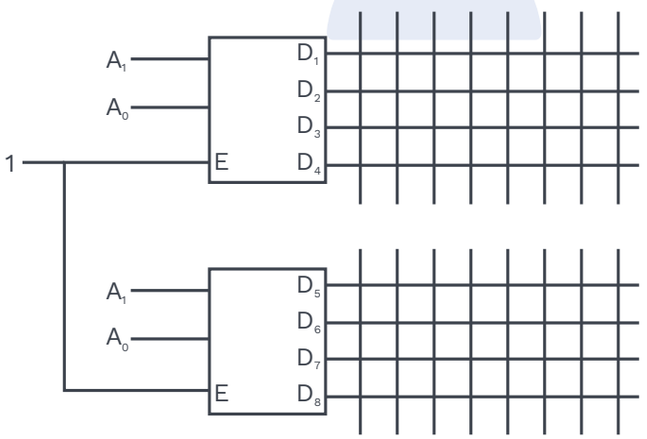


If we give A_0 =0, A_1 = 0 in the address, then we will get the data stored in 'O_0'
O_0 will 0 1 0 1 0 1 0 0
Total size = 4 words × 8 bits.
After expanding the address, i.e. increasing the number of words that can be stored (the size of each word is the same)



Total size = 8 word × 8 bit

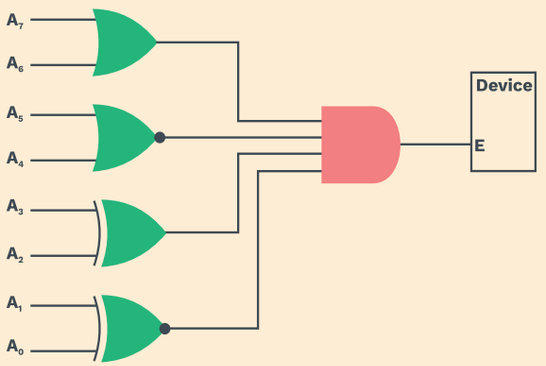
Word expansion of ROM:



When $A_1 = 0$, $A_0 = 0$, D_1 and D_5 are going to get selected.
(overall, we'll have 8+8 = 16 bits)
Size of word = 16 bits
Number of words = 4

SOLVED EXAMPLES

Q1 **Consider a device that is enabled using 8 address bits as shown below. For how many address the device is enabled:**



Sol:

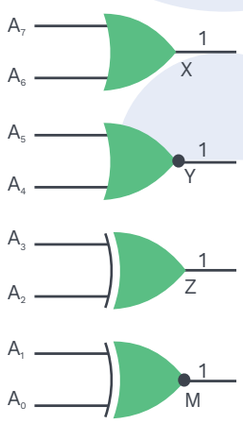


Fig. 2.32

'X' can be made '1' in 3 ways:

$A_7 = 1 \quad A_6 = 0$

$A_7 = 0 \quad A_6 = 1$

$A_7 = 1 \quad A_6 = 1$

Similarly,

Y → 1 way

Z → 2 ways

M → 2 ways

Total ways = 3 × 2 × 2 = 12 ways.

2.6 DEMULTIPLEXER

Definition

A multiplexer takes several inputs and transmits one of them to the output. A demultiplexer performs the reverse operations of multiplexer. It takes a single input and distributes it over many outputs. Basically Demultiplexer can be thought of a distributor, since it transmits same data to different destinations.

Note:

Demultiplexer takes one input and selectively distributes it to 1-of-N output channel like a multi position switch.

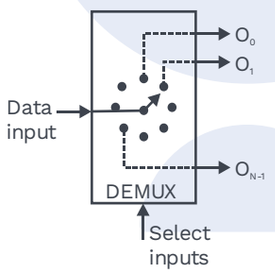
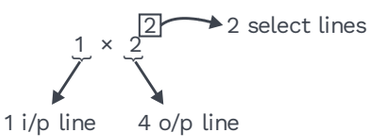
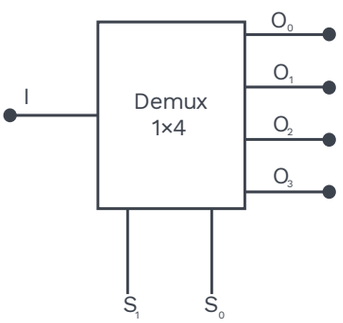


Fig. 2.33 Functional Diagram of a General Demultiplexer.

Let's look at (1×2^2) demultiplexer,





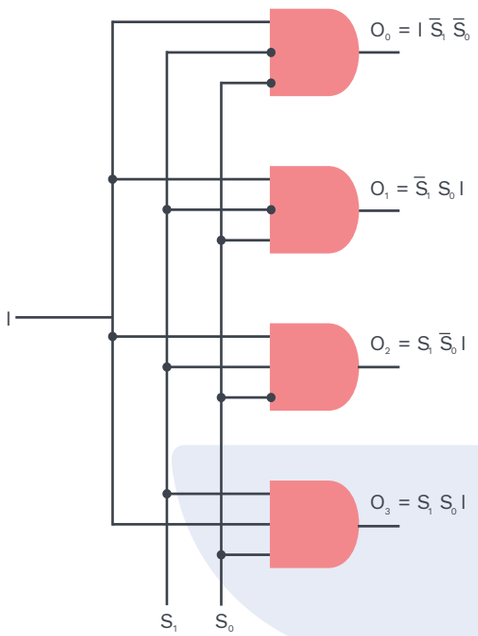


Fig. 2.34 Logic Circuit Diagram of 1×4 DeMUX

S_1	S_0	O_0	O_1	O_2	O_3
0	0	I	0	0	0
0	1	0	I	0	0
1	0	0	0	I	0
1	1	0	0	0	I

Table 2.6 Truth Table for 1×4 Demux

2.7 MULTIPLEXERS

Definition:

Multiplexing means sharing. A common example of multiplexing or sharing occurs when several hardware devices share a single transmission bus to communicate with a computer.

A multiplexer (MUX) is a logic circuit that accepts several data inputs and allows only one of them at a time to get through the output. The routing of the data input to the output is controlled by select lines.

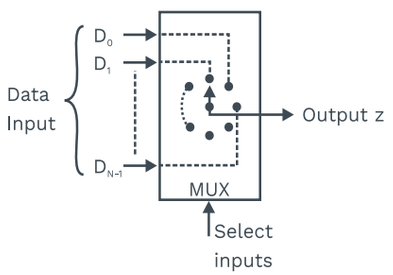
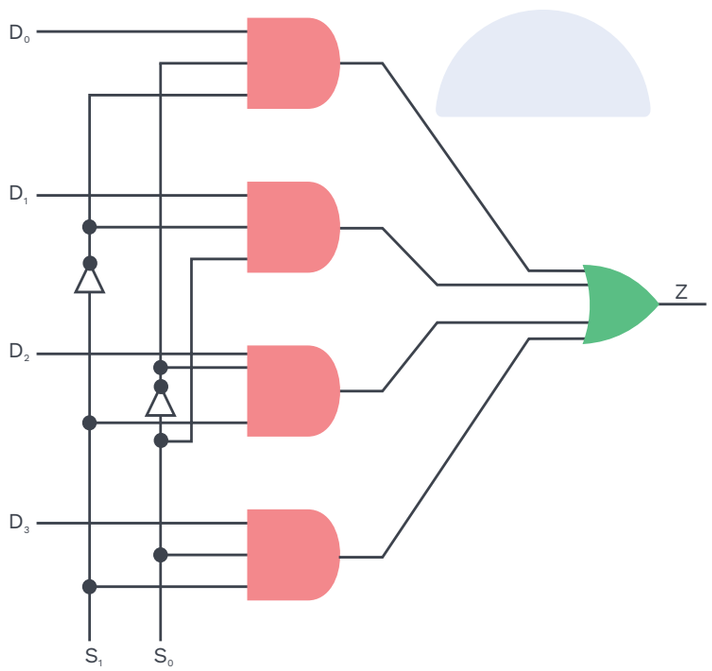


Fig. 2.35 Functional Diagram of Digital Multiplexer

The 4-input multiplexer:

Fig. 2.27 a) shows the logic circularly for a 4 input multiplexer with data inputs D0, D1, D2, and D3 and data select inputs S0 and S1. The logic levels applied to the S0 and S1 inputs determine which AND gate is enabled so that its data input passes through the OR gate to the output. The function table at Fig. 2.27 b) gives the output for the input select codes as

$$Z = \overline{S_1}\,\overline{S_0}D_0 + \overline{S_1}S_0D_1 + S_1\overline{S_0}D_2 + S_1S_0D_3$$



a) Logic Diagram

Select inputs		o/p
S_1	S_0	Z
0	0	D_0
0	1	D_1
1	0	D_2
1	1	D_3

b) Function Table

MUX as universal logic gates:

Implementation of NAND using 2:1 MUX

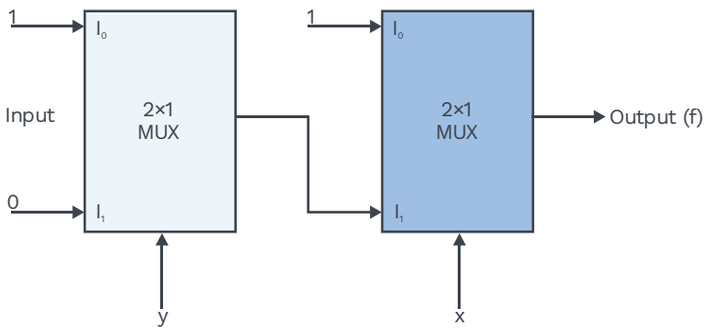


Fig. 2.36 Implementation of NAND Using MUX

The First multiplexer will act as NOT gate, which will produce complemented input to the second multiplexer.

Implementation of NOR gate using 2:1 MUX

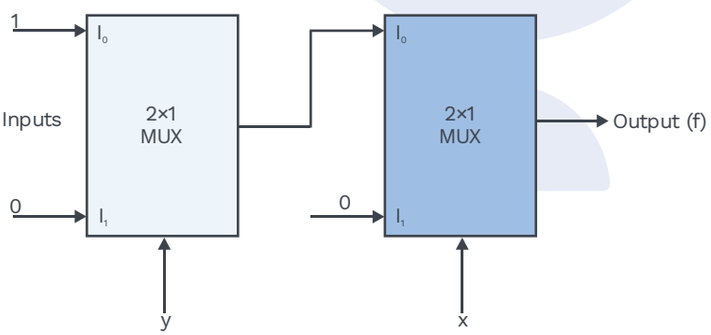
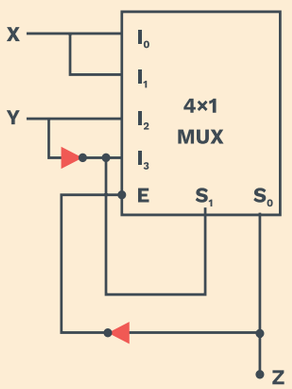


Fig. 2.37 Implementation of NOR Gate Using MUX

SOLVED EXAMPLES

Q1



Sol: $f = E(I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0)$

$f = \bar{Z}(XY\bar{Z} + XYZ + Y\bar{Y}\bar{Z} + \bar{Y}\bar{Y}Z)$

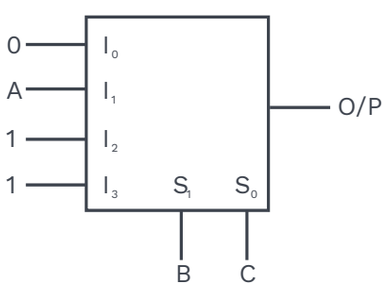
$f = \bar{Z}(XY + \bar{Y}Z)$

$f = XY\bar{Z}$ **Sol:**

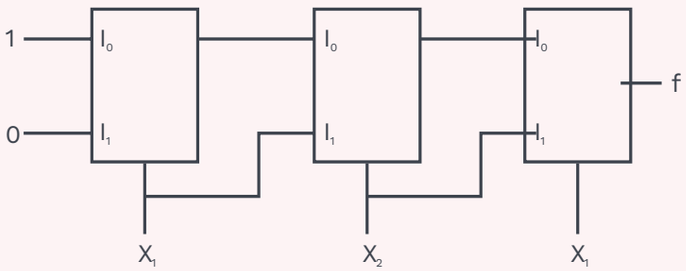
Q2 **F(A, B, C) = Σ(2, 3, 5, 6, 7)**
S_1 = B, S_0 = C
Implement using 4x1 MUX.

Sol: $F = \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$

$F = 0(\bar{B}\bar{C}) + A(\bar{B}C) + 1(B\bar{C}) + 1(BC)$



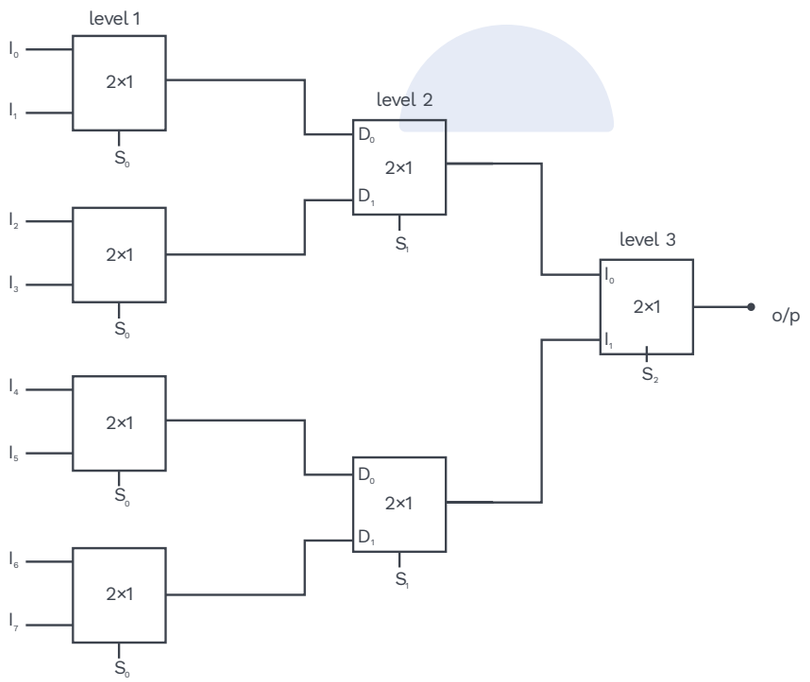
Rack Your Brain



Find 'f'.

Q3 **Construct t×1 MUX using 2×1**

Sol:



When $S_2S_1S_0$ = 011, I_1, I_3, I_5, and I_7 are selected at level 1, D_1, D_1 will be selected at level 2, I_0 will be selected at level 3.

∴ I_3 will appear at the output.

M×1 MUX using N×1 MUX:

In N×1 MUX: 'N' lines $\rightarrow$ 1 device

1 line $\rightarrow \frac{1}{N}$ device

For M lines, $\frac{M}{N}$ devices in 1st level

For $\frac{M}{N}$ lines, $\frac{M}{N} \times \frac{1}{N} = \frac{M}{N^2}$ devices in 2nd level

$\vdots$

$\vdots$

$\vdots$

$\frac{M}{N^K} \leq 1$ at K^{th} level.

$N^K \geq M$

$\boxed{K \geq \log_N M}$ $K \rightarrow$ No. of level

Total no. of devices - $\boxed{D = \sum_{K=1}^{\lceil \log_N M \rceil} \left(\frac{M}{N^K}\right)}$

Applications:

Logic function generator:

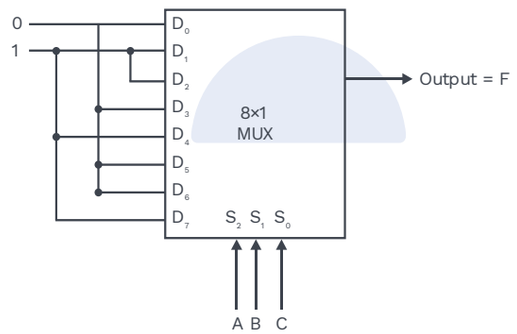
SOLVED EXAMPLES

Q1 **Use a multiplexer to implement the logic function F = A ⊕ B ⊕ C**

Sol: Since F = 1 when ABC = 001, 010, 100 and 111, we connect logic 1 to data inputs D_1, D_2, D_4 and D_7. Logic 0 is connected to other data inputs D_0, D_3, D_5 and D_6. When the data select inputs are any of the combinations for which F = 1, the output will be 1.

S_2	S_1	S_0	F = A⊕B⊕C
A	B	C	
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

Table 2.7



Previous Years' Question

Suppose only one multiplexer and one inverter are allowed to be used to implement any Boolean function of n variables. What is the minimum size of the multiplexer needed?

1) 2^n line to 1 line

2) 2^{1+1} line to 1 line

3) 2^{n-1} line to 1 line

4) 2^{n-1} line to 1 line

Sol: 3) **(CS-2007)**

2.8 COMPARATORS

A comparator is a logic circuit used to compare the magnitudes of two binary numbers. Depending on the design, it may either simply provide an output that is active when the two numbers are equal or provide outputs that signify which of the numbers is greater when equality does not hold.

Note:

The X-NOR gate is a basic comparator, because it outputs a 1 only if it's two input bits are equal.

2-bit comparator:

Let two numbers be $\overbrace{(A_2A_1)}^{A}$ and $\overbrace{(B_2B_1)}^{B}$ where A_1 and B_1 are the least significant bits of two numbers.

Case (i) A = B

∵ X NOR gate is called equality detector.

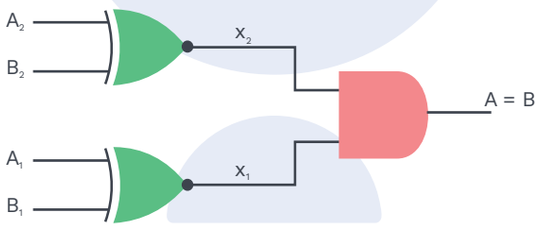


Fig. 2.38 "A = B" Comparator

If $x_1 = 1$ and $x_2 = 1$, o/p is 1

$$\boxed{x_1.x_2 = 1}$$

Case (ii) A > B

If $(A_2 > B_2)$ OR $(A_2 = B_2$ & $A_1 > B_1)$, o/p is 1

$A_2 = 1$ $A_1 = 1$

$B_2 = 0$ $B_1 = 0$

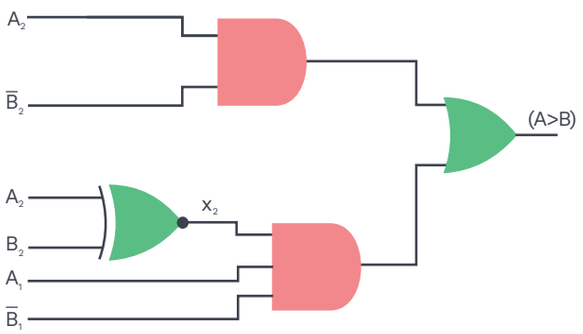


Fig. 2.39 "A>B" Comparator

For A > B, $\boxed{A_2\bar{B}_2 + x_2.(A_1\bar{B}_1)} = 1$

Case (iii) A < B

If (A2 < B2) OR ($A_2 = B_2$ & $A_1 < B_1$)

$A_2 = 0$ $A_1 = 0$

$B_2 = 1$ $B_1 = 1$

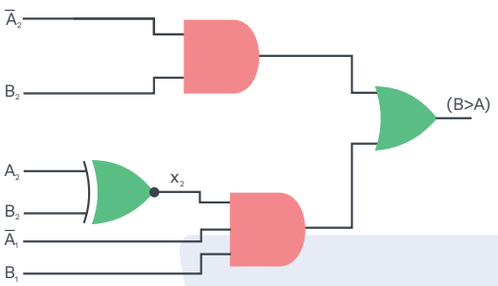
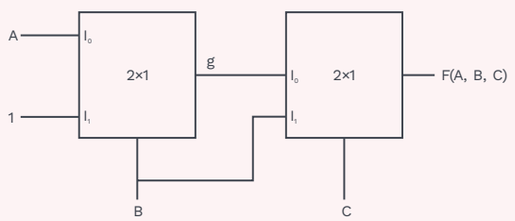


Fig. 2.40 "A<B" Comparator

For B > A, $\boxed{\bar{A}_2B_2 + x_2.(\bar{A}_1B_1)} = 1$

Rack Your Brain



What is the function 'F' in Canonical SOP?

2.9 HAZARDS

Hazards are unwanted switching transients that may appear at the output of a circuit because different paths have different propagation delays. Such a transient is also called a glitch or a spurious spike, which is caused by the hazardous behaviour of a logic circuit. Hazards occur in combinational circuits as well as in sequential circuits.

The permanent hazards are resulted due to open circuits or short circuits of the connecting lead. (terminals)

Note:

The hazards can be "stuck at 0" (s-a-0) or "stuck at 1" (s-a-1)

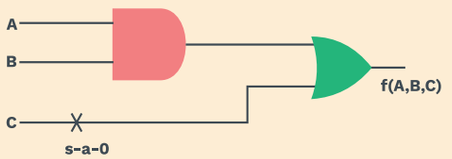
The single fault analysis is made using the “path sensitization” technique. This technique provides a test vector.

Procedure for finding the test vectors:

- Inactive all the other paths except the tested path.
  a) For “AND” or “NAND”, apply logic ‘1’ for inactivation.
  b) For “OR” or “NOR”, apply logic ‘0’ for inactivation.
- Apply the opposite logic value at the tested place.
- The i/p combinations satisfying the above requirements form the test vector.

SOLVED EXAMPLES

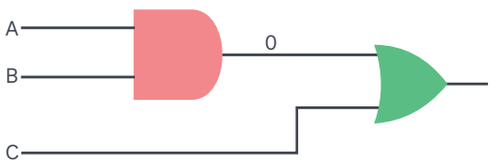
Q1



Is ‘c’ at s-a-0?

Sol:

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



If for all 3 i/ps, o/p is 1, then c is not s-a-0

Chapter Summary

- Half adder: 2 binary inputs and 2 binary outputs.
  $S = P \oplus Q$
  $C = PQ$
- Full adder: 3 binary inputs and 2 binary outputs.
  $S = P \oplus Q \oplus R$
  $C = R(A \oplus B) + AB$
- Carry look ahead adder:
  Carry is generated at each stage independently.
  Carry is dependent on generator (G_i), propagation (P_i) and initial carry (C_0)
- Parallel adder: Output carry from each full-adder is connected to the input carry of the next full adder.
- BCD adder: Adds two 4 bits BCD code groups using straight binary addition.
- Half subtractor: Difference $= A \oplus B$
  Borrow $= \overline{A}B$
- Full subtractor: Difference $= A \oplus B \oplus C$
- Borrow $= \overline{A}B + (A \oplus B)C$
- Encoder: Outputs are coded representations of the inputs.
- Priority encoder: It responds to just one input in accordance with some priority system.
- Decoder: Only one output line is activated for each possible combination of the input
- Demultiplexer: It takes one input data source and selectively distributes it to 1-of-N output channels
- Multiplexer: It accepts several data inputs and allows one of them at a time to get through the output.
- Hazards: Stuck-at-0
  Stuck-at-1

3 Sequential Circuit

3.1 BASICS OF SEQUENTIAL CIRCUITS

In a sequential circuit, output at any instant is dependent on the present input as well as the previous state of the circuit.

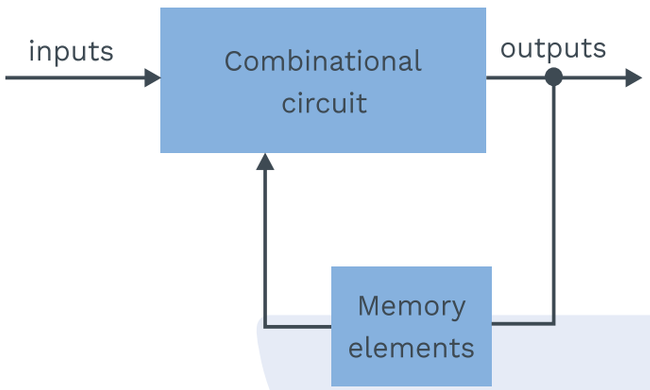


Fig. 3.1 Block Diagram of a Sequential Circuit

Combinational circuits	Sequential circuits
1) The output at an instant depends only on the present input.	**1)** Output at an instant of time is dependent on the present input as well as the previous state of the circuit.
2) Memory units are not required in combinational circuits.	**2)** Memory units are required to store the past history of the system.
3) Combinational circuits are faster than sequential circuit as delay exists between the input and the output, which is due to the propagation delay of the logic gates.	**3)** Sequential circuits are slower than combinational circuits.

Table 1.1 Comparison Between Combinational and Sequential Circuit

Synchronous sequential circuit:

- In synchronous circuits, clocked flip-flops are the memory elements.
- The change in input signals affects memory elements upon activation of a clock signal.
- Operating speed of the clock is directly proportional to the time required by the single memory element in the circuit.

Asynchronous sequential circuit:

- Memory elements of the asynchronous circuits are either unclocked Flip-flops (external clock is not given, the output of flip-flops is given as clock to next flip-flop) or time delay units.
- In asynchronous circuits, any changes in the input signals may affect memory elements at any point of time.

3.2 FLIP-FLOP

Latch:

Definition

The term Latch is used for certain flip-flops. It refers to non-clocked flip-flops because these flip-flops 'Latch on' to '1' or '0' immediately upon receiving the inputs pulse called SET or RESET. They are not dependent on clock signal for their operation. Clocked flip-flops are latches which respond to the inputs and latch on to 1 or 0 only when they are enabled.
On the other hand a flip-flop is a sequential device that normally samples it's inputs and changes it's outputs only at times determined by clock pulses.

S-R Flip-flop:

Figure 3.2 a) shows a logic diagram of edge triggered S-R flip-flop and Figure 3.2 b) shows a logic symbol of SR flip-flop. The characteristic table and excitation table of S-R flip-flop are shown in Figure 3.2 c) and Figure 3.2 d), respectively.

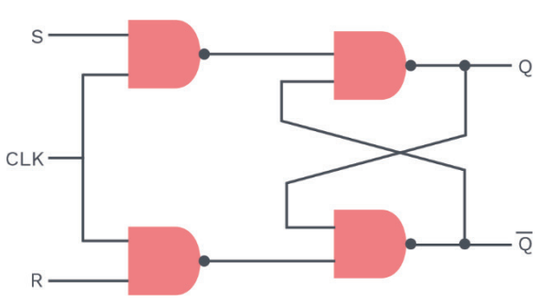


Fig. 3.2 a) Logic Diagram

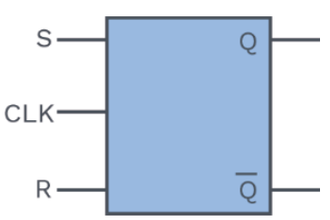


Fig. 3.2 b) Logic Symbol

CLK	S	R	Q_N	Q_{N+1}	
↑	0	0	0	0	Latch Mode
↑	0	0	1	1	
↑	0	1	0	0	Reset
↑	0	1	1	0	
↑	1	0	0	1	SET
↑	1	0	1	1	
↑	1	1	0	x	Invalid
↑	1	1	1	x	

Table 1.2 c) Characteristic Table

Q	Q_{N+1}	S	R
0	0	0	x
0	1	1	0
1	0	0	1
1	1	x	0

Table 1.3 d) Excitation Table

Characteristic equation: $Q_{n+1} = S + Q_n R'$

J-K flip-flop:

J-K flip-flop is similar to the S-R flip-flop; the only difference is that it does not have any invalid state like in S-R flip-flop.

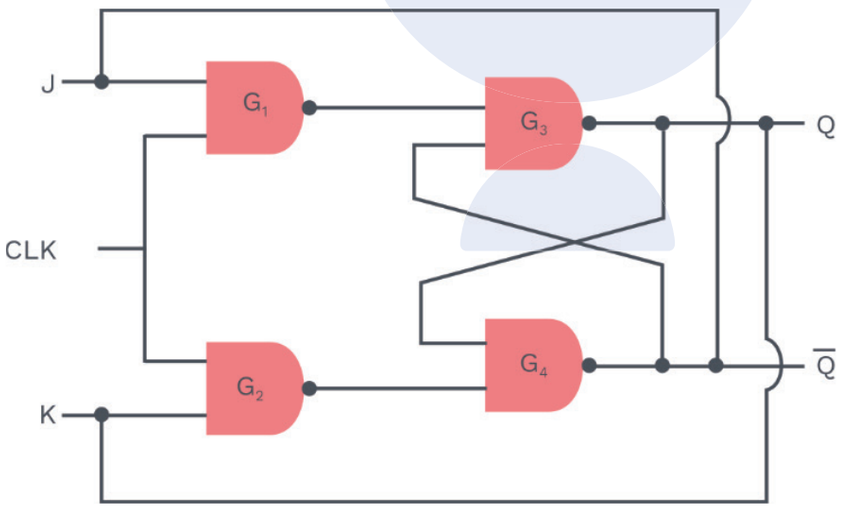


Fig. 3.3 Logic Diagram

Function table:

J	K	Q_{N+1}	
0	0	Q_N	→ Latch/Hold
0	1	0	→ Reset
1	0	1	→ Set
1	1	$\overline{Q}_N$	→ Toggle

Table 1.2 Functional Table of JK Flip-Flop

Characteristic table:

J	K	Q_N	Q_{N+1}	
0	0	0	0	Latch
0	0	1	1	
0	1	0	0	Reset
0	1	1	0	
1	0	0	1	Set
1	0	1	1	
1	1	0	1	Toggle
1	1	1	0	

Table 1.3 Characteristic Table of JK Flip-Flop

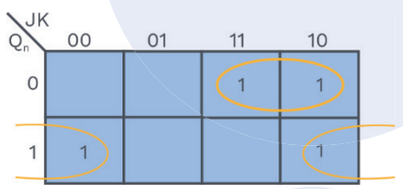


Fig. 3.4 K-map Simplification of JK Flip-flop

2 Prime implicants
2 essential prime implicants

$$Q_{n+1} = J\overline{Q}_n + \overline{K}\,Q_n \longrightarrow \text{Characteristic Equation}$$

Q_{n+1} ⟶ Next state

Q_n ⟶ Present state

Excitation table:

Q_n	Q_{n+1}	J	K
0	0	0	ϕ
0	1	1	ϕ
1	0	ϕ	1
1	1	ϕ	0

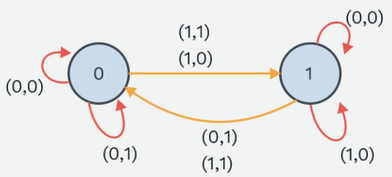
Table 1.4 Excitation Table of JK Flip-Flop

$\phi \rightarrow$ Don't care.

Grey Matter Alert!

As Flip–Flop is 1 bit memory, it can either store 0 or 1

Note: (0, 1) implies J = 0, K = 1



State Diagram of J–K Flip–Flop

D-Flip-flop:

The edge-triggered D flip-flop has one input. D-flip-flop can be obtained from a J-K flip-flop by placing an inverter between the J and K terminal.

Function table:

D	Q_{n+1}
0	0
1	1

Table 1.5 Functional Table of D Flip-Flop

Characteristic table:

D	Q_n	Q_{n+1}
0	0	0
0	1	0
1	0	1
1	1	1

Table 1.6 Characteristic Table of D Flip-Flop

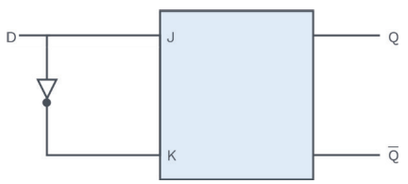


Fig. 3.5 Representation of D Flip-Flop Using JK Flip-Flop

Note:

From the characteristic table

$Q_{n+1} = D\overline{Q}_n + D\overline{Q}_n$

$Q_{n+1} = D(Q_n + \overline{Q}_n) = D$ = Characteristic Equation

Excitation table:

Q_n	Q_{n+1}	D
0	0	0
0	1	1
1	0	0
1	1	1

Table 1.7 Excitation Table of D Flip-Flop

State diagram:

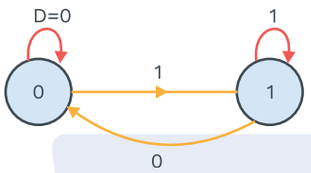


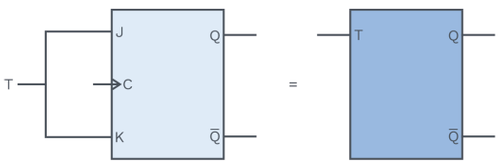
Fig. 3.6 State Diagram of D Flip-Flop

T flip-flop:

A T flip-flop has single control-input, labelled T for Toggle. When T is '1', the flip-flop toggles on every clock pulse; when T is 0, the flip-flop remains in the same state.

Grey Matter Alert!

It is easy to convert a J-K flip-flop to the functional equivalence of a T flip-flop by just connecting J and K together and labelling common connection as 'T'.



(a) T Flip–flop from J–K flip flop

(b) Logic Symbol

C	T	Q_N	Q_{N+1}	State
↑	0	0	0	No change
↑	0	1	1	
↑	1	0	1	Toggle
↑	1	1	0	
0	x	0	0	No change
0	x	1	1	

(c) Truth table

Fig. 3.7 Edge Triggered T Flip-Flop

Function table:

T	Q_{n+1}
0	Q_n
1	$\overline{Q_n}$

Table 1.9 Functional Table of T Flip-Flop

Characteristic table:

T Q_n	Q_{n+1}
0 0	0
0 1	1
1 0	1
1 1	0

Table 1.10 Characteristic Table of T Flip-Flop

Characteristic equation:

$$Q_{n+1} = \overline{T}\,Q_n + T\,\overline{Q_n}$$

$$\boxed{Q_{n+1} = T \oplus Q_n}$$

Excitation table:

$Q_n \rightarrow Q_{n+1}$	T
0 0	0
0 1	1
1 0	1
1 1	0

Table 1.11 Excitation Table of T Flip-Flop

State diagram:

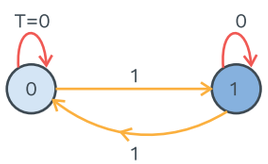


Fig. 3.8 State Diagram of T Flip-Flop

Triggering:

Definition

The momentary change in control of the latch or flip-flop to switch it from one state to the other is called a trigger, and the transition it causes is said to trigger the flip-flop. The process of applying the control of applying the control signal to change the state of Flip-flop is called triggering.

There are two types of triggering:

i) Level triggering

ii) Edge triggering

In level triggering, flip-flop changes its state when the clock is at logic 1 level.

In edge triggering, the input signals affect the Flip-flop if they are present at the positive going or negative going edge of the clock pulse.

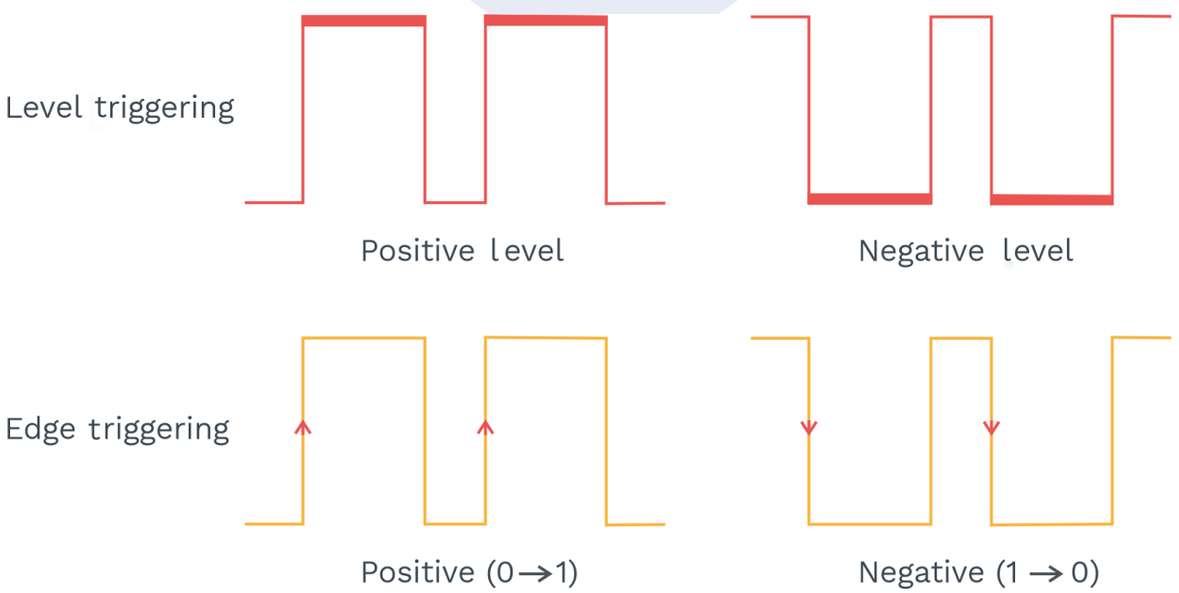


Fig. 3.9 Waveform of Clock Pulse

Race around condition:

For the J-K flip-flop shown in Figure 3.5, consider the assignment of excitations J=K=1. If the clock pulse is high for too long, the output of the flip-flop will keep fluctuating between '0' and '1', and at the end of the clock, its state will become uncertain. This is known as a race around the condition.

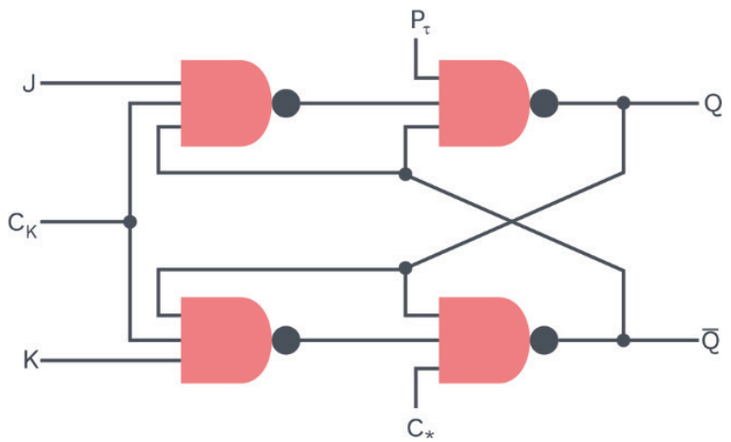


Fig.3.10 Level Triggering Flip-Flop

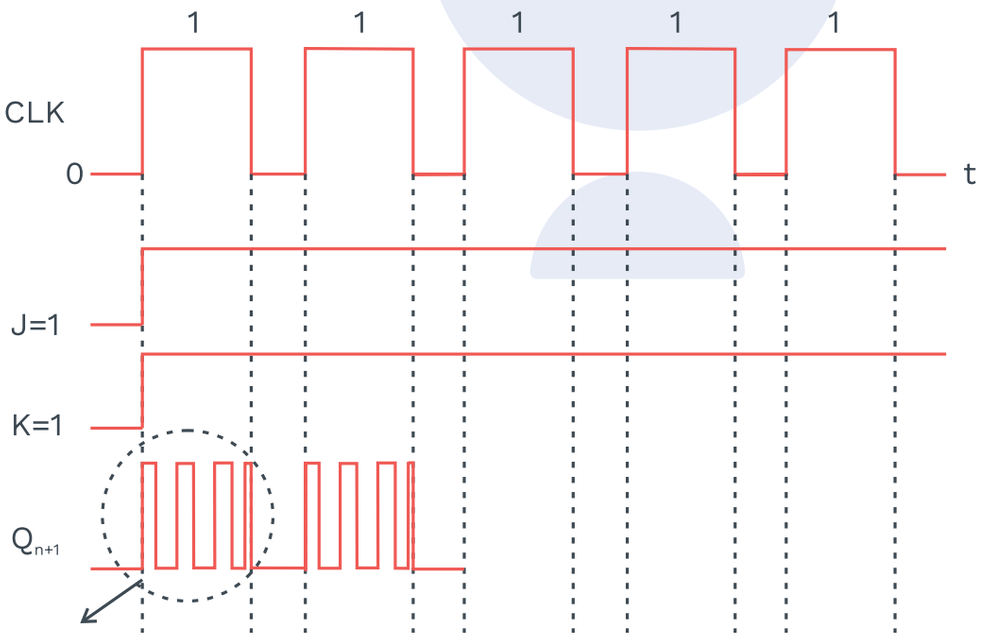


Fig. 3.11 Waveform of JK Flip-Flop

Conditions to avoid racing:

i) Edge triggering JK flip-flop

ii) Master slave JK flip-flop

Master-slave S-R flip-flop:

The truth table is the same as that of the edge-triggered S-R flip-flop, except for the way it is clocked.

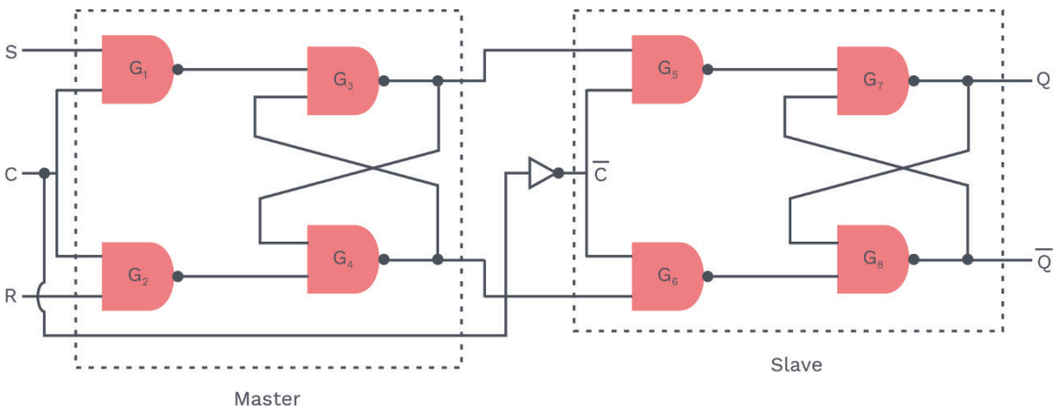


Fig. 3.12 Logic Diagram of Master-Slave S-R Flip-Flop

Inputs			Output	Comments
S	R	CLK	Q_N	
0	0	⎍	Q_{n-1}	Latch
0	1	⎍	0	Reset
1	0	⎍	1	Set
1	1	⎍	x	Invalid

Table 1.12 Truth Table of Master-Slave S-R Flip-Flop

Conversion between flip-flops:

1) The characteristic table of the target flip-flop is obtained.

2) The next state in the characteristic table (obtained in step 1) is replaced using the excitation table of the given flip-flop.

3) The expression for the i/p of the given flip-flop is then realised.

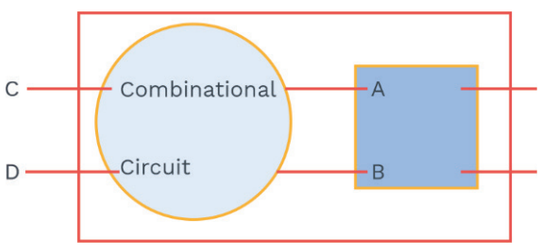


Fig. 3.13

SOLVED EXAMPLES

Q1 **Convert S–R flip-flop to J–K flip-flop.**

Sol: Step 1: Characteristic table of J-K flip-flop.

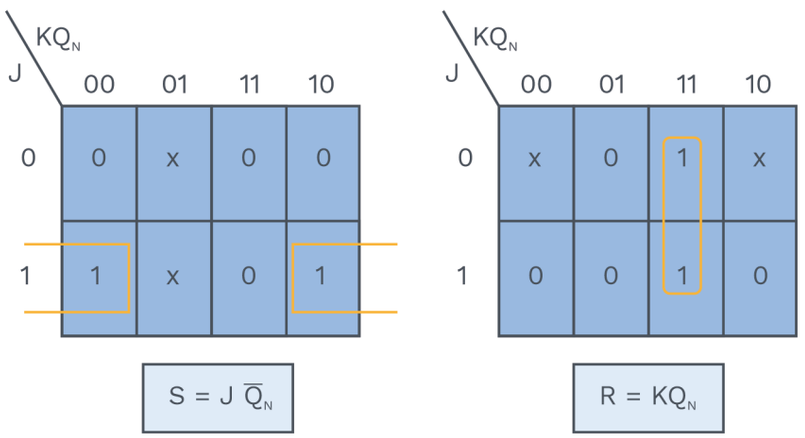
J	K	Q_N	Q_{N+1}
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	0

Step 2: Replace the next state using the excitation table of SR flip-flop

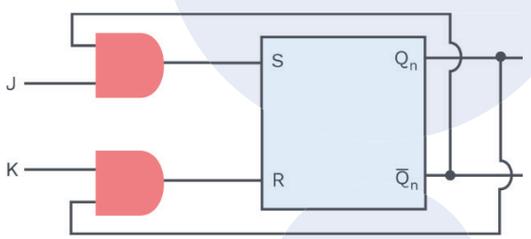
J	K	Q_N	Q_{N+1}	S	R
0	0	0	0	0	x
0	0	1	1	x	0
0	1	0	0	0	x
0	1	1	0	0	1
1	0	0	1	1	0
1	0	1	1	x	0
1	1	0	1	1	0
1	1	1	0	0	1

where ‘x’ is don’t care

Step 3: The expression for the input of a given flip-flop is obtained and realised.



Realisation

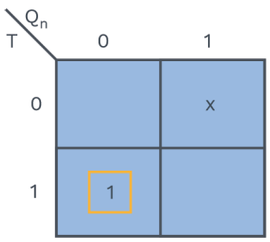


Q2 **Convert SR flip-flop to T flip-flop.**

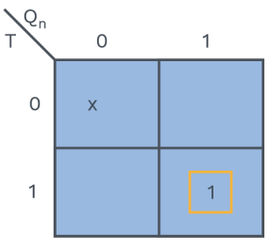
Sol: Following is the characteristic table of T flip-flop and excitation table of S-R flip-flop.

T	Q_n	Q_{n+1}	S	R
0	0	0	0	x
0	1	1	x	0
1	0	1	1	0
1	1	0	0	1

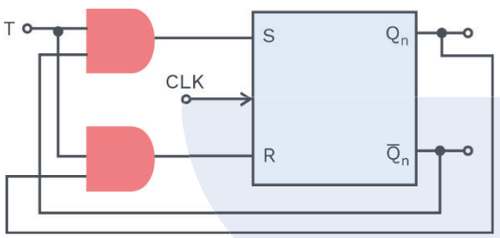
Step 3:



$S = T\,\overline{Q}_N$



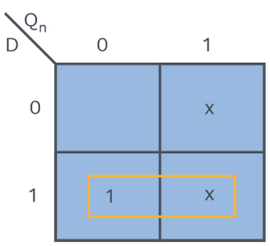
$R = T\,Q_N$



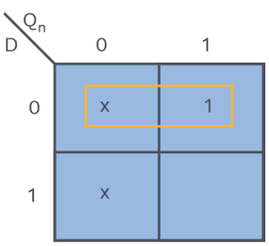
Q3 **Convert J–K flip-flop to D flip-flop.**

Sol: Let's construct the characteristic table of D flip-flop and replace next state with the excitation table of J-K flip-flop.

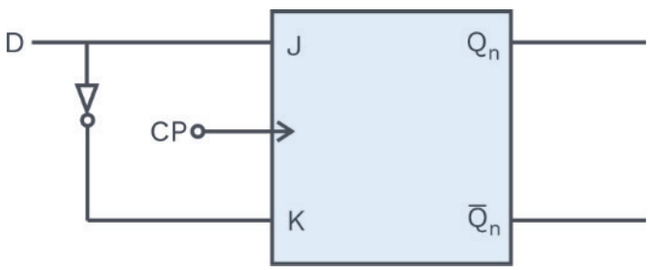
D	Q_N	Q_{N+1}	J	K
0	0	0	0	x
0	1	0	x	1
1	0	1	1	x
1	1	1	x	0



$J = D$

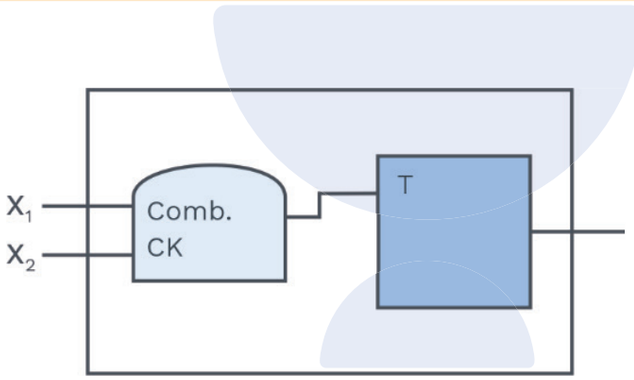


$K = \overline{D}$



Q4 **A new Flip-flop x_1, x_2 has characteristic equation $Q_n = \overline{x}_1 \overline{Q} + \overline{x}_2 Q$. Realise it using T–flip-flop.**

Sol:



Characteristic equation:

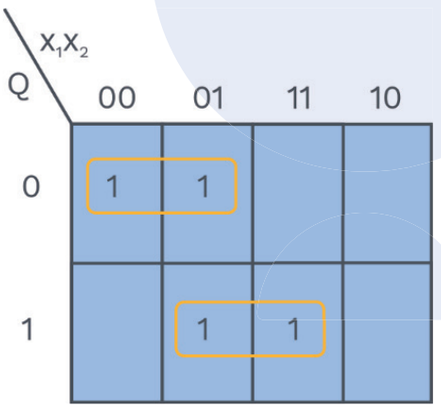
$$Q_n = \overline{x_1}\,\overline{Q} + \overline{x_2}\,Q$$

Excitation table for T flip-flop.

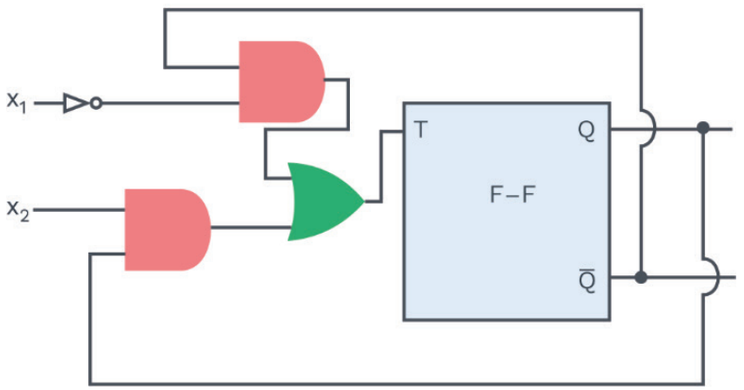
Q	Q_N	T
0	0	0
0	1	1
1	0	1
1	1	0

From the characteristic equation of new flipflop and excitation table of the given flip-flop

x_1	x_2	Q	Q_N	T
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	0	1
1	0	0	0	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	1

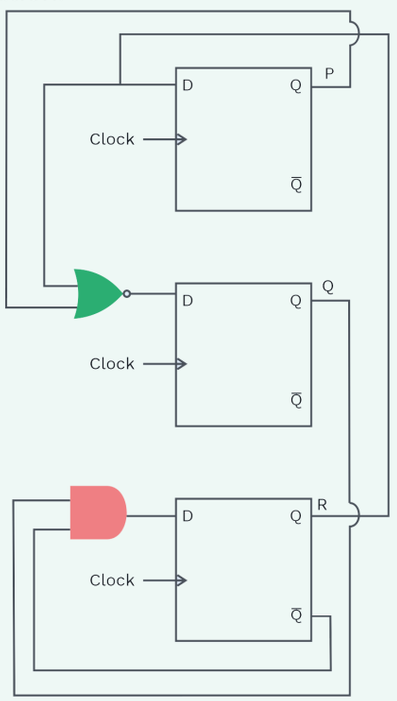


$$T = \overline{x_1}\,\overline{Q} + x_2\,Q$$



Previous Years' Question

Consider the following circuit involving three D–type Flip-flops used in a certain type of counter configuration. **(GATE-2011)**



If at some instance prior to the occurrence of the clock edge, P, Q and R have value 0, 1 and 0 respectively, what shall be the value of PQR after the clock edge?

a) 000 **b)** 001
c) 010 **d)** 011

Sol: d)

3.3 REGISTERS

A flip-flop can only store 1 bit of data, either '0' or '1'; it is also called single bit register. The more is the number of bits to be stored, the more number of flip-flops are used. A register is a group of flip-flops that is used to store digital data. The storage capacity of the register depends on the size(number of bits) of the digital data it can store. The process of setting and resetting registers is known as loading. Loading can be serial or parallel.

Serial in serial out (SISO) shift register:

SISO shift registers accept data as input serially, i.e. one bit at a time and also output digital data serially. The logic diagram of 4 bit serial-in,

serial-out shift register is shown in figure 3.6. With four stages, i.e. four flip-flops, the register can store up to 4 bits of data. Serial data is applied at the D_1 input of the first flip-flop. The Q_1 output of the first flip-flop is connected to the D_2 input of the second flip-flop and the Q_2 output of the second flip-flop is connected to the D_3 input of the third flip-flop and the same happens for the last flip-flop. The Q_4 terminal of the last flip-flop outputs the data.

Grey Matter Alert!

When serial data is transferred into a register, each new bit is clocked into the first flip-flop at the postive-going edge of each clock pulse. The bit that was previously stored by the first flip-flop is transferred to the second flip-flop and so on. The bit that was stored by the last flip-flop is shifted out.

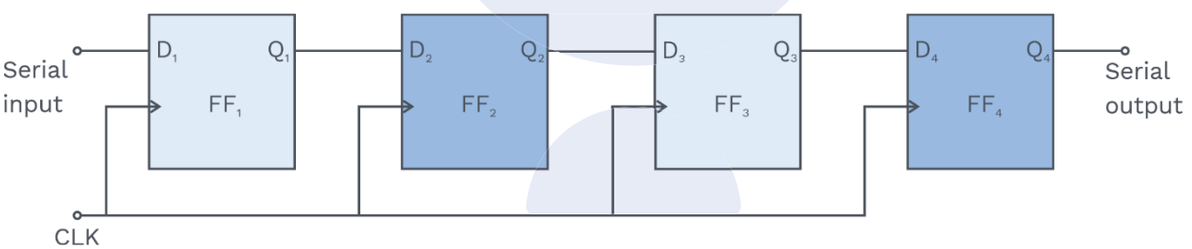


Fig. 3.14 4-Bit Serial-In, Serial-Out Shift-Register

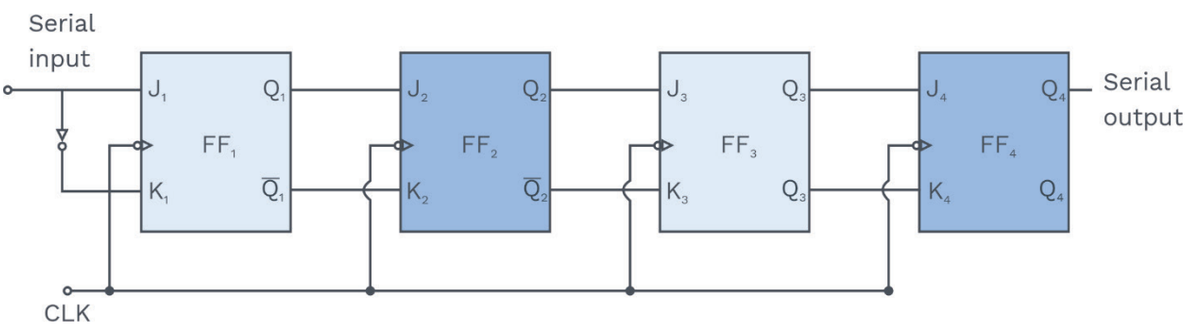


Fig. 3.15 A 4 Bit Serial-In, Serial-Out Shift-Register Using JK FF

Serial-in, parallel-out shift register:

Figure 3.8 shows the diagram and the logic symbol of a 4-bit serial-in, parallel-out, shift register. In SIPO registers, data are fed into the registers serially and register shifts output the data in parallel form.

Once the data bits are stored, all bits are available simultaneously.

Grey Matter Alert!

The serial–IN, Parallel–OUT shift register can be used as a serial–IN, serial–OUT shift register if the output is taken from the Q terminal of the last Flip-flop.

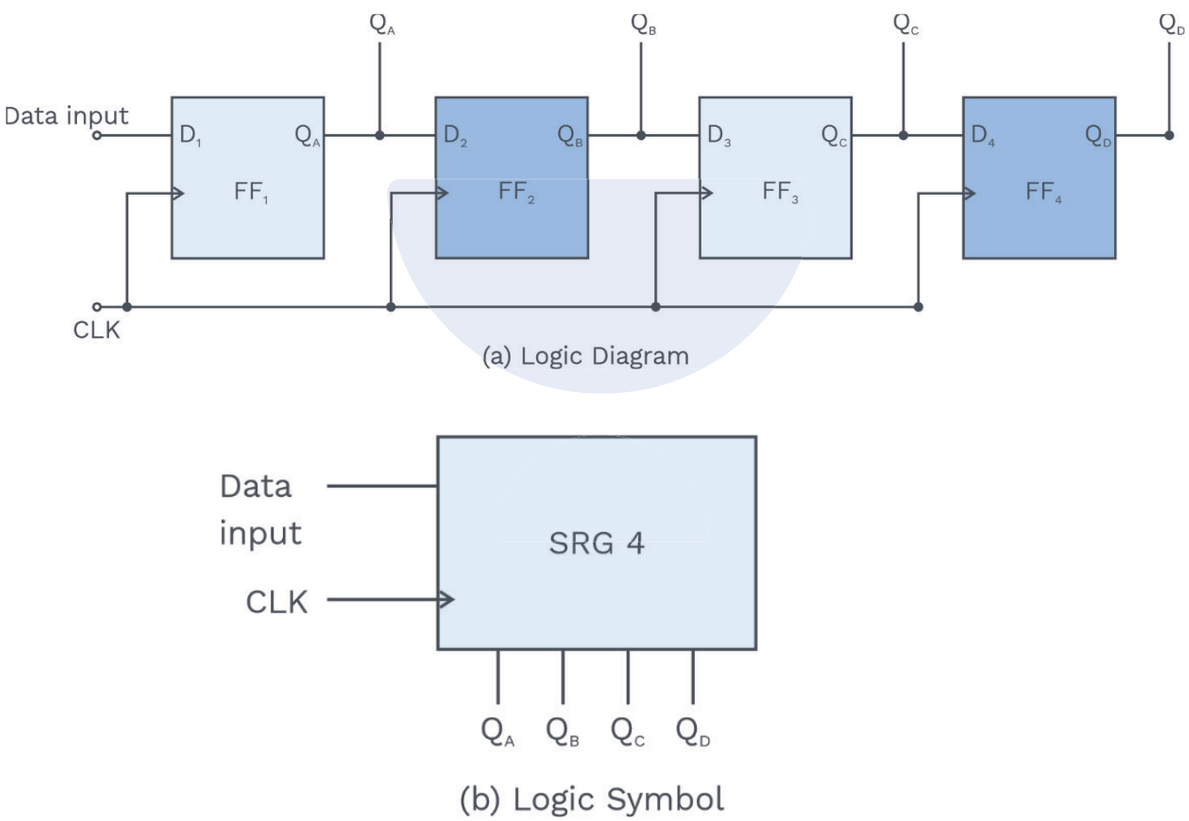


Fig. 3.16 A 4 Bit Serial-IN Parallel-OUT Shift Register

Parallel IN parallel out (PIPO):

In PIPO, shift register input and output of the data is in a parallel way.

Figure 3.9 shows a 4-bit parallel-in, parallel-out shift register using D flip-flops. When clock pulse is applied, and data is applied to the 'D' i/p terminals of the flip-flops, the D i/ps are shifted in the output terminal of the flip-flops.

The stored data is outputted instantaneously in parallel form.

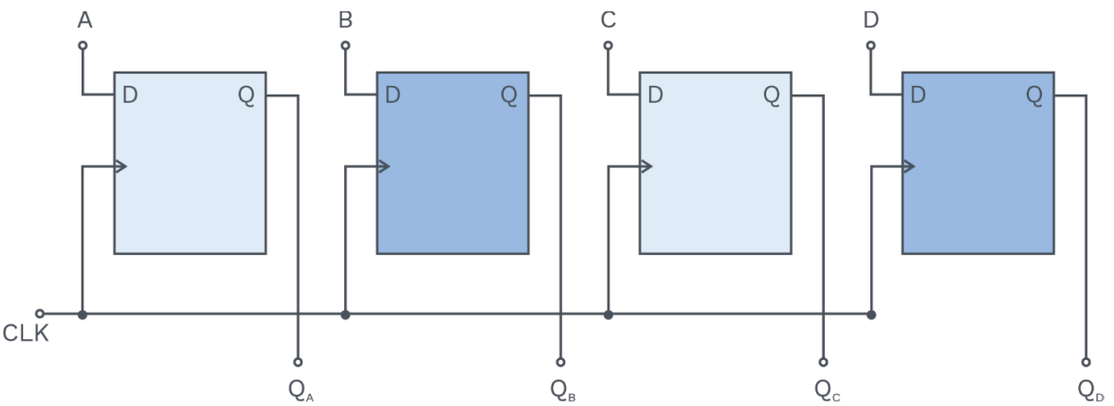
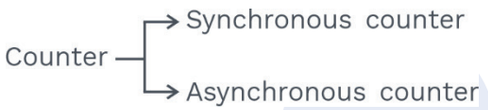


Fig. 3.16 PIPO Shift Register

3.4 COUNTER

Introduction:

A digital counter is a group of Flip-flops that changes its state in response to the clock applied to the counter. The flip-flops are interconnected in such a way that their combined state at any time is the binary equivalent of the total number of pulses that have occurred till that moment.



Another name for the asynchronous counter is ripple counter.

Grey Matter Alert!

In ripple counter, the flip-flops within the counter are not made to change the states exactly at the same time. This is because the flip-flops within the counter are not triggered simultaneously. An asynchronous counter uses T flip-flops usually to perform a counting function.

In the asynchronous counter, the output of the first stage acts as a clock to the second stage. The output of the second stage acts as a clock to 3rd stage and so on.

Synchronous counters are clocked such that each of the flip-flops in the counter is triggered at the same time.

Grey Matter Alert!

Synchronous counters are faster than asynchronous counters because the propagation delay involved is less.

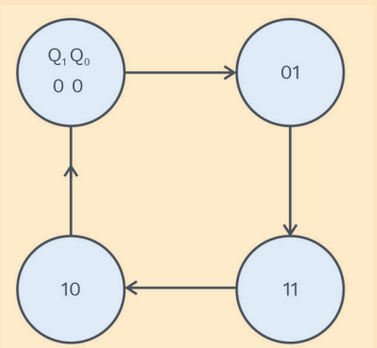
Counter design:

1) From the state diagram, get the state table.

2) Identify the flip-flop to be used and replace the concerned next state using the excitation table of the associated flip-flop.

3) Get the expressions for inputs and realise them.

Q5 **Design a synchronous for the following state diagram using T – FFs**



Sol: Number of states = 4

Minimum number of flip-flops required = $\lceil \log_2 4 \rceil = 2$

State table:

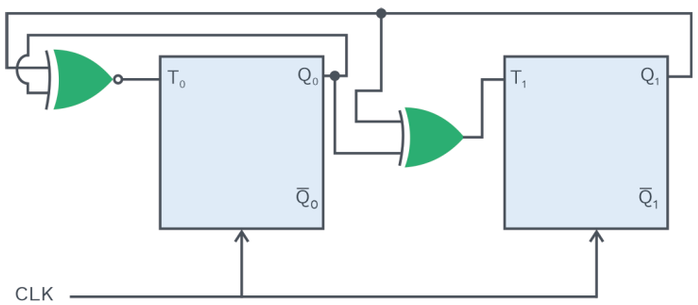
Q_1	Q_0	Q_{1N}	Q_{0N}	T_1	T_0
0	0	0	1	0	1
0	1	1	1	1	0
1	1	1	0	0	1
1	0	0	0	1	0

$$T_1 = \bar{Q}_1 Q_0 + Q_1 \bar{Q}_0$$

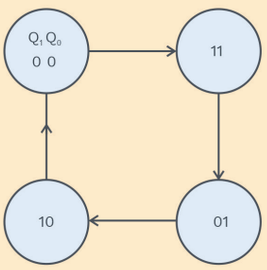
$$\boxed{T_1 = Q_1 \oplus Q_0}$$

$$T_0 = \bar{Q}_1 \bar{Q}_0 + Q_1 Q_0$$

$$\boxed{T_0 = Q_1 \odot Q_0}$$



Q6 **Consider the following state diagram which is to be designed using T-FF for MSB (Most Significant Bit) and an unknown flip-flop 'XY' for LSB (Least Significant Bit). The behaviour of XY flip-flop is shown below. Design a synchronous counter.**



X	Y	Q_N
0	0	0
0	1	Q
1	0	$\bar{Q}$
1	1	1

Sol: Characteristic table for XY flip-flop from the given function table

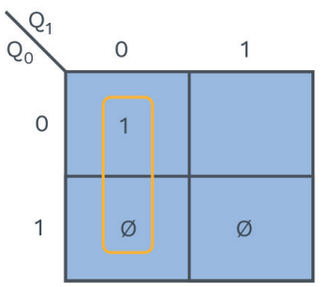
X	Y	Q	Q_N
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

From the state diagram:

Q_1	Q_0	Q_{1N}	Q_{0N}	T	x	y
0	0	1	1	1	1	Ø
1	1	0	1	1	Ø	1
0	1	1	0	1	Ø	0
1	0	0	0	1	0	Ø

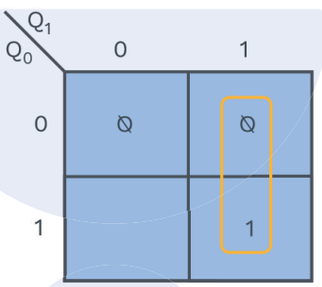
Ø ⟶ Don't care

for x :

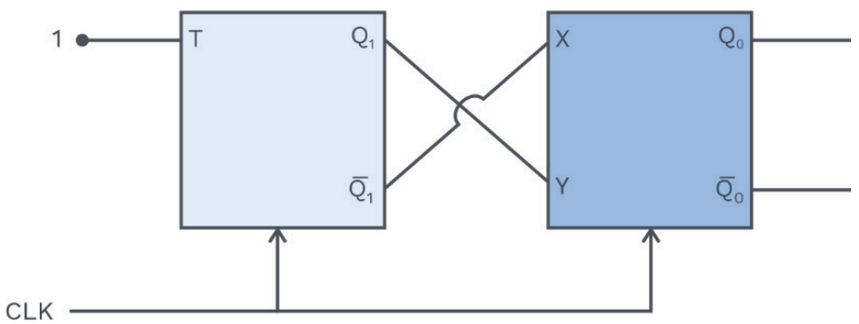


$x = \overline{Q}_1$

for y :



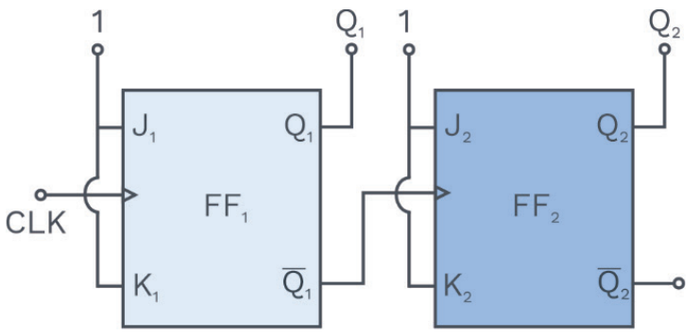
$Y = Q_1$



Asynchronous counter:

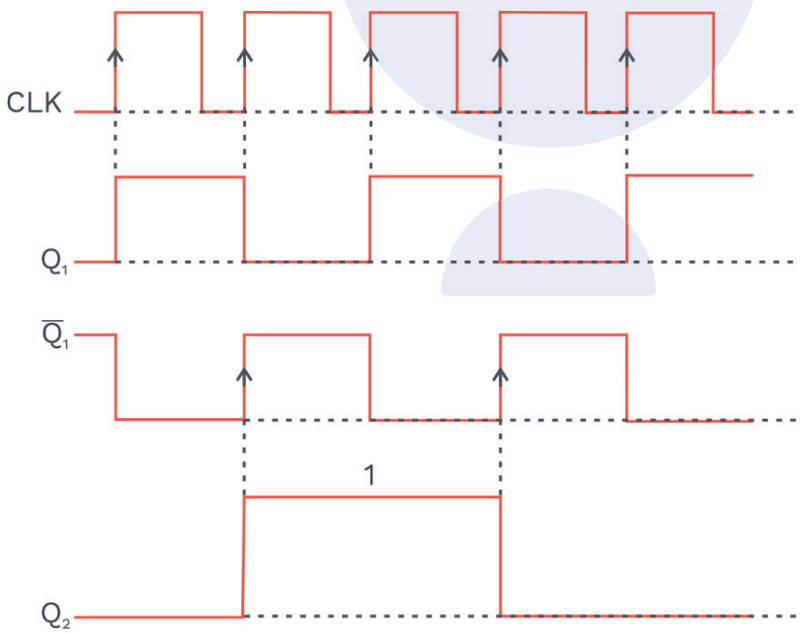
Two-bit ripple up-counter using positive edge-triggered flip-flops:

A 2-bit ripple up-counter, using (positive edge-triggered) J-K Flip-flops, are shown in figure 3.10. The $\overline{Q}_1$ output of the first flip-flop is connected to the clock of FF_2. The external clock signal is applied to the first flip-flop FF_1. The FF_1 toggles (changes its state) at the positive edge of each clock pulse, and FF_2 toggles when $\overline{Q}$ changes from '0' to '1'.



a) Logic Diagram

Fig. 3.17 Asynchronous 2-bit up Counter



b) Timing Diagram

Fig. 3.18

Two-bit ripple down-counter using positive edge-triggered flip-flop:

A 2-bit ripple down-counter using positive edge-triggered J-K FFs is shown in figure 3.11. The Q_1 output of the first flip-flop is connected to the clock of FF_2. The external clock pulse is directly applied to FF_1. The FF_1 toggles at the positive-going edge of each clock pulse.

The counting sequence is 00, 11, 10, 01, 00, 11, 10 ..... etc.

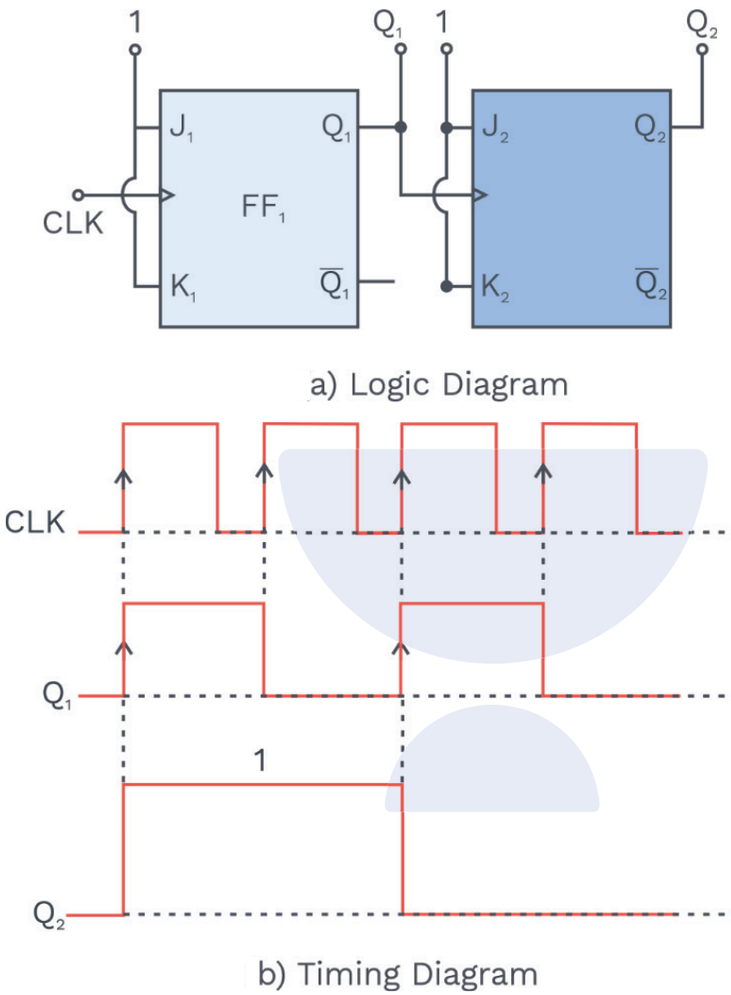


Fig. 3.19 2-bit Ripple Up Counter Design

Mod-8 up-counter:

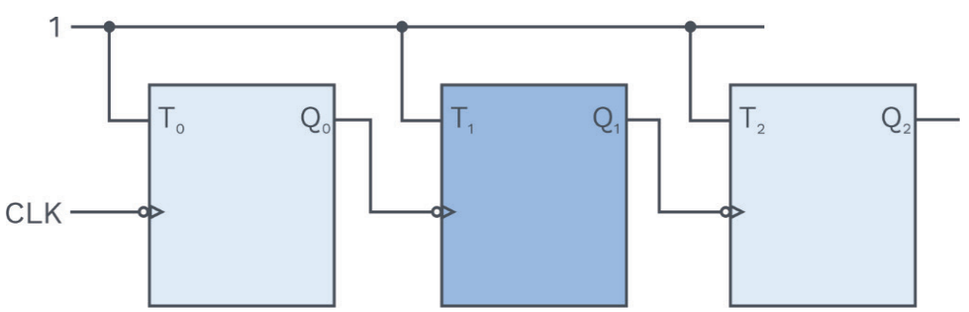


Fig. 3.20 Mod-8 Up Counter

All the flip-flops are negative edge triggered, i.e. changes in their state when the clock makes a transition from 1 to 0.

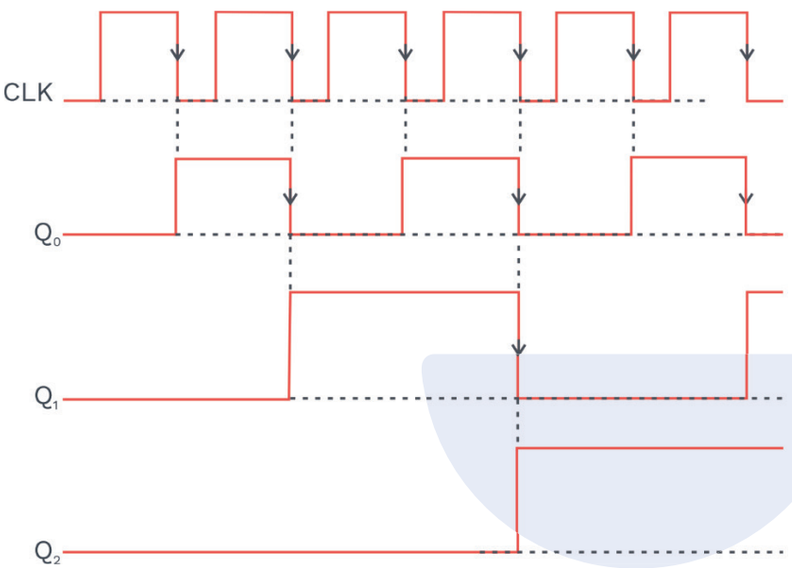


Fig. 3.21 Timing Diagram of Mod-8 up Counter

Time period of $Q_0 = 2 \times T_{CLK}$

Frequency, $f_{Q0} = \frac{1}{2} f_{CLK}$

Time period $Q_1 = 2 \times T_{Q0}$

$$f_{Q1} = \frac{1}{2} f_{Q0} = \frac{1}{4} f_{CLK}$$

Time period $Q_2 = 2 \times T_{Q1}$

$$f_{Q2} = \frac{1}{2} f_{Q1}$$

$$= \frac{1}{4} f_{Q0}$$

$$= \frac{1}{8} f_{CLK}$$

In the state Transition table,

$Q_{0N} = \overline{Q}_0$, for every clock

$Q_{1N} = \overline{Q}_1$ {When $Q_0 : 1 \rightarrow 0$}

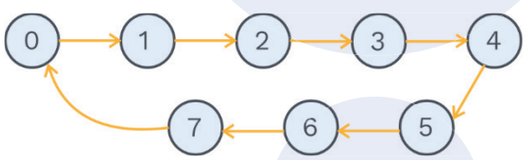
$Q_{2N} = \overline{Q}_2$ {When $Q_1 : 1 \rightarrow 0$}

State transition table:

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}
0	0	0	0	0	1
0	0	1	0	1	0
0	1	0	0	1	1
0	1	1	1	0	0
1	0	0	1	0	1
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	0

Table 1.13 State Transition Table

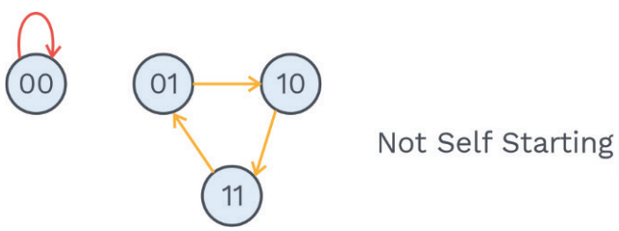
State diagram:



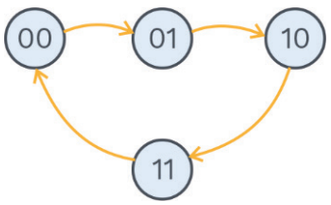
Self starting and free running counters:

- A counter is called self starting counter if the counter can enter counting loops irrespective of the starting state of the counter.

Example



- A counter is called to be free running counter if the counter contains all possible states in the counting loop.

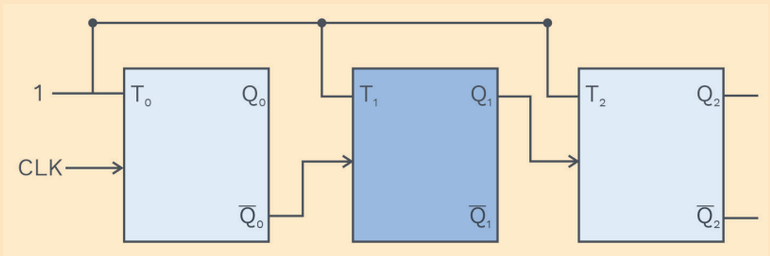


It is both self starting and free running.

Grey Matter Alert!

Every free running counter is self starting counter but every self starting counter is not necessarily be the free running counter.

Q7



If the initial state of the counter is $Q_2\ Q_1\ Q_0$ = 101. What will be the state after 5 clock cycles?

Sol: $Q_0 \rightarrow$ Present state

$Q_{0N} \rightarrow$Next state

$Q_{0N} = \overline{Q}_0$, for every clock

$Q_{1N} = \overline{Q}_1 \ \ \overline{Q}_0 : 0 \rightarrow 1$

$Q_0 : 1 \rightarrow 0$

$Q_{2N} : \overline{Q}_2, \quad Q_1 : 0 \rightarrow 1$

State table:

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}
1	0	1	0	1	0
0	1	0	1	1	1
1	1	1	0	0	0
0	0	0	0	0	1
0	0	1	1	1	0

Table 1.14 State Table

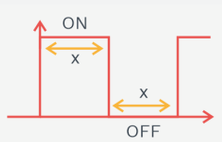
Sol: 110

Cascading of ripple counters:

Ripple counters can be cascaded to increase the modulus of the counter. A mod-M and a mod-N counter cascaded gives a mod-MN counter. While cascading, the most significant stage of the first counter, is connected to the toggling stage of the second counter. The order of cascading does not affect the frequency division; however, the duty cycle of the most significant output may depend on the order in which the counters are cascaded.

Grey Matter Alert!

A duty cycle is the function of one period in which a signal or system is active. It is commonly expressed in percentage.



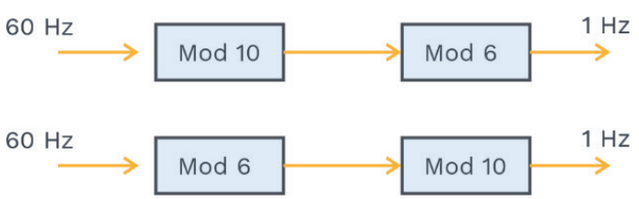


Fig. 3.22 Examples of Cascaded Counters

Rack Your Brain

A binary ripple counter is required to count up to $(16, 584)_{10*}$ How many flip-flop are required? If the clock frequency is 7.940 MHz, what is the frequency at the output of the MSB?

Synchronous counter:
Synchronous counters are good in terms of speed and decoding, but they require more circuitry than that of asynchronous counters.

Design of synchronous counters:
Step 1: Find the required value of 'n' where 'n' is the number of the flip-flop. The smallest value of n is such that the number of states $N \leq 2^n$ and the desired counting sequence is to be found out.

Step 2: The state diagram is drawn, showing all the possible states.
Step 3: Choice of flip-flop and excitation table:
Select the type of flip-flops to be used and write the excitation table.

Step 4: Minimal expressions for excitations:
The minimal expression is obtained for the excitations of the flip-flop using the K-maps drawn.

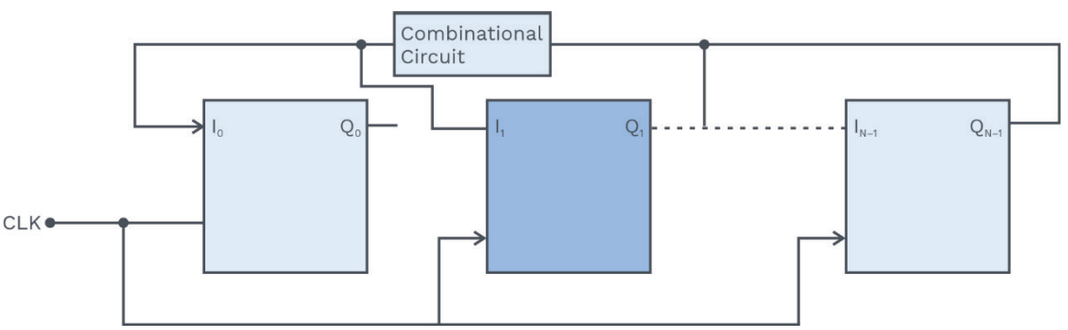


Fig. 3.23 Synchronous Counter

- Propagation delay in synchronous counter is:

$$T_{psyn} = T_{FF} + T_{combination}$$

$$T_{CLK} \geq T_{psyn}$$

- Propagation delay in asynchronous counter is:

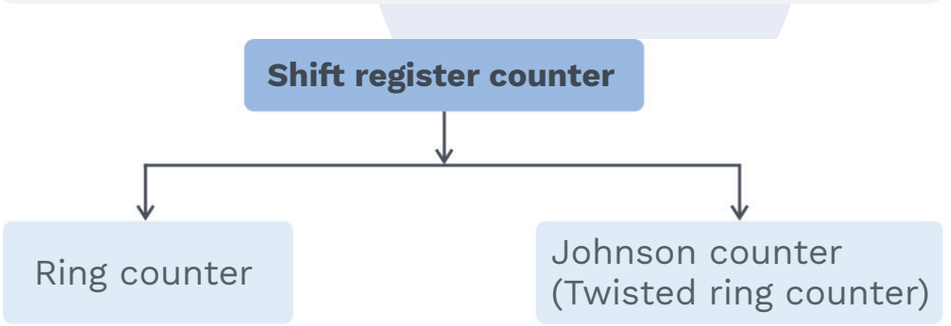
$$T_{psyn} = N \times T_{FF} + T_{combination}$$

$$T_{CLK} \geq T_{psyn}$$

Shift register counters:
The application of the shift register is that it can be arranged to form different types of counters.

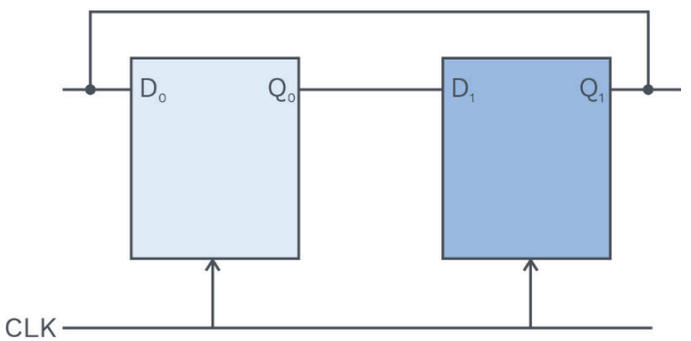
Grey Matter Alert!

Shift register counter are obtained from serial–in, serial–out shift register by providing feeback from the output of the last flip-flop to the input of the first flip-flop.



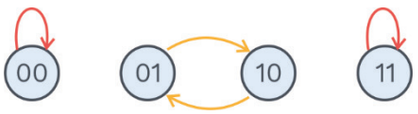
Ring counter:
The ring counter is the simplest shift register counter. The flip-flop is arranged the same as in a normal shift register, i.e. Q output of each stage is connected to the D input of the next stage, but the Q output of the last flip-flop is connected back to the input terminal of the first flip-flop.

Mod 2 ring counter:



Mod 2 ring counter

D_1	D_0	Q_1	Q_0	Q_{1N}	Q_{0N}
0	0	0	0	0	0
1	0	0	1	1	0
0	1	1	0	0	1
1	1	1	1	1	1



Note:

Even though there are all four states, it is performing mod-2 count.

Mod-3 Ring counter:

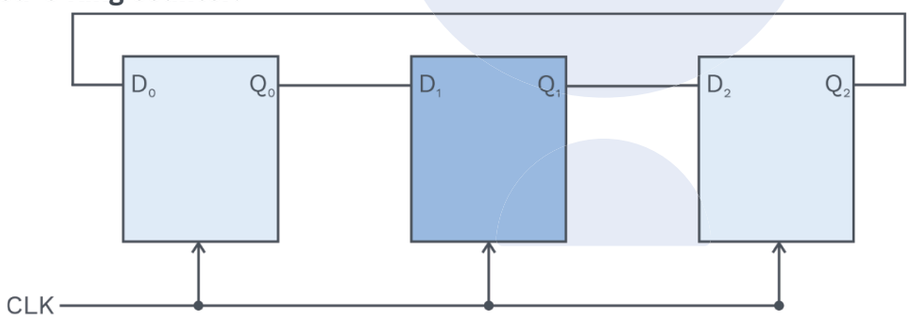
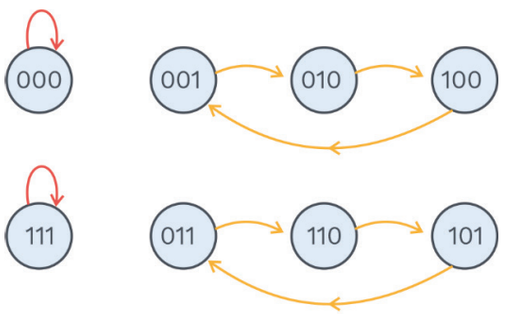


Fig. 3.24 Mod 3 Ring Counter

State table:

Q_2	Q_1	Q_0	Q_{2N}	Q_{1N}	Q_{0N}	D_2	D_1	D_0
0	0	0	0	0	0	0	0	0
0	0	1	0	1	0	0	1	0
0	1	0	1	0	0	1	0	0
0	1	1	1	1	0	1	1	0
1	0	0	0	0	1	0	0	1
1	0	1	0	1	1	0	1	1
1	1	0	1	0	1	1	0	1
1	1	1	1	1	1	1	1	1



$\therefore$ It performs mod–3 count

Grey Matter Alert!

The ring counter performs mod (N) count.
N $\rightarrow$ Number of FFs
Example Given mod-4 ring counter then, N = 4

Previous Years' Question

We want to design a synchronous counter that counts the sequence 0 – 1 – 0 – 2 – 0 – 3 and then repeats. The minimum number of J–K flip-flops required to implement this counter is ..... **(GATE-2016, SET-1)**

Sol: 4

Twisted ring counter (johnson counter):

This counter is designed from a SISO shift register by giving feedback from the inverted output of the last FF to the input of the first flip-flop. The output of every stage is connected to the input terminal of the next stage and so on. $\overline{Q}$ the output of the last stage is connected to the input terminal of the first stage.

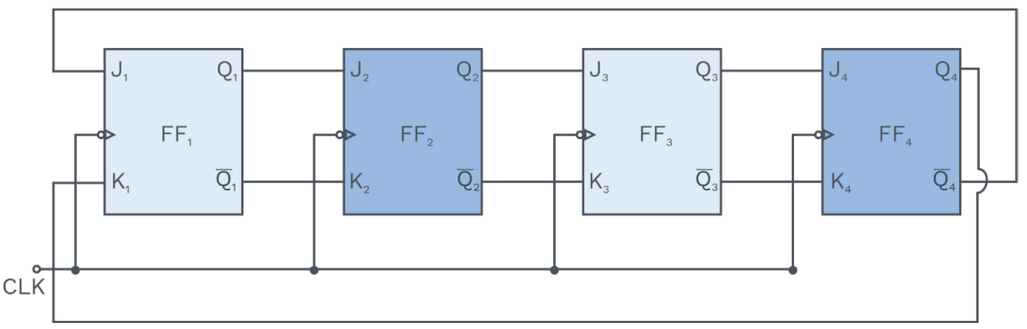


Fig. 3.25 Logic Diagram of a 4-bit Johnson Counter Using J-K Flip-Flop

Mod 4 johnson counter:

Grey Matter Alert!

Johnson counter performs mod (2N) count.
N → Number of FFs
Example given mod-4 ring counter then, 2N = 4 ⇒ N = 2

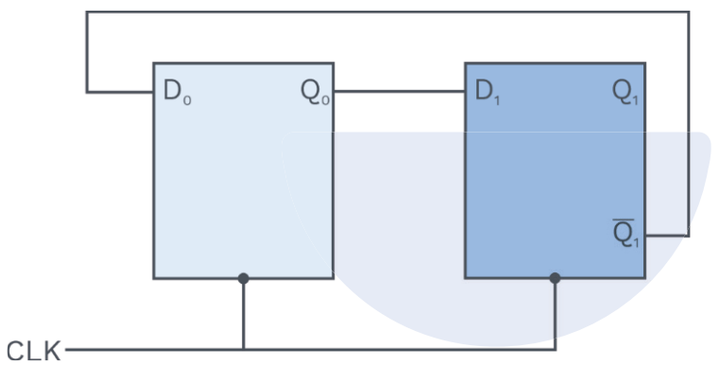


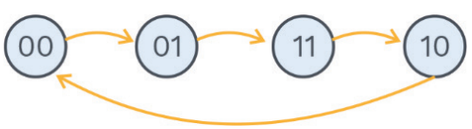
Fig. 3.26

State table:

Q_1	Q_0	Q_{1N}	Q_{0N}	D_1	D_0
0	0	0	1	0	1
0	1	1	1	1	1
1	0	0	0	0	0
1	1	1	0	1	0

Table 1.15

State diagram:



∴ The counter performs mod-4 count.

Chapter summary

- Sequential circuit → Output variables at any instant are dependent on the present state and past history of the system.
- Synchronous sequential circuit → Memory elements are clocked flip-flops
- Asynchronous sequential circuit → Memory elements are either unlocked flip—flops (external clock is not given, output of flip—flops is given as clock to next flip-flops) or time delay elements.
- S—R flip—flop → For S = 1, R = 1, it is a invalid input combination.
- J—K flip—flop → For J = 1, K = 1, flip-flop toggles.
  T—flip—flop → J = K = T
- Registers
  - → SISO
  - → SIPO
  - → PIPO
- Counter
  - → Synchronous counter
  - → Asynchronous counter
- Synchronous counter
  - → Ring counter ($1_O = Q_{N-1}$) [Where Q_{N-1} is the output of the last flip-flop]
  - → Johnson counter ($1_O = \overline{Q}_{N-1}$)

4 Number System

4.1 INTRODUCTION TO DIGITAL SYSTEM

Digital system is very common nowadays. A digital system is easier to design. It is a combination of devices that handles logical information represented in binary form.

Digital computer:

Digital computers follow a sequence of instructions, called a program, that operates on given data. They have the ability to manipulate and represent discrete elements of information.

Introduction to number system:

A number system is a mathematical way of representing numbers. The number system is a basis for counting various items.

Base of number system:

It is also called as radix of a number system, denoted by 'r'.

The base/radix of any number system is the total number of digits/symbols used in that number system.

It decides the total number of digits available in that system.

For example, for a decimal number system, the base is 10, i.e. (r = 10)

Types of number systems:

The commonly used number systems are:

1) Decimal number system (r = 10)

2) Binary number system (r = 2)

3) Octal number system (r = 8)

4) Hexadecimal number system (r = 16)

4.2 BINARY TO DECIMAL CONVERSION

Using the positional weight method, we can convert binary numbers to their decimal equivalents. In positional weight, each binary digit of the number is multiplied by its positional weights, and to obtain the final decimal numbers, all the product terms are added.

SOLVED EXAMPLES

Q1 **Convert $(110010)_2$ to decimal.**

Sol: $(1\ 1\ 0\ 0\ 1\ 0\ )_2$

Decimal equivalent:

$= 2^5 \times 1 + 2^4 \times 1 + 2^3 \times 0 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0$

$= 32 + 16 + 0 + 0 + 2 + 0$

$= (50)_{10}$

Q2 **Convert $(011010.011)_2$ to decimal.**

Sol: $(0\ 1\ 1\ 0\ 1\ 0 \cdot 0\ 1\ 1\ )_2$

Decimal equivalent:

$= 2^5 \times 0 + 2^4 \times 1 + 2^3 \times 1 + 2^2 \times 0 + 2^1 \times 1 + 2^0 \times 0 + 2^{-1} \times 0 + 2^{-2} \times 1 + 2^{-3} \times 1$

$= 0 + 16 + 8 + 0 + 2 + 0 + 0 + 0.25 + 0.125$

$= (26{\cdot}375)_{10}$

Decimal to binary conversion:

- Decimal to binary conversion is done using successive division and multiplication methods.
- The decimal integer is converted to a binary integer by successively dividing by 2, and a decimal fraction is converted to binary by successively multiplying by '2'.
  Decimal fractions are successively multiplied till we get the fraction part of the product as 0.

SOLVED EXAMPLES

Q3 **Convert $(52)_{10}$ to binary.**

Sol:

		Remainder
2 | 52		
2 | 26	–	0
2 | 13	–	0
2 | 6	–	1
2 | 3	–	0 ↑
1	–	1

Reading the remainders from bottom to top, the result is $(52)_{10} = (110100)_2$.

Q4 **Convert $(0.75)_{10}$ to binary.**

Sol:

Given fraction	0.75
Multiply 0.75 by 2	1.50
Multiply 0.50 by 2 ↓	1.00

Reading the integers from top to bottom, $(0.75)_{10} = (0.11)_2$.

Q5 **Convert $(105.15)_{10}$ to binary.**

Sol: Conversion of integer 105

2 | 105	
2 | 52	– 1
2 | 26	– 0
2 | 13	– 0
2 | 6	– 1
2 | 3	– 0
1	– 1

Reading the remainders from bottom to top.

$$(105)_{10} = (1101001)_2$$

Conversion of fraction $(0.15)_{10}$

Given fraction	→ 0.15
Multiply 0.15 by 2	→ 0.30
Multiply 0.30 by 2	→ 0.60
Multiply 0.60 by 2	→ 1.20
Multiply 0.20 by 2	→ 0.40
Multiply 0.40 by 2	→ 0.80
Multiply 0.80 by 2	→ 1.60

This fraction can not be expressed in binary exactly. This process may be terminated after a few steps.
Reading the integers from top to bottom.

$$(0.15)_{10} = (0.001001)_2$$

Therefore, the final result is $(105.15)_{10} = (1101001.001001)_2$.

4.3 OCTAL NUMBER SYSTEM

It is also a positional-weighted system. Its base is 8. Since Its base $8 = 2^3$. The binary number can be grouped into 3, and each group can be represented by an equivalent octal digit.

Octal to decimal conversion:
To convert an octal number to its decimal equivalent, each digit present in the octal number gets multiplied by its positional weight, and then each product term gets added.

SOLVED EXAMPLES

Q6 **Convert $(4057.06)_8$ to decimal.**

Sol:
$$(4057.06)_8 = 4 \times 8^3 + 0 \times 8^2 + 5 \times 8^1 + 7 \times 8^0 + 0 \times 8^{-1} + 6 \times 8^{-2}$$
$$= 2048 + 0 + 40 + 7 + 0 + 0.0937$$
$$= (2095.0937)_{10}$$

Decimal to octal conversion:
To convert a mixed decimal number to a mixed octal number, convert the integer and fraction parts separately.

Q7 **Convert $(378.93)_{10}$ to octal.**

Sol:

Successive division			Remainders
8 | 378	—		2 ↑
8 | 47	—		7 ↑
8 | 5	—		5 ↑
0			

Read the remainders from bottom to top.

$$(378)_{10} = (572)_8.$$

Conversion of $(0.93)_{10}$ to octal

$$0.93 \times 8 = 7.44$$
$$0.44 \times 8 = 3.52$$
$$0.52 \times 8 = 4.16$$
$$0.16 \times 8 = 1.28$$

Read the integers to the left of the octal point downwards.

Therefore, $(0.93)_{10} = (0.7341)_8$

Hence, $(378.93)_{10} = (572.7341)_8$.

Binary to octal conversion:

A binary number can be converted into octal numbers by grouping the binary number into 3, and each group of 3 can be represented by an octal digit.

SOLVED EXAMPLES

Q8 **Convert $(110101.101010)_2$ to octal.**

Sol: Groups of three bits 110 101 101 010

Convert each group of 3 bits to octal 6 5 5 2

The result is $(65.52)_8$.

4.4 HEXADECIMAL NUMBER SYSTEM

A Hexadecimal number system is a positional-weighted system. The base of this number system is 16, i.e., it has 16 independent symbols. The symbols used are 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F.

Hexadecimal to decimal conversion:

To convert a hexadecimal number to its decimal equivalent, each digit present in the hexadecimal number gets multiplied with its positional weight and then each product term gets added.

Decimal to hexadecimal conversion:

To convert a decimal integer number to hexadecimal, the decimal number is successively divided by 16 till the quotient is 0.

SOLVED EXAMPLES

Q9 **Convert $(2598.675)_{10}$ to hexadecimal.**

Sol:

Successive division		Remainder	
		Decimal	Hex
16 | 2598	—	6	6
16 | 162	—	2	↑ 2
16 | 10	—	10	A
0			

Reading the remainder upwards, $(2598)_{10} = (A26)_{16}$

Conversion of $(0.675)_{10}$

$0.675 \times 16 \quad — \quad 10.8$ ↓
$0.800 \times 16 \quad — \quad 12.8$
$0.800 \times 16 \quad — \quad 12.8$
$0.800 \times 16 \quad — \quad 12.8$

Reading the integers to the left of hexadecimal point downwards,

$$(0.675)_{10} = (0.ACCC)_{16}$$

Therefore, $(2598.675)_{10} = (A26.ACCC)_{16}$

Hexadecimal to binary conversion:
To convert a hexadecimal number to it's equivalent binary representation, each hexadecimal digit is replaced by its 4 bit equivalent binary.

SOLVED EXAMPLES

Q10 **Convert $(2BAC)_{16}$ to binary.**

Sol: $(2\ B\ A\ C)_2$
Convert the earn digit to its 4-bit binary equivalent

2	B	A	C
0010	1011	1010	1100

Q11 **Convert $(3A9E.B0D)_{16}$ to binary.**

Sol:

Given hex number is	3	A	9	E	B	0	D
Convert each hex digit to 4-bit binary	0011	1010	1001	1110	1011	0000	1101

The result is $(0011\ 1010\ 1001\ 1110\ .\ 1011\ 0000\ 1101)_2$

Binary to hexadecimal conversions:
To convert a binary number to its equivalent hexadecimal number,make each group of 4-bits (if there is no decimal point, then start grouping from LSB, and if there is a decimal point, then start grouping from LSB before the point and start grouping from MSB after the decimal point) and replace each group with its equivalent hexadecimal digit.

SOLVED EXAMPLES

Q12 **Convert $(001011011011)_2$ to hexadecimal.**

Sol:

Group of 4 bits are	0010	1101	1011
Convert each group to hexadecimal	2	D	B

The result is $(2DB)_{16}$.

4.5 BINARY CODED DECIMAL (BCD)

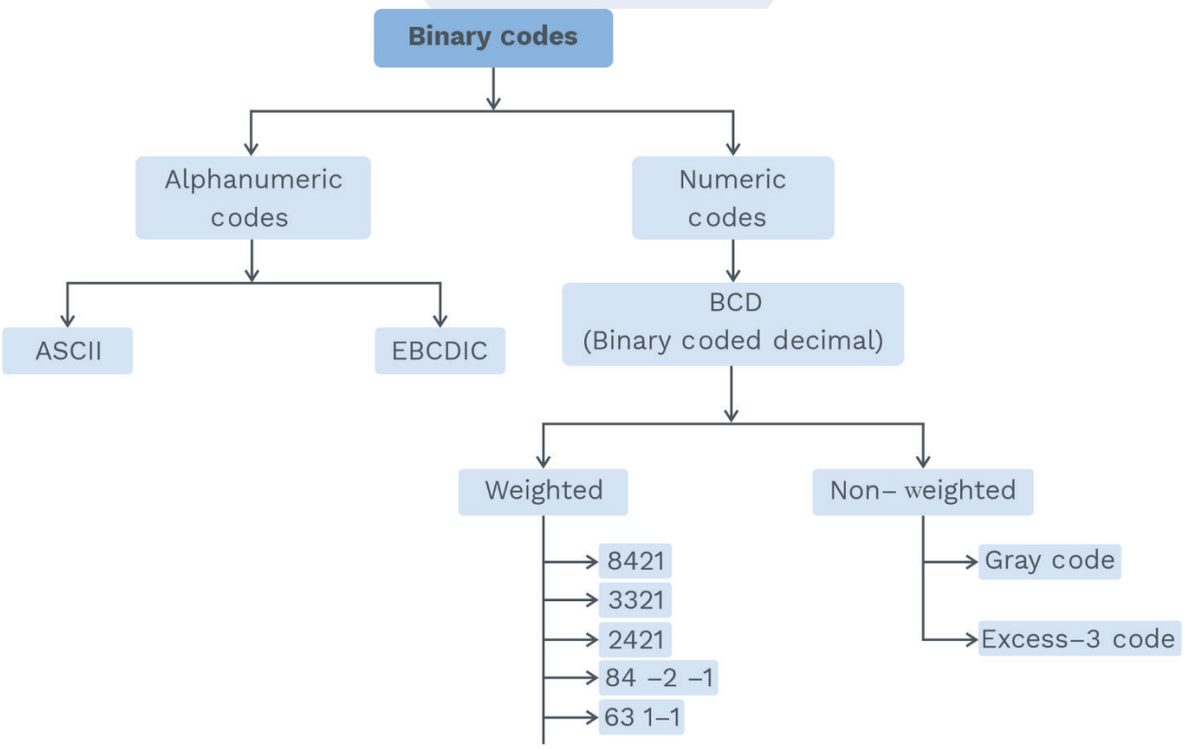
In this type of code, each digit of the decimal number is represented by its 4-bit binary equivalent. It is known as natural binary code due to the positional weights of 8,4,2,1.

Grey Matter Alert!

- The decimal 14 can be represented as 1110 in pure binary but as 00010100 in 8421 code.
- Another disadvantage of the BCD code is that arithmetic operations are more complex than they are in pure binary.

In BCD, there exist six combinations that are illegal and are not used 1010, 1011, 1100, 1101, 1110, and 1111, i.e., they are not part of the 8421 BCD code.

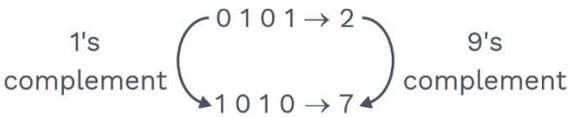
Classification of binary codes:



Self complementary code:

In self complementary code, the code of a digit and the code of 9's complement of the digit are 1's complement to each other.

Example: Let's consider 6 3 1 '-1' code



Q13 **What is the maximum n-bit number in base x, when represented in decimal (10)?**

Sol: A maximum n-bit number possible in base x → In base 'x', maximum digit possible is 'x – 1'. Eg. For x = 10, maximum digit possible is 9.

$$\left(\frac{x-1}{n-1}\cdots\frac{x-1}{2}\,\frac{x-1}{1}\,\frac{x-1}{0}\right)_x$$

↓ in decimal (base 10)

$x^{n-1}(x-1) + x^{n-2}(x-1) + \ldots (x-1)x^0$

$= (x-1)\,[1 + x + x^2 + \ldots x^{n-1}]$

$$= (x-1)\left[\frac{x^n-1}{x-1}\right]$$

$= (x^n - 1)_{10}$

Rack Your Brain

What is the minimum number of bits required when $(617)_{10}$, is represented in base 4 number system?

SOLVED EXAMPLES

Q14 **What is the minimum number of bits required to represent a 32 bit decimal number in binary number?**

Sol: Let's say 'n' bits in binary is required to represent a 32-bit decimal number.

Maximum 32-bit number in decimal = $10^{32} - 1$, and with n-bits, the maximum number in binary when represented in base 10 = $2^n - 1$

Now, $10^{32} - 1 \leq 2^n - 1$

$\Rightarrow$ $10^{32} \leq 2^n$

$\Rightarrow$ $n \geq \log_2 10^{32}$

$\Rightarrow$ $\boxed{n \geq 32 \log_2 10}$

Q15 **What is the minimum number of digits in base 2 required to represent a 10 digit number in base 8?**

Sol: Let's say n bits are required to represent a 10-bit number of base 8 in binary

maximum 10 digit number in base 8 when represented in decimal = $8^{10} - 1$.

With 'n' bits, the maximum number in binary when represented in base 10 = $2^n - 1$

$2^n - 1 \geq 8^{10} - 1$

$\Rightarrow$ $2^n \geq 8^{10}$

$\Rightarrow$ $n \geq \log 8^{10}$

$\Rightarrow$ $n \geq 10 \log 2^3$

$\Rightarrow$ $\boxed{n \geq 30}$

Ans. 30.

Q16 **Solve for a, b, c**
$(11)_2 + (22)_3 + (33)_4 + (44)_5 = (abc)_6$

Sol: Let's convert all the number to base 10

$(11)_2 = (3)_{10}$

$(22)_3 = (8)_{10}$

$(33)_4 = (15)_{10}$

$(44)_5 = (24)_{10}$

Now, $3 + 8 + 15 + 24 = (50)_{10}$

Let's convert $(50)_{10}$ to base 6 number.

$$\begin{array}{r|l} 6 & 50 \\ 6 & 8 \\ \hline & 1 \end{array} \begin{array}{l} -2 \\ -2 \end{array}$$

a = 1
b = 2
c = 2

Ans. $(122)_6$

Q17 **From the following, determine the possible values of x – $(123)_5 = (x8)_y$**

Sol: For base 'b' number, digits range from '0' to 'b – 1'

Example: for the decimal number system (base 10), digit ranges from '0' to '9'

From the given equation following conditions are derived.

i) x < y

ii) y > 8

Converting all the number in base 10 is:

$(38)_{10} = xy + 8$

x * y = 30

x	y		
3	10	✓	
5	6	×	Not possible (y > 8) (Condition not satisfied)
1	30	✓	
2	15	✓	
10	3	×	Not possible (x < y)
6	5	×	Not possible (x < y)
30	1	×	Not possible (x < y)
15	2	×	Not possible (x < y)

(Condition not satisfied)

Ans.3, 1, 2

Q18 **Determine the base of the system.**
Roots of $x^2 - 11x + 22 = 0$ are '3' and '6'.

Sol: Product of roots of $ax^2 + bx + c = 0$ is $\frac{c}{a}$

Let 'b' be the base of the system

$$(3)_b \times (6)_b = \frac{(22)_b}{(1)_b}$$

$$\Rightarrow \quad (3)_{10} \times (6)_{10} = \frac{2 \times b + 2}{(1)_{10}}$$

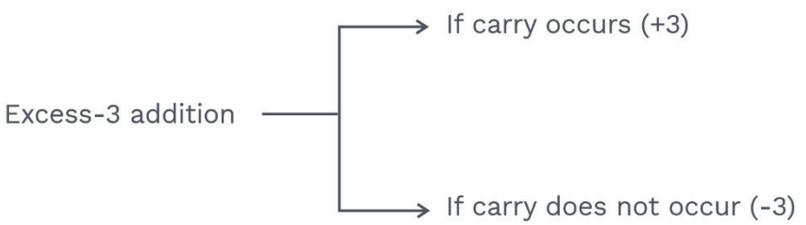
$$\Rightarrow \quad 18 = 2b + 2$$

$$\Rightarrow \quad 2b = 16$$

$$\boxed{b = 8}$$

Excess-3 code:
It is a non-weighted BCD code. Each excess-3 codeword is the corresponding 8-4-2-1 code word push 3 (0011). It is a self complementing code. It is a sequential code.
The excess-3 code has six invalid states 0000, 0001, 0010, 1101, 1110, and 1111.

Excess-3 addition:
The addition of two numbers in excess-3 is performed in a group of 4-bits starting from LSB and adding them column-wise. If there is a carry out from a group, then add 0011 to the sum term of the group and if there is no carry out then subtract 0011 from the sum term of that group.



SOLVED EXAMPLES

Q19 **Perform Excess-3 addition of i) and ii)**
i) 7 + 2 = ? **ii)** 7 + 3 = ?

Sol: **i)**

$7 \rightarrow 1010$
$+2 \rightarrow 0101$
$9 \rightarrow 1111 \longrightarrow$ No carry
$-\ 0011$
(1100) $\rightarrow$This is 9 in excess - 3

ii)

carry
1

07	0011	1010	- '07' in excess-3
+03	+0011	0110	- '03' in excess-3
10	0111	0000	
	-0011	+0011	
	0100	0011	

0100 ← 1 in excess – 3
0011 ← 0 in excess – 3

Q20 **Perform the following addition in excess – 3 code.**
a) 37 + 28 = ? **b) 247.6 + 359.4**

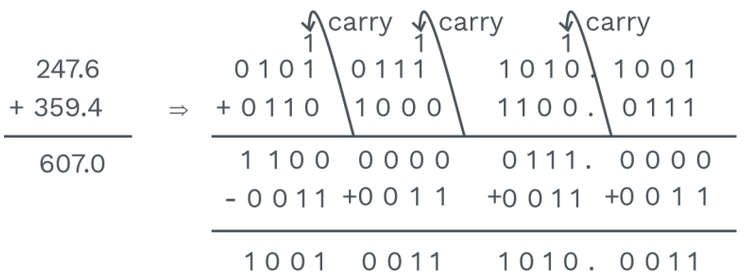
Sol: **a)**

carry
1

37	0110	1010	(37 in Excess - 3)
+28	+0101	1011	(28 in Excess - 3)
65	1100	0101	
	-0011	+0011	
	1001	1000	

1 0 0 1 ← '6' in excess – 3
1 0 0 0 ← '5' in excess – 3

b)



4.6 GRAY CODE

This type of code is also known as cyclic code/unit distance code/reflective code, as any two succeeding binary number equivalent gray codes differ in only one bit. It is a non-weighted code and is not well suited for arithmetic operations.

Grey Matter Alert!

Gray code is reflective. The n least significant bits for 2^n through $2^{n+1} - 1$ are the mirror images of those for 0 through $2^n - 1$. An N-bit gray code can be obtained by reflecting an N-1 bit code about an axis at the end of code and putting the MSB of 0 above the axis and MSB of 1 below axis.

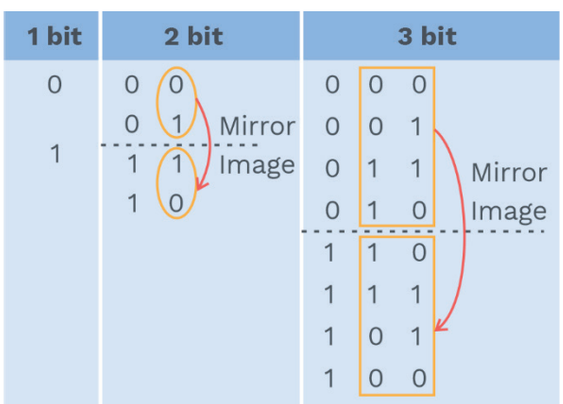
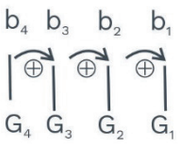


Table 4.1 Table for Gray Code

Binary to gray conversion:
Let the binary number be $b_4\, b_3\, b_2\, b_1$

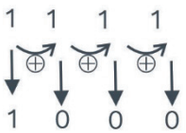


$$G_4 = b_4$$
$$G_3 = b_4 \oplus b_3$$
$$G_2 = b_3 \oplus b_2$$
$$G_1 = b_2 \oplus b_1$$

SOLVED EXAMPLES

Q21 **Convert $(1111)_2$ to gray code.**

Sol:

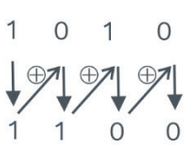


1000 in grey – Ans.

Grey to binary code:
Let the binary number be $b_4\, b_3\, b_2\, b_1$

$$b_4 = G_4$$
$$b_3 = b_4 \oplus G_3 = G_4 \oplus G_3$$
$$b_2 = b_3 \oplus G_2 = G_4 \oplus G_3 \oplus G_2$$
$$b_1 = b_2 \oplus G_1 = G_4 \oplus G_3 \oplus G_2 \oplus G_1$$

Example:



$(1\,1\,0\,0)_2$

4.7 COMPLEMENTARY NUMBER SYSTEM

Advantage → Subtraction can be performed using addition

Base b
- → (b-1)'s complement - diminished radix complement
- → b's complement - radix complement

⇒ Diminished radix complement of 'x' in base 'b':

$$\boxed{(b^n - 1) - x}$$

⇒ Radix complement of 'x' in base 'b':

$$\boxed{b^n - x}$$

Example:

Base 2
- → 1's complement $\boxed{\overline{x} = (2^n\text{-}1) - x}$
- → 2's complement $\boxed{\overline{x} = 2^n - x}$

Base 3
- → 2's complement $\boxed{\overline{x} = (3^n\text{-}1) - x}$
- → 3's complement $\boxed{\overline{x} = 3^n - x}$

Base 2 1's complement → $(2^n - 1) - x$

$x = 1011$

$\overline{x} = (2^4\text{-}1)\text{-}1011$

$\overline{x} = 1111$

$\quad - 1011$

$\quad 0100$

2's complement = 1's complement + 1

$= (2^n - 1) - x + 1$

$= 2^n - x$

$\quad 0100$

$+ \quad 1$

$\quad 0101$

Base – 16 15's complement → $(16^n - 1) - x$

16^{th} complement → $16^n - x$

$x = 1234$

$\overline{x}$ (in 15's complement) = $\begin{array}{r} FFFF \\ 1234 \\ \hline (EDCB)_{16} \end{array}$

$\overline{x}$ (in 16's complement) → $\begin{array}{r} EDCB \\ + \quad 1 \\ \hline (EDCC)_{16} \end{array}$

SOLVED EXAMPLES

Q22 **Find 9's complement of $(685)_{10}$.**

Sol: $(10^n - 1) - 685$

here n = 3

$(10^3 - 1) - 685 = 999 - 685 = 314$

** To find 10's complement = 9's complement + 1

= 314 + 1 = 315

Q23 **Perform the subtraction of the following numbers of base 10 using 9's complement:**
a) 97-23 **b) 23-97**

Sol: **a)** For the general case,

$$
\begin{aligned}
& x-y \\
&= x+(\bar{y}) \\
&= x+(10^2-1-y) \\
&= x-y+99 \\
&= (x-y)+99-100 \\
&= (x-y)-1 \\
&= (x-y)-1+1 \\
&= x-y
\end{aligned}
$$

→ carry exists [Discard carry, Add +1]
→ carry does not exist

$$97 - 23 = 97 + (-23)$$

9's complement of 23 →

$$\begin{array}{r} 99 \\ -23 \\ \hline 76 \end{array}$$

$$\begin{array}{r} 97 \\ +\ 76 \\ \hline (1)\,73 \end{array}$$

Discarded

$$\begin{array}{r} 73 \\ +1 \\ \hline 74 \end{array} \rightarrow \text{Ans}$$

b) 23 – 97

9's complement of 97 →

$$\begin{array}{r} 99 \\ -97 \\ \hline 02 \end{array}$$

23 + (-97)
= 23 + 02
= 25 (No carry)
↓ Taking 9's complement

99
- 25
74
→ (-74) Ans
Negating it

Q24 **Perform the following subtraction using 1's complement for the following binary numbers.**
a) 1011 – 0101 = ? **b) 111 – 10100**

Sol: **a)**

$1011 \leftarrow (11)_{10}$
$0101 \leftarrow (5)_{10}$
↓ Taking 1's complement
1010

1010
+ 1011
(1) 0101
+ 1
0110 - Ans
End around carry is discarded

b) x = 111
y = 10100
x–y = ?

$\bar{y} = 01011$

$x+\overline{y} = 00111 + 01011$

$= 10010$ (No carry)

↓ Taking 1's complement

$= 01101$

$= (13)_{10}$

Negating → –13 (Ans.)

Q25 **Perform the subtraction using b-1's complement for the following numbers in base 3.**
212 – 121 = ?

Sol: x = 212
y = 121
x–y = ?

$\overline{y} = 222 - 121 = 101$

$x+\overline{y} = 212$
$+\ 101$
$1\,020$
$+\ \ 1$
$(021)_3$ Ans.

End around carry is discarded

** in decimal = $(021)_3 = 3 \times 2 + 1 = (7)_{10}$

Q26 **Perform the following subtraction using b's complement for the numbers given in binary 0101 – 1011 = ?**

Sol: x = 0101
y = 1011
x-y = ?

$y = 1011$

↓ 1's complement

$\overline{y} = 0100$

↓ 2's complement

$\overline{y} = 0100+1$

$= 0101$

$x+(\overline{y}) = 0101$

$+\ 0101$

$1010 \rightarrow$ No carry

↓ 2's complement

0110

$-(x-y) = 0110 = (6)_{10}$

$x - y = -6$ Ans.

Q27 **Perform the following subtraction using b's complement for the given numbers in base 3.**
$(212)_3 - (121)_3 = ?$

Sol: $x = (212)_3$

$y = (121)_3$

$\overline{y} = 222$

$-\ 121$

$101 \leftarrow$ 2's complement

$+\ 1$

$102 \leftarrow$ 3's complement

$x+\overline{y} = 212 \rightarrow x$

$+\ 102 \rightarrow (b^n-y)$

$(1)021 \rightarrow (x-y)+b^n$

Discarded

Ans $(021)_3$

Grey Matter Alert!

Complementary number system – summary

The subtraction of two n-bit unsigned number

M-N ($N \neq 0$) in base "b" using diminished radix complement can be done as follows

1) Add M to (b-1)'s complement of N. this performs

$= M + (b^n - 1 - N)$

$= (M - N) + b^n - 1$

2) If $(M-N) \geq 1$, then sum will produce an end carry b^n, will be discarded and 1 is added which gives (M-N)

3) if $(M-N) < 1$, then sum does not produce an end carry. To obtain the answer in familiar form, take (b-1)'s complement of the sum and place a negative sign in front.

Grey Matter Alert!

Complementary number system – summary

The subtraction of two n-bit unsigned number

M-N ($N \neq 0$) in base "b" using radix complement can be done as follows

1) Add M to (b)'s complement of N. this performs

$=M + (b-N)$

$= (M-N) + b$

2) If $(M-N) \geq 0$, then sum will produce an end carry b^n. will be discarded which gives (M-N)

3) If $(M-N) < 0$, then sum does not produce an end carry. To obtain the answer in familiar form, take b's complement of the sum and place a negative sign in front.

$b^n - (b^n + (M - N)) - (-(M - N)) = (M - N)$

Note:

Let's take b = 2

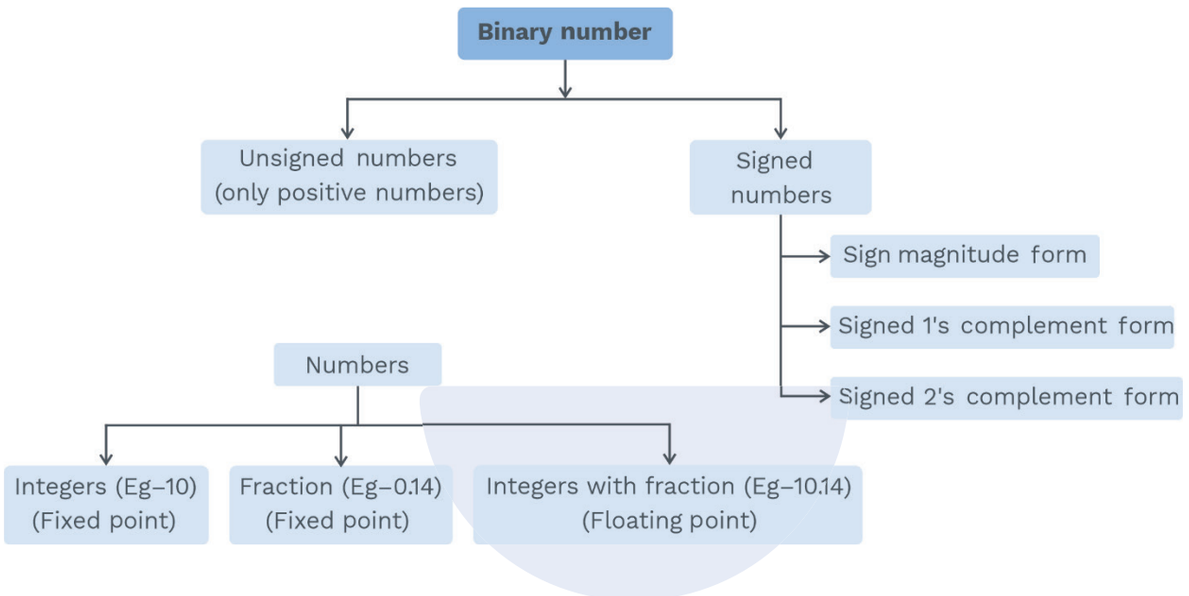
To perform M - N using radix complement we follow the steps as shown below

Step 1: Add M to 2's complement of N.

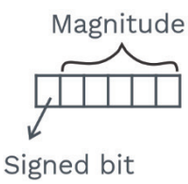
Step 2: If the sum produces carry then it is simply discarded.

Step 3: If the sum does not produce carry then take 2's complement of the sum and place a negative sign in front.

4.8 SIGNED NUMBER REPRESENTATION



Signed magnitude form:
If MSB is 1, the number is negative.
If MSB is 0, the number is positive.



Disadvantage:

i) Two combinations for '0' (zero)
ii) Signed bit (MSB) needs to be checked to take the decision for addition subtraction.
iii) Additional hardware required.

Signed 1's complement form:
Understanding through examples:

Example:

$(010)_2 = 2$

↓ 1's complement

$(101)_2 = -2$

Example:

$(001) = 1$

↓ 1's complement

$(110) = -1$

Disadvantage

Grey Matter Alert!

Q. Let's try to perform x-x using 1's complement (x' is in base 2)

Sol.

$X+\overline{X}$

$= X+(2^n-1-x)$

$= (x-x) + (2^n-1)$

$= 0 + \underbrace{(2^n-1)}$

Undesirable term

↓ Solution

End Around Carry is discarded if any.

2's complement form:

Example:

001 → 1

↓ 1's complement

110

+ 1

111 → -1 (in 2's complement)

Example:

010 → 2

↓ 1's complement

101

+ 1

110 → -2 (in 2's complement)

Weighted code:

→ Unsigned number is weighted representation.

→ Signed magnitude form is weighted code.

→ 2's complement is weighted code.

E.g: 1 0 0

$-2^2\ 2^1\ 2^0$

$= 1 \times (-2^2)$

$= -4$

→ 1's complement is non-weighted.

Examples of signed number representation:

	Unsigned	SM	1's	2's
0 0 0	0	0	0	0
0 0 1	1	1	1	1
0 1 0	2	2	2	2
0 1 1	3	3	3	3
1 0 0	4	0	-3	-4
1 0 1	5	-1	-2	-3
1 1 0	6	-2	-1	-2
1 1 1	7	-3	0	-1

Table 4.2

Ranges:

Number of bits	Signed magnitude	1's	2's
3	(0 to 3) & (0 to -3) OR (-3) to (+3)	(-3) to (+3)	(-4) to (+3)
4	(0 to 7) & (0 to –7) OR -7 to 7	(-7) to (+7)	(-8) to (+7)

Table 4.3

Grey Matter Alert!

In signed magnitude form, 1's complement form, there are two representation for 0(zero)

Generalisation for "n" bit numbers:

Signed Magnitude: 2^n → +ve No. → $\frac{2^n}{2} = 2^{n-1}\,[0-(2^{n-1}-1)]$

2^n → -ve No. → $\frac{2^n}{2} = 2^{n-1}\,[-(2^{n-1}-1)-0]$

$$\boxed{-\left(2^{n-1}-1\right) \text{ to } \left(2^{n-1}-1\right)}$$

2's complement form

$$\left(0 \text{ to } \left(2^{n-1}-1\right)\right) \& \left(-1 \text{ to } \left(-2^{n-1}\right)\right)$$

$$\boxed{-2^{n-1} \text{ to } \left(2^{n-1}-1\right)}$$

SOLVED EXAMPLES

Q28 **To represent 15 in 2's complement number system. What is the minimum number of bits required?**

Sol: $\left(2^{n-1}-1\right) \geq 15$

$\Rightarrow 2^{n-1} \geq 16$

$\Rightarrow n-1 \geq 4$

$\Rightarrow n \geq 5$

$n = 5$ Ans.

What is overflow? How does it happen?

$\Rightarrow$ In 2's complement, 4-bit range (–8 to 7)

"–16" cannot be represented using 4 bits in 2's complement representation. (overflow)

Why is overflow condition crucial in computers?

⇒ While doing arithmetic, the computer stores the result in a register.

If the register is of size 8 bit, the result should not exceed (–128 to 127).

Q29 **–30 is to be represented in 2's complement number representation. What is number of bits (minimum) required?**

Sol: $-2^{n-1} \leq -30$

$2^{n-1} \geq 30$

$n - 1 \geq 4.5$

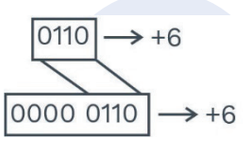
$\boxed{n \geq 5.5}$ Minimum number of bits = 6

Sign bit extension

Sign bit extension expands a smaller bit number without changing Its actual value.

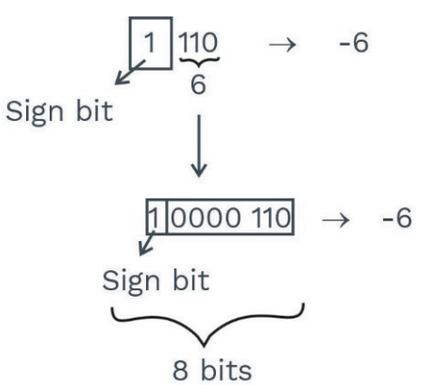
1) Unsigned number

Suppose we want to represent 0110 in 8 bits.

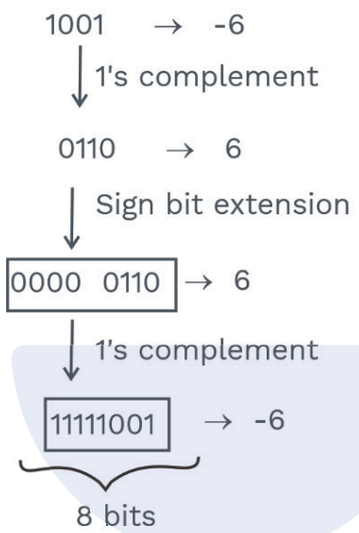


2) Sign magnitude

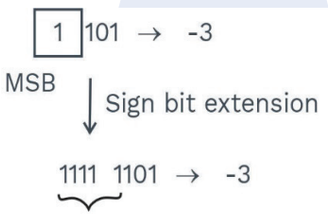
The first bit (MSB) denotes the sign



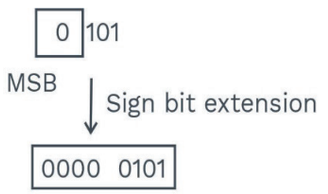
3) 1's Complement
Example:



4) 2's Complement
Example:



Example:



Rack Your Brain

A number representation follows 2's complement representation form. What will be decimal equivalent (base 10) of $(FFF3)_{16}$?

Previous Years' Question

The smallest integer that can be represented by an 8-bit number in 2's complement form is **(GATE-2013)**

1) –256
2) –128
3) –127
4) 0

Sol: 2)

Previous Years' Question

P is a 16 bit signed integer. The 2's complement representation of P is $(F\ 87\ B)_{16}$. The 2's Complement representation of 8 × P is **(GATE-2010)**

1) $(C3D8)_{16}$
2) $(187B)_{16}$
3) $(F878)_{16}$
4) $(987B)_{16}$

Sol: 1)

Previous Years' Question

Let r denote number system radix. The only value(s) of r that satisfy the $\sqrt{121_r} = 11_r$ is/are: **(GATE-2008)**

1) decimal 10
2) decimal 11
3) decimal 10 and 11
4) any value > 2

Sol: 4)

Rack Your Brain

A 2's complement number N = $P_3\ P_2\ P_1\ P_0$ is transformed as $P_3\ P_3\ P_3\ P_2\ P_1\ P_0\ 1$. Which of the following operation is performed on 'N'?

1) 2N+1
2) 2N
3) N + 1
4) 3N + 1

Overflow:

Definitions

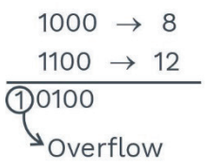
When two numbers of n digits each are added and the sum occupies n + 1 digits, we say that an overflow occurs. (OR) Overflow is said to occur when magnitude of a number exceeds the range allowed by the size of bit-field.

⇒ Addition of two 4-bit numbers might result in 5 bit

$$\begin{array}{r} 1000 \\ +\ 1100 \\ \hline 10100 \end{array}$$

Let's discuss about overflow for different classifications of numbers:

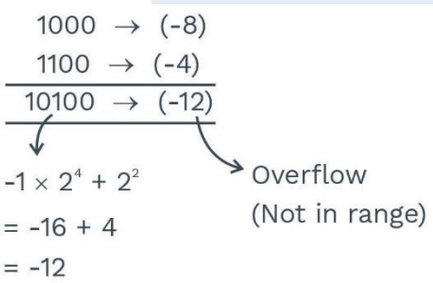
1) Unsigned number:



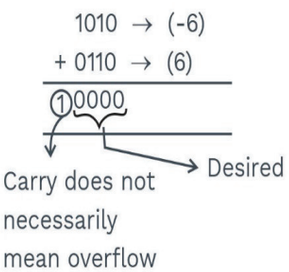
For 4 bits number, number ranges $\left(0-\left(2^4-1\right)\right)$

2) Signed 2's complement:

For 4 bit number, range → (−8 to 7)



Let's look at another example:



Let's combine whatever we learned about the Overflow concept till now:

1) Overflow condition for unsigned numbers
- Carry always indicates overflow

2) Overflow condition for signed numbers.

Case 1 When two positive numbers are added, the result is negative.

Example:

$$\begin{array}{l} 0111 \rightarrow 7 \\ 0001 \rightarrow 1 \\ \hline 1000 \rightarrow 8 \end{array}$$

-ve (Overflow)

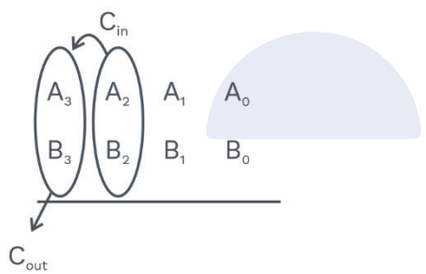
Case 2 When 2 negative numbers are added and the result is +ve.

Example:

$$\begin{array}{l} 1000 \rightarrow (-8) \\ 1100 \rightarrow (-4) \\ + \\ \hline 10100 \\ \hline \end{array}$$

+ve (Overflow)

SHORTCUT (For signed numbers)



$C_{out} \neq C_{in}$ (overflow)
$C_{out} = C_{in}$ (no overflow)

Rack Your Brain

In a Number system, range of numbers are –3 to 3 represented as C, B, A, 0, 1, 2, 3 What will be $(102)_{10}$ in this system?

Previous Years' Question

When two 8-bit numbers A_7 .... A_0 and B_7 .... B_0 in 2's complement representation (with A_0 and B_0 as the least significant bits) are added using ripple carry adder. The sum bits obtained are S_7 .... S_0 and the carry bits are C_7 .... C_0 An overflow is said to have occurred if **(GATE-2008)**

1) The carry bit C_7 is 1

2) All the carry bits (C_7 .... C_0) are 1

3) ($A_7B_7S_7 + A_7'B_7;S_7$) is 1

4) ($A_0.B_0.S_0' + A_0;B_0;S_0$) is 1

Sol: 3)

4.9 ARITHMETIC OPERATIONS

1) Binary addition:

The rules for binary addition are the following:

0 + 0 = 0
0 + 1 = 1
1 + 0 = 1
1 + 1 = 10 i.e. 0 with a carry of 1

SOLVED EXAMPLES

Q30 **Add the binary numbers 1101.101 and 111.011**

Sol:

$8\ 4\ 2\ 1$	$2^{-1}2^{-2}2^{-3}$
$1\ 1\ 0\ 1$.	$1\ 0\ 1$
$1\ 1\ 1$.	$0\ 1\ 1$
+	
$1\ 0\ 1\ 0\ 1$.	$0\ 0\ 0$

In the 2^{-3}'s column, 1+1 = 0 with a carry of 1 to the 2^{-2} column
In the 2^{-2}'s column, 0+1+1 = 0 with a carry of 1 to the 2^{-1} column
In the 2^{-1}'s column, 1+0+1 = 0 with a carry of 1 to the 1's column
In the 1's column, 1+1+1 = 1 with a carry of 1 to the 2's column
In the 2's column, 0+1+1 = 0 with a carry of 1 to the 4's column
In the 4's column, 1+1+1 = 1 with a carry of 1 to the 8's column
In the 8's column, 1+1 = 0 with a carry of 1 to the 16's column

2) Binary subtraction:

Binary subtraction is similar to decimal subtraction and the rules for it are given below:

i) 0–0 = 0
ii) 1–1 = 0
iii) 1–0 = 1
iv) 0–1 = 1 with a borrow of 1.

SOLVED EXAMPLES

Q31 **Subtract $(111.111)_2$ from $(1010.01)_2$**

Sol:

$$\begin{array}{r} 8\,4\,2\,1 \quad 2^{-1}2^{-2}2^{-3} \\ 1\,0\,1\,0 \;.\; 0\,1\,0 \\ \quad 1\,1\,1 \;.\; 1\,1\,1 \\ + \\ \hline 0\,0\,1\,0 \;.\; 0\,1\,1 \end{array}$$

In the 2^{-3} column, a 1 can not be subtracted from a 0. So, borrow a 1 from the 2^{-2} column making the 2^{-2} column 0. The 1 borrowed from the 2^{-2} column becomes 10 in the 2^{-3} column. Therefore, in the 2^{-3} column, 10–1 = 1
In the 2^{-2} column, a 1 can not be subtracted from a 0, borrow a 1 from the 2^{-1} column, but it is also a 0. So borrow a 1 from the 1's column. That is also a 0, so borrow a 1 from the 2's column making the 2's column 0.

This 1 borrowed from the 2's column becomes 10 in the 1's column. Keep one 1 in the 1's column, bring the other 1 to the 2^{-1} column, which becomes 10 in this column.

Keep one 1 in the 2^{-1} column and bring the other 1 to the 2^{-2} column, which becomes 10 in this column. Therefore,

In the 2^{-2}'s column 10–1 = 1
In the 2^{-1}'s column 1–1 = 0
In the 1's column 1–1 = 0

Now, in the 2's column, a'1' cannot be subtracted from a 0, so borrow a 1 from the 4's column. But 4's column has a 0, so borrow a 1 from 8's column, making 8's column 0 and bring it to the 4's column. It becomes 10 in 4's column. Keep one 1 in 4's column and bring the second 1 to the 2's column making it 10 in the 2's column.

Therefore,

In the 2's column = 10–1 = 1
In the 4's column = 1–1 = 0
In the 8's column = 0–0 = 0
Here, the result is $(0010.011)_2$

3) Binary multiplication:

Following are the multiplication rules :

$$0 \times 0 = 0 \; ; \; 0 \times 1 = 0 \; ; \; 1 \times 0 = 0 \; ; \; 1 \times 1 = 1$$

- For multiplication, a multiplier Q and a multiplicand M are required. Let's assume that the multiplier Q = $q_{n-1}\, q_{n-2} \ldots q_1 q_0$ and the multiplicand M = $m_{n-1}\, m_{n-2} \ldots m_1 m_0$.
- To multiply two unsigned numbers, there exists a very simple method known as the paper and pencil method.

Example: Consider the multiplication of two unsigned numbers 14 and 10.

1 1 1 0 (14)	multiplicand (M)
1 0 1 0 (10)	multiplier (Q)
0 0 0 0	(Partial product)
1 1 1 0	(Partial product)
0 0 0 0	(Partial product)
1 1 1 0	(Partial product)
1 0 0 0 1 1 0 0	

10001100 (140) Final product (P)

- In this method, n partial products are calculated.
- If the multiplier bit is 0, then there is no need to compute that partial product.

Partial sum method:

Let's take two decimal numbers $(5)_{10}$ and $(6)_{10}$:
Binary equivalent of $(5)_{10}$ = 0101
Binary equivalent of $(6)_{10}$ = 0110
Using partial sum method:
Multiplicand (M) = 0101
Multiplier (Q) = 0110

```
    0 1 0 1
  × 0 1 1 0
  ---------
    0 0 0 0
  0 1 0 1
0 1 0 1
0 0 0 0
---------
0 0 1 1 1 1 0
```

So, if we multiply two n bit numbers, then in the worst case, the space required to store the result in 2n.

Multiplication using registers:

Multiplying (0101) by (0110) ⟶ 0101 ×0110

0	0	0	0	0	0	0	0

Multiplying 0 with (0101) and performing 1 arithmetic right shift operation after addition.

Similarly, in step 2, multiplying 1 with (0101) and performing 1 arithmetic right shift after addition, in step 3, multiplying 1 with (0101) and performing 1 arithmetic right shift after addition and in step 4 multiplying 0 with (0101) and performing 1 arithmetic right shift after addition.

Step 1:

	0	0	0	0	0	0	0	0
+	0	0	0	0				
	0	0	0	0	0	0	0	0

Shift:

0	0	0	0	0	0	0	0

Step 2:

+	0	1	0	1				
	0	1	0	1	0	0	0	0

Shift:

0	0	1	0	1	0	0	0

Step 3:

+	0	1	0	1				
	0	1	1	1	1	0	0	0

Shift:

0	0	1	1	1	1	0	0

Step 4:

+	0	0	0	0				
	0	0	1	1	1	1	0	0

Shift:

0	0	0	1	1	1	1	0

(final result)

Note:

For every bit in the multiplier (Q). The Shift will be performed, and for every 1 in the multiplier addition will be performed.

So, the number of addition required = The number of 1's in the multiplier.

the number of shifts required = Number of bits in the multiplier.

eg: Q = 011111110

Number of addition required = 7

Number of shifts required = 9

Fixed point sign multiplication:

- Multiplication cannot be extended for both signed and unsigned operands because the sign bit will get disturbed during the computation.
- Hence, a separate algorithm is required for fixed-point signed multiplication, which is known as Booth's algorithm.
- Booths algorithm gives a fast response for the multiplication of fixed point signed numbers.

eg: 1

x = 10, y = 3

x * y = 30

x = 1010 y = 0011

x = 1 0 1 0

y = 0 0 1 1

```
    1 0 1 0
  1 0 1 0
0 0 0 0
0 0 0 0
---------------
0 0 1 1 1 1 0
```

0 0 1 1 1 1 0 in decimal is 30

eg: 2

x = –6, y = 3

x * y = –18

x = $(1010)_2$ (in 2's complement)

y = 3 = $(0011)_2$

x = 1 0 1 0

y = 0 0 1 1

```
    1 0 1 0
  1 0 1 0
0 0 0 0
0 0 0 0
---------------
0 0 1 1 1 1 0
```

0 0 1 1 1 1 0 in decimal is 30.

But result of x * y should be –18.

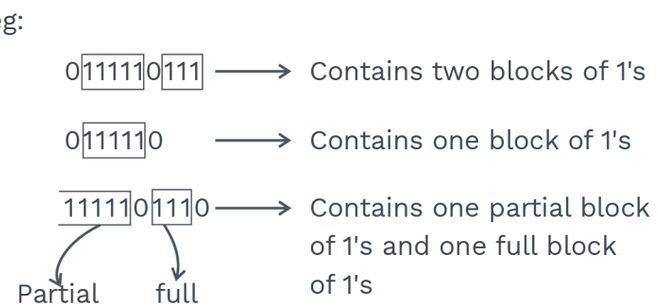
Why need Booth's algorithm?

It is a multiplication algorithm that is used to multiply two signed binary numbers(using 2's complement representation) in an efficient way using very less number of addition or subtraction operations.

Booths algorithm:

1) In this algorithm, a binary pattern of a multiplier is treated as blocks of 1's.

eg:



2) Only beginning and ending 1 of the block will result in an arithmetic operation.
3) Intermediate 1's in the block does not require any arithmetic operation. Hence Booths algorithm saves time as well as space.
4) Every full block needs 1 addition and 1 subtraction, but a partial block will need only subtraction.
5) Booths algorithm uses the partial sum concept, and the partial sum is arithmetically right shifted. Thus, it will protect the sign bit.

The idea behind booth's algorithm:
Booth's algorithm is based on one observation. Suppose we have binary number 011110 where a sequence of 1's is present.

Then this sequence (block) of 1's can be written as the difference between two numbers, where these two numbers are in the power of 2.

Booths notation:

- For the multiplier pattern in a booths notation, if i and j represent the beginning and ending position of the block of 1's.

The value is given as:

$$V = \sum_{m=1}^{K} \left(+2^{j_m+1} - 2^{i_m} \right)$$

Thus, if M(multiplicand) × Q(multiplier) is given where Q = 011110
then partial sum method requires 6 shifts and 4 addition operations, whereas using Booth's algorithm, we require only 1 addition and 6 shift, as:

$$M \times \overset{Q}{\boxed{0\,1\,1\,1\,1\,0}} \text{ can be written as}$$

$$= (2^5)\, M + (2^1)(-M)$$

Example:

1)

$$\begin{matrix} 2^5 & 2^4 & 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 1 & 1 & 1 & 0 \end{matrix} \text{ can write as } (+ 2^5 - 2^1) = 30$$

2)

$$\begin{matrix} 2^3 & 2^2 & 2^1 & 2^0 \\ 0 & 1 & 1 & 0 \end{matrix} = (+ 2^3 - 2^1) = 6$$

3)

$$0\ 1\ 1\ 1\ 1 \ldots\ldots 1\ 1 \ldots\ldots 1\ 0 = (+ 2^{j+1} - 2^i)$$

2^j $\qquad$ 2^i

Booths recording pattern:

1) It contains (–1), 0, and (+1) as coefficients, and all weights are positive.

2) For all 'i' positions, the coefficient is (–1).
For all 'j+1' positions, the coefficient is (+1).
The rest of the places are 0's.

3) It is useful for finding:

i) The value of the number.

ii) The number of arithmetic operations as well as the number of shift operations.

e.g. 1 :

0 1 1 1 1 0 1 1 1

j_2 i_2 j_1 i_1

8	7	6	5	4	3	2	1	0
+1	0	0	0	–1	+1	0	0	–1

$\Rightarrow$ $2^8 (+1) + 2^4 (-1) + 2^3 (+1) + 2^0 (-1)$

$\Rightarrow$ $256 - 16 + 8 - 1$

$\Rightarrow$ 247

Number of additions required = 2

Number of subtractions required = 2

Number of shift operations required = 8 + 4 + 3 + 0 = 15

e.g. 2:

0 1 1 1 1 1 0 0

7	6	5	4	3	2	1	0
+1	0	0	0	0	–1	0	0

$\Rightarrow \quad 2^7 (+1) + 2^2 (-1)$

$\Rightarrow \quad 128 - 4$

$\Rightarrow \quad 124$

Number of additions required = 1

Number of subtractions required = 1

Number of shift operations required = 7 + 2 = 9

Example 3 : Give Booths recording pattern for (–57).

Sol: (–57) in 2's complement binary number = $(1000111)_2$

6	5	4	3	2	1	0
–1	0	0	+1	0	0	–1

$\Rightarrow \quad 2^6(-1) + 2^3(+1) + 2^0(-1)$

Number of addition required = 2

Number of subtraction required = 2

Number of shift operation required = 6 + 3 + 0 = 9

Example: 4 : For which of the following, the Booth's algorithm gives a better performance?

1) 00011111

2) 10101010

3) 11110111

4) 01110111

Sol: Option 1), because it requires an overall less number of additions and subtractions.

Option 1):

0 0 0 11111

1 full block of 1's

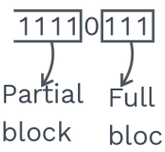
It requires 1 addition and 1 subtraction

Option 2):

10101010

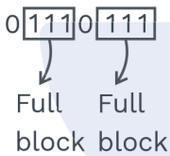
It contains 3 full block and 1 partial. Thus it requires 3 addition and 4 subtraction.

Option 3):



It contains 1 partial and 1 full block.
Thus it requires 1 addition and 2 subtraction.

Option 4):



It contains 2 full blocks.
Thus it requires 2 addition and 2 subtractions.

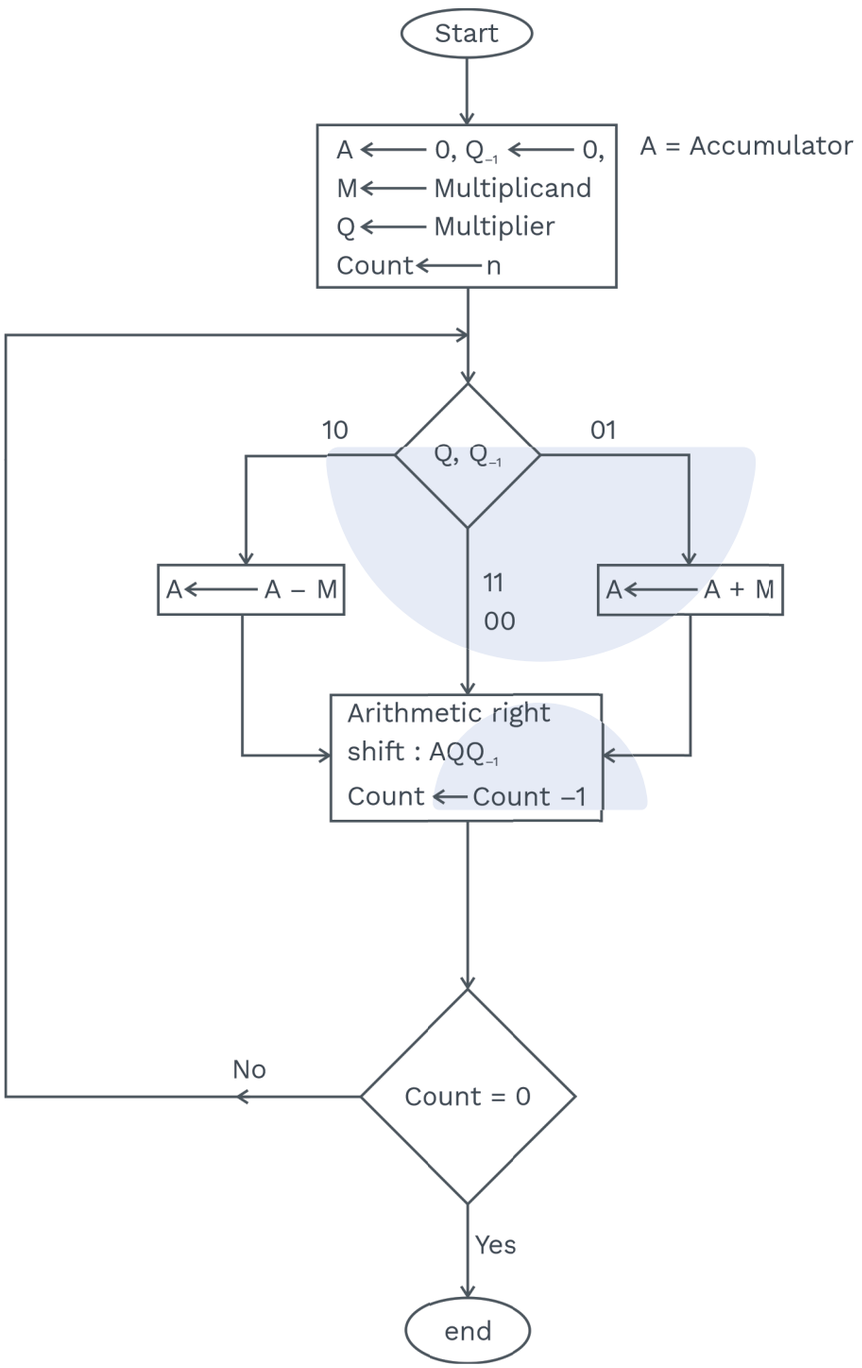


Fig. 4.1 Flow Chart of Booth's Algorithm

Booth's algorithm example:

Example 1: Consider the multiplication of two numbers –5 and –4 using Booth's algorithm
Here, M(multiplicand) = –5 and Q(multiplier) = –4
M in 2's complement = 1011
Q in 2's complement = 1100

- Size of accumulator register = Size of multiplicand register
- Initially, the accumulator and Q_{-1} value is initialised to 0(zero).

Operation	A	Q	Q_{-1}	
	0000	1100	0	(Least significant bit of Q = 0, Thus the value of QQ_{-1} = 00 performs arithmetic right shift (ARS))
1) ARS	0000	0110	0	(Least significant bit of Q = 0, Thus the value of QQ_{-1} = 00 perform arithmetic right shift (ARS))
2) ARS	0000	0011	0	(Now, LSB of Q = 1, Thus the value of QQ_{-1} = 10 perform A ¬ A–M and then ARS (arithmetic right shift))
A – M = A + 2's complement of M = 0000 + 0101 = 0101				
3) A – M ARS	0101 0010	0011 1001	0 1	(Least significant bit of Q = 1, Thus the value of QQ_{-1} = 11 perform arithmetic right shift (ARS))
4) ARS	0001	0100	1	

Table 4.4

Product = AQ = 00010100 = (+20)

Example 2: Consider the multiplication of two numbers –5 and 4 using Booth's algorithm
Here, M(multiplicand) = –5 and Q(multiplier) = 4 = $(0100)_2$
M in 2's complement = $(1011)_2$

Operation	A	Q	Q_{-1}	
	0000	0100	0	(Least significant bit of Q = 0, Thus the value of QQ_{-1} = 00 perform arithmetic right shift (ARS))
1) ARS	0000	0010	0	(Least significant bit of Q = 0, Thus the value of QQ_{-1} = 00 perform an arithmetic right shift (ARS))
2) ARS	0000	0001	0	(Now, LSB of Q = 1, Thus the value of QQ_{-1} = 10 perform A ¬ A–M and then ARS (arithmetic right shift))
A – M = A + 2's complement of M = 0000 + 0101 = 0101				
3) A – M ARS	0101 0010	0001 1000	0 1	(Now, LSB of Q = 0, Thus the value of QQ_{-1} = 01 perform A ¬ A+M and then ARS (arithmetic right shift))
A + M = 0010 + 1011 = 1101				
4) A + M ARS	1101 1110	1000 1100	1 0	

Table 4.5

Final result of multiplication = AQ = 11101100 = (–20)

Note:

The number of steps that need to be performed in Booth's algorithm is equal to the number of bits in the multiplier.

Advantages and disadvantages of Booth's algorithm:

Advantages:

1) In general, the number of operations required will be reduced.
E.g., 0111110 required 1 addition, 1 subtraction and 7 shift operation.

2) It is used to perform signed multiplications.
3) It gives better performance if negative numbers are involved.

Disadvantages:
When there are pairs of alternatives 0's and 1's in the multiplier, Booth's algorithm gives the worst performance.
e.g., 01010101010101

Previous Years' Question

The two numbers given below are multiplied using the Booth's algorithm.

Multiplicand:	0101	1010	1110	1110
Multiplier:	0111	0111	1011	1101

How many additions/subtractions are required for the multiplication of the above two numbers? **(GATE IT-2008)**

1) 6
2) 8
3) 10
4) 12
Sol: 2)

4) Octal addition:
Octal addition can be performed by first converting an octal number to its binary equivalent and then applying binary arithmetic rules. Octal subtraction can also be performed by first converting it into binary and then using 1's complement or 2's complement method of binary arithmetic for subtraction.

SOLVED EXAMPLES

Q32 **Perform the given addition (write base as 8 for each bracket).**
a) 25 + 13 **b) (27.5) + (74.4)**

Sol: **a)**

25	010	101
+ 13	001	011
40	100	000

$5 + 3 = (8)_{10} = (10)_8$

$2 + 1 + 1 = (4)_{10} = (4)_8$

b)

$$\begin{array}{rcl} (27.5)_8 & = & 010\ \ 111\ .\ 101 \\ +(74.4)_8 & = & 111\ \ 100\ .\ 100 \\ \hline (124.1)_8 & = & 1010\ 100\ .\ 001 \end{array}$$

Q33 **Subtract:**

a) $(45)_8$ from $(66)_8$
b) $(73)_8$ from $(25)_8$

Sol: Using 8-bit representation and 2's complement method.

a)

$$\begin{array}{rcl} (66)_8 & = & 00\ \ 110\ \ 110 \\ -(45)_8 & = & +\ 11\ \ 011\ \ 011\ (2\text{'s complement of } 45)_8 \\ \hline (21)_8 & = & (1)00\ 010\ 001 \end{array}$$

Discard

The final carry is ignored since the answer is positive.

b)

$$\begin{array}{rcl} (25)_8 & = & (00\ \ 010\ \ 101)_2 \\ -(73)_8 & = & +\ (11\ \ 000\ \ 101)_2\ (2\text{'s complement of } 73)_8 \\ \hline (-48)_8 & = & 11\ \ 011\ \ \ 010_2\ \ \ (\text{Answer is negative}) \end{array}$$

2's complement of $(11011010)_2 = 00100110 = -48_8$

5) Hexadecimal arithmetic:

Hexadecimal arithmetic rules are similar to the rules of binary or decimal number system arithmetic rules.

A hexadecimal number is first converted to its binary equivalent and then binary arithmetic rules can be used for any operation.

Hexadecimal subtraction can be performed either by converting it into its binary equivalent and then using 1's complement or 2's complement method of binary arithmetic for subtraction or by using 15's complement and 16's complement method.

SOLVED EXAMPLES

Q34 **Add $(6E)_{16}$ and $(C5)_{16}$**

Sol:

$$\begin{array}{rl} (6E)_{16} & = \ 0110 \ \ 1110 \\ +\,(C5)_{16} & = \ 1100 \ \ 0101 \\ \hline (133)_{16} & = 10011 \ \ 0011 \end{array}$$

Q35 **Subtract $(7B)_{16}$ from $(C4)_{16}$**

Sol:

$$\begin{array}{rl} (C4)_{16} & = \ 1100 \ \ 0100 \\ -\,(7B)_{16} & = \ 1000 \ \ 0101 \ \text{(2's complement form of } 7B_{16}) \\ \hline (49)_{16} & = \ 1 \ 0100 \ \ 1001 \end{array}$$

Discard

The final carry is ignored since the answer is positive.

Q36 **Subtract $(5D)_{16}$ from $(3A)_{16}$**

Sol:

$$\begin{array}{rl} (3A)_{16} & = \ \ 0011 \ \ 1010 \\ -\,(5D)_{16} & = +1010 \ \ 0011 \ \text{(2's complement form of } 5D)_{16} \\ \hline (-23)_{16} & = (1101 \ \ 1101)_2 \end{array}$$

$\therefore$ No carry, answer is negative.

Q37 **Convert the following numbers with the given radix to decimal and then to binary.**

a) $(4433)_5$ **b) $(1199)_{12}$** **c) $(5654)_7$**

Sol:

a) $(4433)_5$

i) $(\ )_5$ to $(\ )_{10}$

$= 5^3 \times 4 + 5^2 \times 4 + 5^1 \times 3 + 5^0 \times 3$

$= 125 \times 4 + 25 \times 4 + 5 \times 3 + 1 \times 3$

$= 500 + 100 + 15 + 3$

$= (4433)_5 \Rightarrow (618)_{10}$

i) $(\ )_{10}$ to $()_2$

$(618)_{10} = (1001101010)_2$

b) $(1199)_{12}$

i) $(\ )_{12}$ to $(\ )_{10}$

$= 12^3 \times 1 + 12^2 \times 1 + 12^1 \times 9 + 12^0 \times 9$

$= 1728 + 144 + 108 + 9$

$= (1199)_{12} = (1989)_{10}$

ii) $(\ )_{10} - (\ )_2$

$(1989)_{10} = (11111000101)_2$

c) $(5654)_7$

i) $(\ )_7$ to $(\ )_{10}$

$= 7^3 \times 5 + 7^2 \times 6 + 7^1 \times 5 + 7^0 \times 4$

$= 343 \times 5 + 49 \times 6 + 7 \times 5 + 1 \times 4$

$(5654)_7 = (2048)_{10}$

ii) $(\ )_{10}$ to $(\ )_2$

$(2048)_{10} = (100000000000)_2$

Rack Your Brain

The following arithmetic operations is valid in at least one number system. Find possible base for each of the following operation.

a) $1234 + 5432 = 6666$

b) $\frac{41}{3} = 13$

c) $\frac{33}{3} = 11$

d) $44 + 32 + 14 + 23 = 223$

e) $\frac{302}{20} = 12.1$

f) $\sqrt{41} = 5$

Rack Your Brain

The number of bits required to assign binary roll number to a class of 60 students is -

1) 5 **2)** 6 **3)** 7 **4)** 8

Previous Years' Question

If $N^2 = (7601)_8$ where 'N' is a positive integer, then the value of N is: **(ISRO CS-2008)**

1) $(241)_5$

2) $(143)_6$

3) $(165)_7$

4) $(39)_{16}$

Sol: 2)

6) BCD addition:

BCD addition is performed in a group of 4-bits starting from LSB, and adding them column-wise. If there is no out-carry from a group and sum term is also not an illegal code, then there is no need to do anything further.

If there is an out carry from a group or the sum term is an illegal code, then $(6)_{10}$ is added to the sum term of that group and the resulting carry is added to the next group.

SOLVED EXAMPLES

Q38 **Perform the following decimal addition in 8421 code**

a) 25 + 13 **b) 679.6+536.8**

Sol: **a)**

25	=	0010	0101	(25 in BCD)
+ 13	+	0001	0011	(13 in BCD)
38		0011	1000	

→ No carry, no illegal code.

b)

679.6 ⇒ 0110 0111 1001 . 0110
+ 536.8 ⇒ 0101 0011 0110 . 1000

1216.4

1011 1010 1111 . 1110 ← All are illegal codes
0110 0110 0110 . 0110

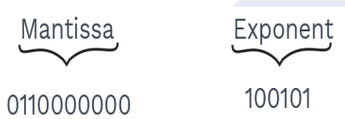
(1)0001 0000 0101 . 0100
+1 +1 + 1 + 1

0001 0010 0001 0110 . 0100
1 2 1 6 4

Sol: 1216.4

4.10 FLOATING-POINT NUMBERS

In the decimal system, very large and very small numbers are expressed by stating a number (mantissa) and an exponent of 10. Examples are 2.78×10^{24} and 3.98×10^{-26}. Binary numbers can also be expressed in the same notation by stating a number and an exponent of 2. However, the format for a computer may be as shown below:



In this machine, the 16-bit word consists of two ports, a 10-bit mantissa, a 6-bit exponent. The mantissa is in 2's complement form. So the MSB can be thought of as the sign bit.

Floating–point is always interpreted to represent in the following form:

$$m \times r^e$$

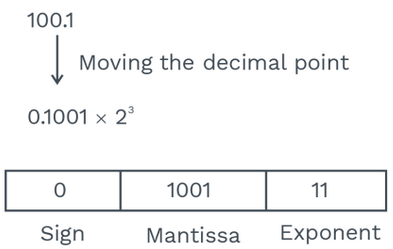
m = mantissa
e = exponent

- Mantissa can be integer or fractional.
- Exponent indicates the actual position of the decimal point.

Why do we need floating-point?

To represent a very large or a very small fraction of data, large memory space is required so to represent them within a less amount of memory space, floating point representation is used.

Let's represent $(4.5)_{10}$ in the above format:



$$\overset{-1\ -2\ -3\ -4}{0.1001} \times 2^3$$

$$= (2^{-1} + 2^{-4})\ 2^3$$

$$= 2^2 + 2^{-1}$$

$$= 4 + 0.5$$

$$= 4.5$$

Normalisation:

A floating-point number representation is not unique in the sense that, for the same number N, there are multiple representations.

Example: $X = 0.004 \times 10^{15}$

$X = .04 \times 10^{14}$

$X = .4 \times 10^{13}$

For hardware implementation, it is desirable to have a unique representation for a number. One way to achieve this is to represent all floating-point numbers having non-zero leading digits in their most significant position. A number expressed in this form is called normalised.

Normalisation is a process of eliminating leading zeroes from the mantissa. Its main objective is to maximise precision, given a limited number of bits.

Example: $X = 0.003 \times 10^7$ (not normalised number)

$X = 0.3 \times 10^5$ (normalised number)

Note:

- In the hexadecimal number system, mantissa should start with a non-zero leading digit (other than zero).
- In the binary number system, mantissa should start with one.

1) Explicit normalisation:

Move the radix point before the most significant '1'

Let's consider the number – $(100.1)_2$

↓ Moving the radix point

0.1001×2^3 (Normalized form)

So, given a binary representation of the form:

We can convert it into decimal:

$$\boxed{(-1)^S \times 0.M \times 2^E}$$

Biasing:

Definition

Biasing of exponent is required in order to convert negative exponent to postive exponent

We already know, for 'n' bits number, in 2's complement form, the range of number

$$\rightarrow \left(-2^{n-1} \text{ to } 2^{n-1} - 1\right)$$

Here, $2^{(n-1)}$ is the bias for n bit exponent.

E(Unsigned biased exponent) = e + bias
Let's consider 0.101×2^{-2} (Normalised from)
n = 5 (5 bits number)
Range of number → (–16 to + 15)

$$E = -2$$

Here, bias = $2^{n-1} = 2^{5-1} = 16$
E(Unsigned biased exponent) = –2 + 16 = 14

S	E(5)	M
0	01110	1010

To represent it in decimal:

$$(-1)^S \times 0.M \times 2^{E-Bias}$$

E → Biased exponent.

Grey Matter Alert!

Implicit Normalization gives more precission than Explicit Normalization.

2) Implicit normalisation:

Move the radix point after the most significant 1.
Let's consider 100.111
Let mantissa = 5 bits, exponent E = 4 bits
After normalisation the number will be 1.00111×2^2
Bias = 8, true exponent (e) = 2

$$E = 2 + 8 = 10$$

To store floating-point numbers, the register is partitioned as follows:

S	E(4)	M(5)
0	1010	00111

To get the corresponding decimal number, use the below expression.

$$(-1)^S \times 1.M \times 2^{E-bias}$$

SOLVED EXAMPLES

Q39 **Consider an IBM system which uses 32 bit register for representing the floating point number. The mantissa is implicitly normalized. The exponent is in excess –128 form and the base of the system is 2. What will be the hexadecimal pattern for storing the value $(127.5)_{10}$.**

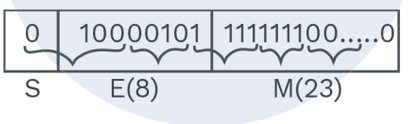
Sol: Value = $(-1)^S * (1.M) * 2^{E-biased}$

given the number is positive, S = 0

$$(127.5)_{10} = (1111111.1)_2 = (1.1111111)_2 \times 2^6$$

M = 1111111000.....0

E = 128 + 6 = 134 = $(10000110)_2$



Ans. $0 \times 37FF0000$

Rack Your Brain

What is the difference between 1st smallest positive and 2nd smallest positive number given

The drawback of floating-point representation:

1) Floating-point number distribution is not uniform since there is an unequal gap between any two consecutive numbers.
2) We can not represent a very small number (0) and a very large number (∞).
3) Floating point numbers are exposed to errors like overflow and underflow.

Introduction to IEEE standards:

IEEE standards:

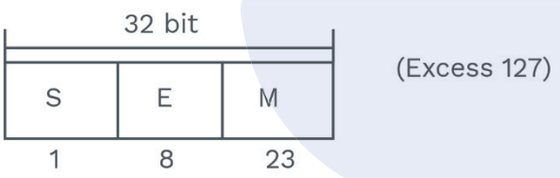
1985: single, double precision

2008: half, quadruple precision

1985 standard:

- The base of the system is 2
- Two kinds of precision [single precision – 32 bits, double precision-64 bits]
- The floating-point number can be represented in:
  * Fractional form
  * Implicit normalised form
- Certain mantissa and exponent combinations do not represent any number (NAN → Not A Number)
- It is used to represent '0' p and '∞'

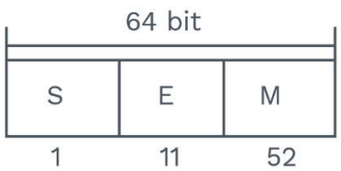
Single precision:



S(1)	E(8)	M(23)	Value
0 or 1	0000 0000 E = 0	000 ... 0 M = 0	± 0
0 or 1	1111 1111 E = 255	0000 ... 0 M = 0	± ∞
0 or 1	1 ≤ E ≤ 254	M = × × ... ×	Implicit normalized form $(-1)^s \times 1.M \times 2^{E-127}$
1 or 0	E = 0	M ≠ 0	Fractional form $(-1)^s \times 0.M \times 2^{-126}$
1 or 0	E = 255	M ≠ 0	NAN

Table 4.6

Double precision:



(Excess 1023)

S(1)	E(11)	M(52)	Value
0 or 1	00000000000 E = 0	0000 ... 0 M = 0	± 0
0 or 1	11111111111 E = 2047	00000... 0 M = 0	± ∞
0 or 1	1 ≤ E ≤ 2046	M = ××× ... ×	Implicit Normalized form $(-1)^s \times (1.M) \times 2^{E-1023}$
0 or 1	E = 0	M ≠ 0	Fractional form $(-1)^s \times 0.M \times 2^{-1022}$
0 or 1	E= 2047	M ≠ 0	NAN

Table 4.7:

SOLVED EXAMPLES

Q40 **IEEE 754 single precision format is used to store floating point numbers, what is the value of A-B?**
Where 'A' is the smallest number represented using implicit normalized form
'B' is the largest number represented using fractional form.
A,B are all positive numbers.

Sol:

(Fractional form) (Implicit normalized form)

$E = 0$, $M \neq 0$ $\quad$ $1 \leq E \leq 254$, $M = xxx...x$

Since A and B are positive numbers so, S = 0

S	E	M
1	8	23

32 bits

$B = (-1)^S \times (0.M) \times 2^{-126}$

$B = (-1)^0 \times (0.M) \times 2^{-126}$

$B = (0.M) \times 2^{-126}$

$0.11111....111$

GP series

$$\frac{1}{2} + \frac{1}{2^2} + \frac{1}{2^3} + ... \frac{1}{2^{23}}$$

$$= \frac{\frac{1}{2}\left(1 - \frac{1}{2^{23}}\right)}{\left(1 - \frac{1}{2}\right)} = 1 - 2^{-23}$$

$B = (1 - 2^{-23}) \times 2^{-126}$

$A = (-1)^S \times 1.M \times 2^{E-127}$

$000.....0$ $\quad E = 00000001$

$= 1.0 \times 2^{1-127}$

$= 1.0 \times 2^{-126}$

$A - B = 2^{-126} - (1 - 2^{-23})\, 2^{-126}$

$= 2^{-126} [1 - 1 + 2^{-23}]$

$= 2^{-149}$

Previous Years' Question

Given the following binary number in 32 bit (single precision) IEEE - 754 format:
00111110011011010000000000000000
The decimal value closest to this floating-point number is: **(GATE 2017–Set-2)**

a) 1.45×10^{1}
b) 1.45×10^{-1}
c) 2.27×10^{-1}
d) 2.27×10^{1}

Sol: c)

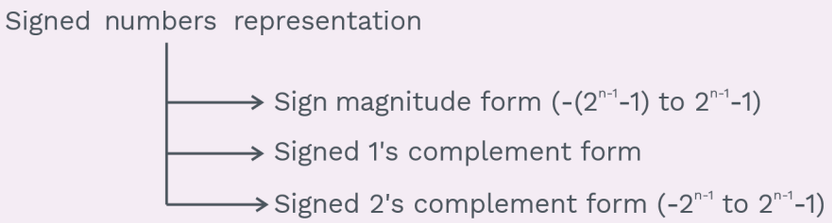
Chapter Summary

- Conversion from binary to decimal using positional weights.
- Conversion from decimal to binary by successive division method.
- Base-8 (octal) number system and Its conversions.
- Base-16 (hexadecimal) number system and Its conversions.
- Binary coded decimal or 8421 code.

Valid number ranges from 0-9.

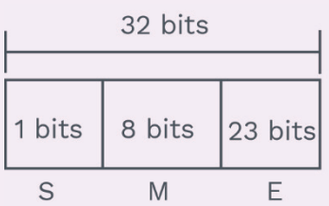
- Self-complementary code 631-1 is a self-complementary code.
- Excess-3 code - non-weighted BCD code
- Excess-3-addition → if carry occurs (add '3')
  Excess-3-addition → if carry does not occur (subtract '3')
- Gray code - Non-weighted, cyclic-unit distance code.
- Complementary number system → Diminished Radix complement $[(b^n-1)-x]$
  Complementary number system → Radix complement $[b^n-x]$

- Signed numbers representation



- Overflow: $C_{out} \neq C_{in}$ (overflow)
  $C_{out} = C_{in}$ (no overflow)
- Floating point number:

- Normalization
  - Explicit normalization
  - Implicit normalization
- IEEE Standards
  - Single precision (Excess 127)
  - Double precision (Excess 1023)
- Single-precision format:



- Double-precision format: