GeeksforGeeks

GATE फर्स्ट

CSE

DIGITAL LOGIC

SHORT NOTES

ENROLL NOW

TO EXCEL IN GATE
AND ACHIEVE YOUR DREAM IIT OR PSU!

ENROLL NOW

NUMBER SYSTEM

Base (Radix)

Total number of digit used in the system
Decimal

Base	Digit
2	0,1
3	0, 1, 2
4	0, 1, 2, 3
5	0, 1, 2, 3, 4
6	0, 1, 2, 3,4,5
7	0, 1, 2, 3, 4, 5, 6
8	0, 1, 2, 3, 4, 5, 6, 7
9	0, 1, 2, 3, 4, 5, 6, 7, 8
10	0, 1, 2, 3, 4, 5, 6, 7, 8, 9
11	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A
12	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B
13	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C
14	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D
15	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E
16	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, F

In base conversion 2 key points are there:

(A) Any base to Decimal conversion
(B) Decimal to any other base conversion

(A) Any base to Decimal conversion:

$(a_3a_2a_1a_0 \cdot a_{-1}a_{-2})_r = (\ ?)_{10}$
$(a_3 \times r^3 + a_2 \times r^2 + a_1 \times r^1 + a_0 \times r^0 + a_{-1} \times r^{-1} + a_{-2} \times r^{-2})_{10}$

Case (1) : Binary to Decimal conversion

Ex. $(1011.11)_2 = ()_{10}$
⇒ $[(1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (1 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2})]$
⇒ $[8 + 0 + 2 + 1 + 0.5 + 0.25]_{10}$
⇒ $(11.75)_{10}$

Case (2) : Octal to Decimal conversion

Ex. $(721.4)_8 = ()_{10}$
⇒ $[(7 \times 8^2) + (2 \times 8^1) + (1 \times 8^0) + (4 \times 8^{-1})]_{10}$
⇒ $[448 + 16 + 1 + 0.5]_{10}$
⇒ $(465.5)_{10}$

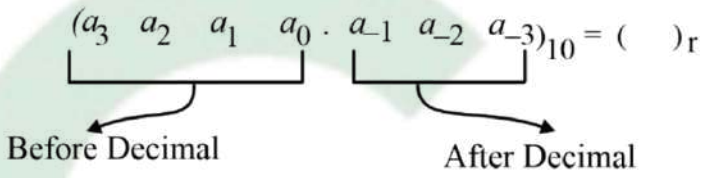
Case (3) : Hexadecimal to Decimal conversion

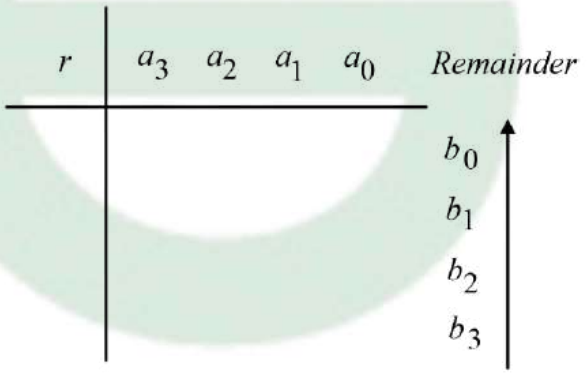
Ex. $(\ A2B.C\ )_{16} = ()_{16}$
⇒ $[(A \times 16^2) + (2 \times 16^1) + (B \times 16^0) + (C \times 16^{-1})]_{10}$
⇒ $[(10 \times 256) + (2 \times 16) + (11 \times 1) + (12 \times 16^{-1})]_{10}$
⇒ $[2560 + 32 + 11 + 0.75]_{10}$
⇒ $(2603.75)_{10}$

Case (4) : Base 5 to Decimal conversion

Ex. $(432.22)_5 = ()_{10}$
⇒ $[(4 \times 5^2) + (3 \times 5^1) + (2 \times 5^0) + (2 \times 5^{-1}) + (2 \times 5^{-2})]_{10}$
⇒ $[100 + 15 + 2 + 0.4 + 0.08]_{10}$
⇒ $(117.48)_{10}$

(B) Decimal to any other Base conversion





$0 \cdot a_{-1}a_{-2}a_{-3} \times r = x_0 \cdot x_{-1}x_{-2}$
$0 \cdot x_{-1}x_{-2} \times r = x_1 \cdot x_{-3}x_{-4}$
$0 \cdot x_{-3}x_{-4} \times r = x_2 \cdot x_{-5}x_{-6}$
$(a_3a_2a_1a_0 \cdot a_{-1}a_{-2}a_{-3})_{10} = (b_3b_2b_1b_0 \cdot x_0x_1x_2)_r$

Case (1) : Decimal to Binary Base conversion.

Ex.

$(19.75)_{10} = (\quad)_2$

Before Decimal R After Decimal

2	19	1
2	9	1
2	4	0
2	2	0
	1	1

$0.75 \times 2 = 1.5$ 1
$0.5 \times 2 = 1.0$ 1

$(19.75)_{10} = (10011.11)_2$

Case (2) : Decimal to Octal Base conversion.

Ex.

$(210.23)_{10} = (\quad)_8$

Before DecimalRemainder After Decimal

8	210	2
8	26	2
	3	3

$0.23 \times 8 = 1.84$ 1
$0.84 \times 8 = 6.72$ 6
$0.72 \times 8 = 5.76$ 5

$(210.23)_{10} = (322.165)_8$

Case (3): Decimal to Hexadecimal Base conversion.

Ex.

$(1228.55)_{10} = (\quad)_{16}$

Before Decimal Remainder After Decimal

16	1228	12 (C)
16	76	12 (C)
	4	4

$0.55 \times 16 = 8.8$ 8
$0.8 \times 16 = 12.8$ 12(C)

$(1228.56)_{10} = (4CC.8C)_{16}$

Decimal Digits	BCD 8421	Excess-3	Octal digits	BCO	Hexadecimal Digits	BCH
0	0000	0011	0	000	0	0000
1	0001	0100	1	001	1	0001
2	0010	0101	2	010	2	0010
3	0011	0110	3	011	3	0011
4	0100	0111	4	100	4	0100
5	0101	1000	5	101	5	0101
6	0110	1001	6	110	6	0110
7	0111	1010	7	111	7	0111
8	1000	1011			8	1000
9	1001	1100			9	1001
					A	1010
					B	1011
					C	1100
					D	1101
					E	1110
					F	1111

Some Special Case

Special Case:

Example: $(10110111)_2 = (\,)_8$

Octal → means base 8

$8 = 2^3$

Every three digits of binary represent one digit of octal

010 110 111

2 6 7

Hence $(10110111)_2 = (267)_8$

Case (2): Binary to Hexadecimal base conversion

Example: $(1011011)_2 = (\,)_{16}$

Hexadecimal → means base 16

$16 = 2^4$

Every four digits of binary represent one digit of Hexadecimal.

0101 1011

5 11(B)

Hence $(1011011)_2 = (5\ B)_{16}$

BCD (Binary Coded Decimal)

- In this each digit of the decimal number is represented by its four-bit binary equivalent. It is also called natural BCD or 8421 code. It is weighted code.
- Excess - 3 Code: This is an non weighted binary code used for decimal digits. Its code assignment

is obtained from the corresponding value of BCD after the addition of 3 .

- BCO (Binary Coded Octal): In this each digit of the Octal number is represented by its three-bit binary equivalent.
- BCH (Binary Coded Hexadecimal): In this each digit of the hexadecimal number is represented by its four bit binary equivalent.

Don't care values or unused states in BCD code are 1010,1011,1100,1101,1110,1111.
Don't care values or unused states in excess - 3 code are 0000,0001,0010,1101,1110,1111.
The binary equivalent of a given decimal number is not equivalent to its BCD value.

Example: $25_{10} = 11001_2$.

The BCD equivalent of decimal number $25 = 00100101$ from the above example the BCD value of a given decimal number is not equivalent to its straight binary value.
The BCO (Binary Coded Octal) value of a given Octal number is exactly equal to its straight binary value.

Example: $25_8 = 21_{10} = 010101_2$

The BCO Value of 258 is 010101 .
From the above example, the BCO value of a given Octal number is same as binary equivalent of the same number.
The BCH (Binary Coded Hexadecimal) value of a given hexadecimal number is exactly equal to its straight binary.

Example: $25_{16} = 37_{10} = 100101_2$

The BCH value of hexadecimal number $25_{16} = 00100101$.
From this example the above statement is true.

1. R-1's Complement (Radix minus 1's complement)

- Definition: For a number system with radix R, the $\mathrm{R}-1$'s complement of a number is found by subtracting each digit from $R-1$.
- This means:
- In binary $(R=2) \rightarrow R$-1's complement is 1 's complement (flip 0 to 1 , and 1 to 0).
- In decimal $(\mathrm{R}=10) \rightarrow \mathrm{R}-1$'s complement is 9 's complement (replace each digit d with $9-d$).

Example (Binary):

Number: 1010_2
Radix $R=2$, so $R-1=1$.
Replace $1 \rightarrow 0,0 \rightarrow 1$:
1 's complement $= 0101_2$

Example (Decimal):

Number: 458
Radix $R=10$, so $R-1=9$.
Replace each digit with $9-d$:
$9-4=5,9-5=4,9-8=1$
9's complement = 541

2. R's Complement (Radix complement)

- **Definition**: The R 's complement of a number is obtained by adding 1 to its $\mathrm{R}-1$'s complement.
- This means:
- In binary $(R=2) \rightarrow R$'s complement is 2's complement (flip bits and add 1).
- In decimal $(\mathbf{R}=\mathbf{10}) \rightarrow$ R's complement is $\mathbf{10}$'s complement (9's complement +1).

Example (Binary):

Number: 1010_2
1's complement = 0101 $_2$
Add 1: $0101_2 + 0001_2 = 0110_2$
2's complement $= 0110_2$

Example (Decimal):

Number: 458
9's complement = 541
Add 1: $541 + 1 = 542$
10's complement = 542

	Binary	Octal	Decimal	Hexadecimal
	$r = 2$	$r = 8$	$r = 10$	$r = 16$
Complement ($r - 1$)'s r's Complement	1 's	7's	9's	15 's
	2's	8's	10's	16 's

Example: Add the two Binary numbers 101101_2.
Augned 101101
addend 100111
1111
Sum 1010100

Example: Subtract the Binary number 100111_2 from 101101_2.
Minuend : 101101
Subtracted: 100111
Difference: 000110

Example: Multiple the Binary number 1011_2 from 101_2.

Multiplicand:	1011
Multiplier:	X101
	1011
	0000
	+
Product:	11011

While storing the signed binary numbers in the internal registers of a digital computer} most significant bit position is always reserved for sign bit and the remaining bits are used for magnitude.

When the binary number is positive, the sign is represented by ' 0 '. When the number is negative, the sign is represented by ' 1 '.
Fixed-Point Representation and Floating-Point Representation;
The representation of the decimal point (ordinary point) in a register is complicated by the fact that it is characterized by a position between two flip-flops in the register.
There are two ways of specifying the position of the decimal point in a register.
(1) Fixed Point and
(2) Floating Point.

The 2's complement of a given binary number can be formed by leaving all least significant zeros and the first non-zero digit unchanged, and then replacing 1's by 0's and 0's by 1's in all other higher significant digits.

Example:
The 2's complement of 10011000_2 is 01101000 .

Subtraction using 2's complement: Represent the negative number in signed 2's complement from, add the two numbers, including their sign bit, and discard any carry out of the most significant bit. Since negative numbers are represented in 2's compliment form, negative results also obtained in signed 2's compliment form.

Example: 1's complement:

$$\begin{array}{cccc} +60000110 & -61111001 & +60000110 & -61111001 \\ +90001001 & +90001001 & -91110110 & -91110110 \\ \hline +150001111 & +3\ (i)\ 0000010 & -31111100 & -15\ (1)\ 1101111 \\ & + \begin{array}{c}\text{Carry } +1 \\ +30000011 \\ +\text{ carry}\end{array} & & \begin{array}{c}\text{Carry } +1 \\ 1110000 \\ \text{carry}\end{array} \end{array}$$

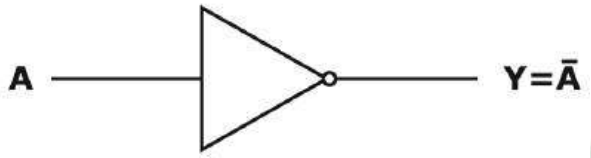
The advantage of signed 2's complement representation over the signed 1's compliment from (and the signed - magnitude form) is that it contains only one type of zero.

LOGIC OPERATIONS

In Boolean algebra, all the algebraic functions performed is logical. The AND, OR and NOT are the basic operations that are performed in Boolean algebra. There are some derived operations such as NAND, NOR, EX-OR, EX-NOR that are also performed in Boolean algebra.

NOT operation:

Symbol:



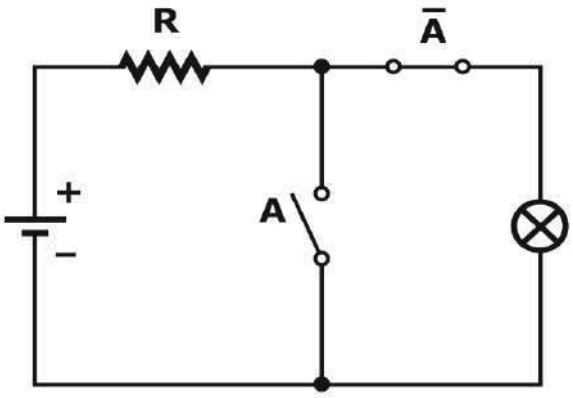
$A \xrightarrow{NOT} \bar{A}$ or A' (Complementation law)

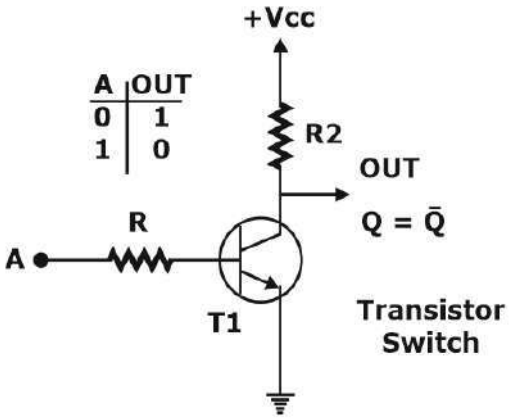
and $\bar{\bar{A}} = A \Rightarrow$ Double complementation law

Truth table for NOT operation:

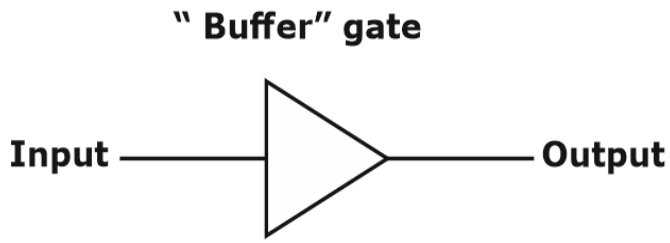
Input A	Output $Y = \bar{A}$
0	1
1	0

A NOT gate can be represented using switch whose circuit representation is shown in figure below.





A **buffer** is a basic logic gate that passes its input, unchanged, to its output. Its behaviour is the opposite of a NOT gate.

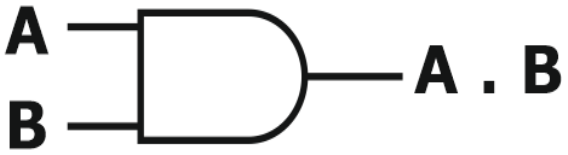


Input	Output
0	0
1	1

AND operation:

Symbol:

$A.A = A,\ A.0 = 0,\ A.1 = A,\ A\bar{A} = 0$

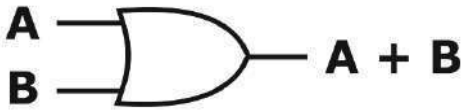


Truth table for AND operation:

Input		Output
A	B	Y = AB
0	0	0
0	1	0
1	0	0
1	1	1

OR operation:

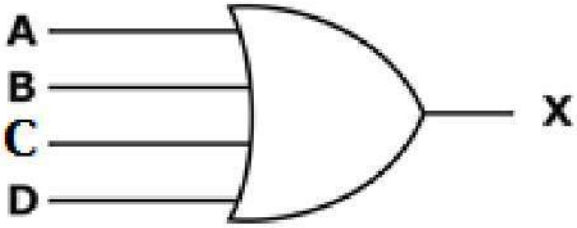
Symbol:



$A + A = A$, $A + 0 = A$, $A + 1 = 1$, $A + \bar{A} = 1$

Truth table for OR operation:

Input		Output
A	B	Y = A+B
0	0	0
0	1	1
1	0	1
1	1	1



Enable or Disable Input:

For a two input AND gate:

- One input is the signal and the other input is the enable pulse.
- The enable of an AND gate is high active. That is, when the enable is high the input signal will appear on the output.
- When the AND gate enable input is low, the output will remain a constant low signal.

For a two input OR gate:

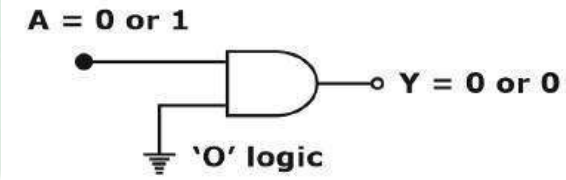
- A two OR gate can also be used with one input the desired signal and the other input is the enable.
- The enable input an OR gate is low active. This means that the output will be copy of the input signal when the enable is low.
- When the enable input of an OR gate is high, the output of the gate will be constant high signal.

Example:

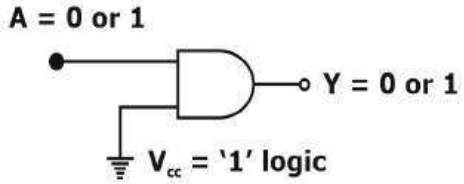
Show the control enable and disable outputs for AND and OR gate.

Solution:

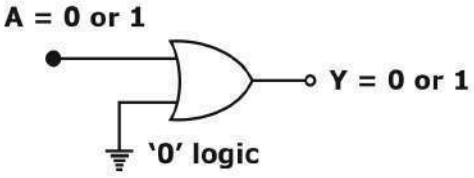
Control '0' disable



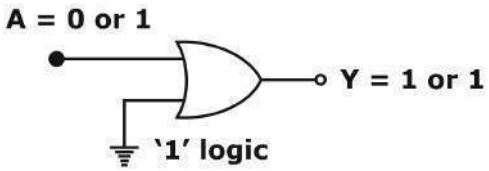
Control '1' enable (Buffer)



Control '0' enable (Buffer)



Control '1' Always enable



Basic law applications in AND/OR gate.

a. Commutative Law:

The commutative law allows change in position of AND or OR variables. There are two commutative laws.

$A + B = B + A$

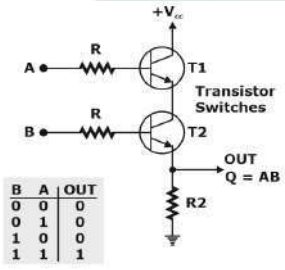
$A \cdot B = B \cdot A$

b. Associative Law:

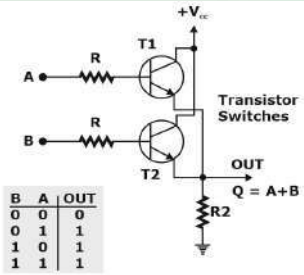
$(A + B) + C = A + (B + C)$

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

Circuit Diagram for AND/OR gate.

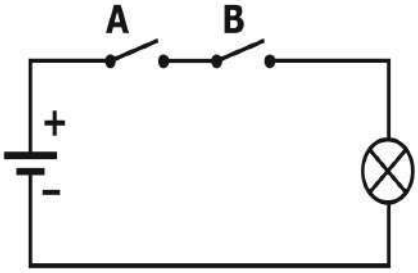


B	A	OUT
0	0	0
0	1	0
1	0	0
1	1	1

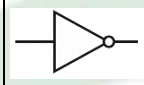
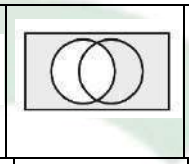
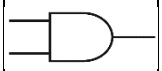
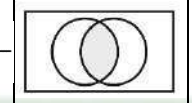
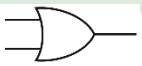
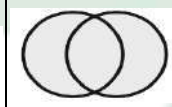


B	A	OUT
0	0	0
0	1	1
1	0	1
1	1	1

The circuit shown below shows the switch representation of AND gate which is basically the series connection of switches A and B.



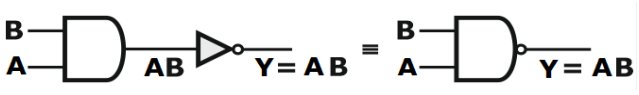
Venn Diagram:

NOT			$\bar{A}$	A		Output
				0		1
				1		0
AND			A.B	A	B	Output
				0	0	0
				0	1	0
				1	0	0
				1	1	1
OR			A + B	A	B	Output
				0	0	0
				0	1	1
				1	0	1
				1	1	1

NAND gate:

The term NAND implies NOT-AND

Symbol:

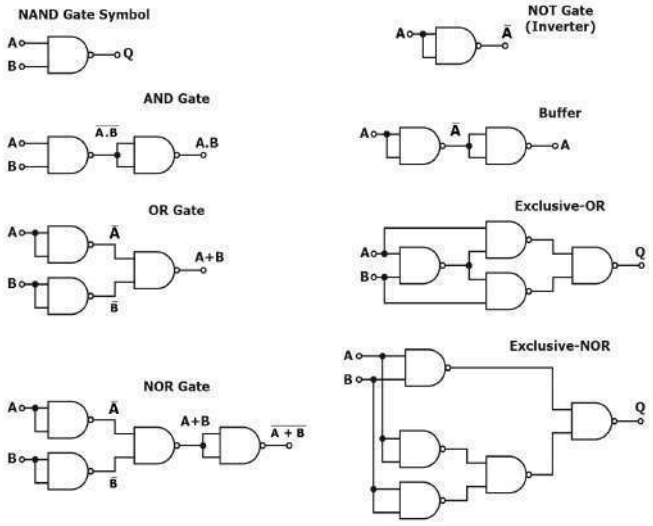


Truth table of 2-input NAND gate.

Input		Output
A	B	$Y = \overline{AB}$
0	0	1
0	1	1
1	0	1
1	1	0

NAND gate acts as Universal Gate

Logic Gates using only NAND Gates



All the logic gate functions can be created using only NAND gates. Therefore, it is also known as a Universal logic gate.

NOR gate:

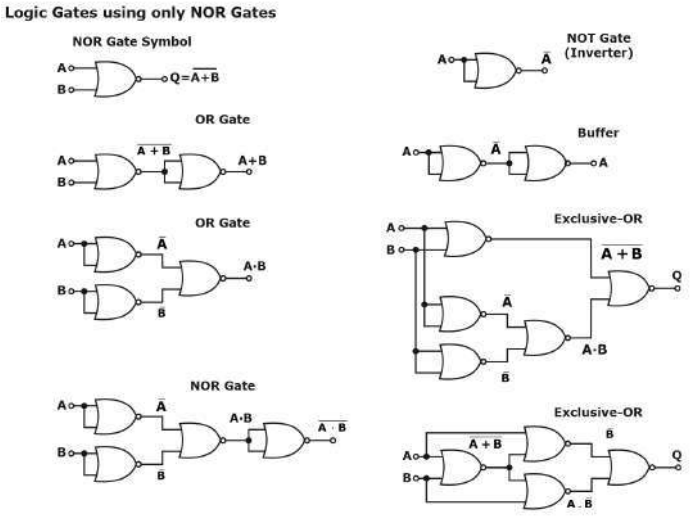
A NOR gate is equivalent to OR gate followed by a NOT gate.

Symbol:

Truth Table for 2-input NOR gate

Input		Output
A	B	$Y = \overline{A + B}$
0	0	1
0	1	0
1	0	0
1	1	0

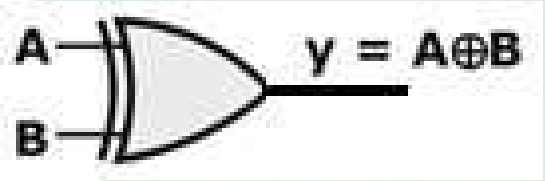
NOR gate acts as Universal Gate.



All the logic gate functions can be created using only NOR gates. Therefore, it is also known as a Universal logic gate.

XOR gate:

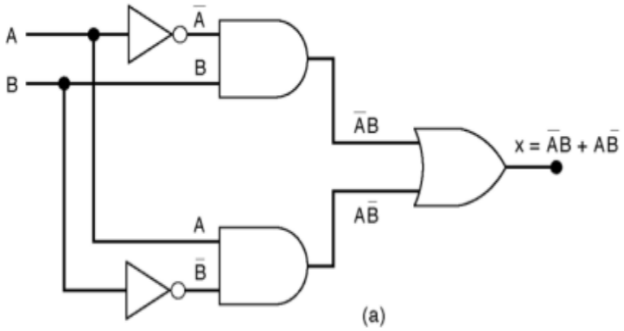
Symbol of two input XOR gate



Truth table for 2-input XOR gate

Input		Output
A	B	Y = A ⊕ B
0	0	0
0	1	1
1	0	1
1	1	0

XOR gate using AND OR and NOT gate



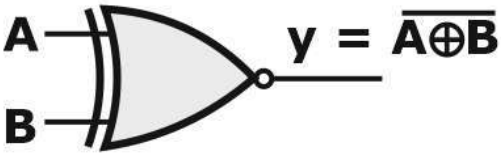
Truth Table:

A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

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

X-NOR gate:

Symbol for two input X-NOR gate



Truth table for 2-input X-NOR gate

Input		Output
A	B	Y = A ⊙ B
0	0	1
0	1	0
1	0	0
1	1	1

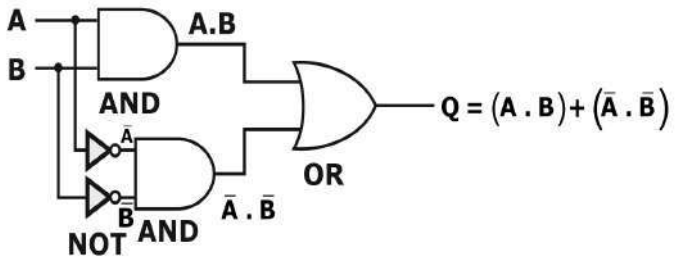
Boolean expression for EX-NOR gate is $Y = \overline{A \oplus B}$

Apply De-Morgan's theorem:

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

The output of a two input EX-NOR gate is logic '1' when the inputs are same and a logic '0' when they are different.

X-NOR gate using AND OR and NOT gate

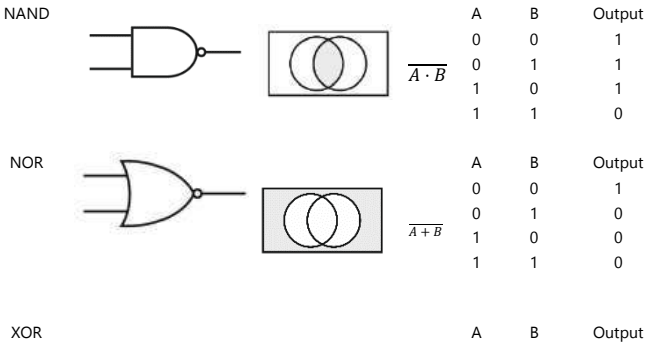


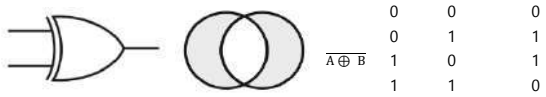
Truth Table:

A	B	Y
0	0	1
0	1	0
1	0	0
1	1	1

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

Venn Diagram:

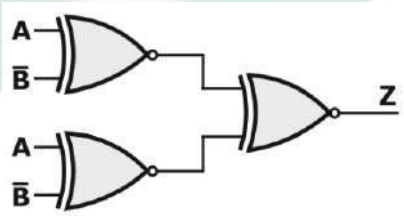




ALTERNATE LOGIC GATE REPRESENTATION

Logic	Normal symbol	Alternate symbol
NOT	A — $\bar{A}$	A — $\bar{A}$
AND	A, B → AB	A, B → $\overline{\bar{A}+\bar{B}}=AB$
OR	A, B → A+B	A, B → $\overline{\bar{A}\bar{B}}=A+B$
NAND	A, B → $\overline{A.B}$	A, B → $\bar{A}+\bar{B}=\overline{A.B}$
NOR	A, B → $\overline{A+B}$	A, B → $\bar{A}.\bar{B}=\overline{A+B}$

Example: In the following circuit, the find the output Z?



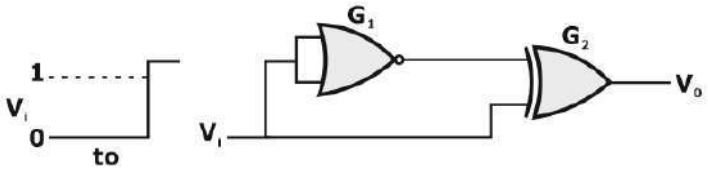
Solution:

From the given circuit, we can observe that input to last XNOR is same, so, the XNOR output is given by (let input is X)

$$Z = X.X + \overline{X}.\overline{X} = X + \overline{X} = 1$$

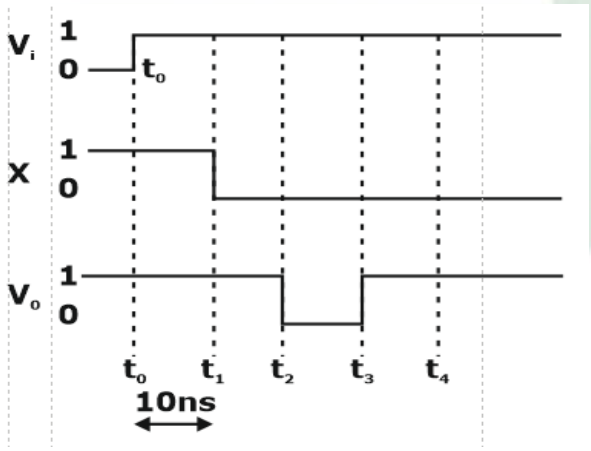
i.e. the output will be high [logic 1] irrespective of the inputs A and B.

Example: The gate G_1 and G_2 in figure shown below have propagation delays of 10ns and 20ns respectively.



If input V_i makes an abrupt change from logic 0 to 1 at $t = t_0$, then find the output waveform V_0?
Here, $t_1 = t_0 + 10$ ns, $t_2 = t_1 + 10$ns, $t_3 = t_2 + 10$ ns.

Solution:
Let the output of $G_1 = X$
The output waveform will be as shown in figure below.



BOOLEAN ALGEBRA

Boolean algebra is a system of mathematical logic. It is an algebraic system consisting on the set of elements (0, 1) two binary operators called OR, AND and one unary operator NOT. It is the basic mathematical tools in the analysis and the synthesis of switching circuits. It is a way to express logic functions algebraically.

Note:- Any functional relation in Boolean algebra can be provided by the method of perfect induction perfect inductions the method of proof, where by a function relation is verified for every possible combination of values that the value may assume.

Axioms of Boolean Algebra:-

	AND operator	OR operator	NOT operators
Axioms 1:	0.0 = 0	0 + 0 = 0	$\overline{1} = 0$
Axioms 2:	0.1 = 0	0 + 1 = 1	$\overline{0} = 1$
Axioms 3:	1.0 = 0	1 + 0 = 1	
Axioms 4:	1.1 = 1	1 + 1 = 1	

1. NOT Operation:-

The NOT operation in Boolean algebra is similar to inversion in ordinary algebra

$1 : \overline{0} = 1$

$2 : \overline{1} = 0$

3 : if A = 0 then $\overline{A} = 1$

4 : $\overline{\overline{A}} = A$ (Double inversion)

2. AND operation:- It is a logical operation that are performed by and gate. The AND operation in Boolean Algebra is similar to multiplication in ordinary algebra.

1: $A \cdot 0 = 0$ (Null Law)

2: $A \cdot 1 = A$ (Identity law)

3: $A \cdot A = A$

4: $A \cdot \overline{A} = 0$

3. OR Operation:- It is the logical operation that are performed by OR gate. The OR operation in Boolean Algebra is similar to addition in ordinary algebra.

1: A + 0 = A (Null law)

2: A + 1 = 1 (Identity law)

3: A + A = A

4: $A + \overline{A} = 1$

4. NAND Operation:- The NAND operation in Boolean Algebra is performed by AND operation followed by NOT operation i.e., the negation of AND operation is performed by NAND gate.

5. NOR Operation:- The NOR operation in Boolean Algebra is performed by OR operation followed by NOT operation i.e., the negation of OR operation is performed by NOR gate.

Laws of Boolean Algebra

(1) Commutative law:-

1: A + B = B + A

A + B + C = B + C + A = C + A + B = B + A + C

2: AB = BA

$A \cdot BC = B \cdot CA = C \cdot AB = B \cdot AC$

Violation:- Inhibition (1) for Example x/y (x but not y) is not commutative law it means

$x/y \neq y/x$

(2) Associative law:- This law arrows grouping of variables

1: (A + B) + C = A + (B + C)

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

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

2: $(A \cdot B)C = A \cdot (B \cdot C)$

$A\,(BCD) = (ABC) \cdot D$

$A\,(BCD) = AB \cdot CD$

Variation:- NAND and NOR gates are not Associative

(3) Distributive Law:-

1: A(B + C) = AB + AC

A + BC = (A + B) (A + C)

(4) Redundant literal rule:-

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

$A\left(\overline{A} + B\right) = AB$

(5) Idempotent law:-

1: $A \cdot A = A$

2: $A + A = A$

(6) Absorption law:-

1: $A + AB = A$

2: $A(A + B) = A$

(7) Involutionary law:- The law that for any variable A.

$\overline{\overline{A}} = (A')' = A$

(8)Consensus theorem:

There are two consensus theorems

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

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

De-Morgan's theorem:

De-Morgan's theorem represents two of the most important rules of Boolean algebra.

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

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

The above two laws can be extended for 'n' variables as,

$\overline{A_1.A_2.A_3 \ldots + A_n} = \overline{A}_1 + \overline{A}_2 + \ldots \overline{A}_n$

and $\overline{A_1 + A_2 + \ldots + A_n} = \overline{A}_1.\overline{A}_2 \ldots \overline{A}_n$

1.9. Duality theorem:

Duality Theorem states that,

a) Change each OR sign by an AND sign and vice versa.

b) Compliment any '0' or '1' appearing in expression

c) keep literals as it is.

NOTE: With N variables, maximum possible distinct logic function or self-dual possible = 2^{2^n}

Basic Rules of Boolean Algebra

1. A + 0 = A	**7. A · A = A**
2. A + 1 = 1	**8.** $A \cdot \overline{A} = 0$
3. A · 0 = 0	**9.** $\overline{\overline{A}} = A$
4. A · 1 = A	**10. A + AB =**
5. A + A = A	**11.** $A + \overline{A}B = A + B$
6. $A + \overline{A} = 1$	**12. (A + B) (A + C) = A + BC**

Q. Apply Demorgans theorem to expression $F = \overline{AB}(CD + EF)\left(\overline{AB} + \overline{CD}\right)$

Sol.

The given expression is

$$F = \overline{\overline{AB}(CD + EF)\left(\overline{AB} + \overline{CD}\right)}$$

$$= \overline{\overline{AB}} + \left(\overline{CD + \overline{E}\overline{F}}\right) + \left(\overline{\overline{AB} + \overline{CD}}\right)$$

$$= AB + \overline{CD} \cdot \overline{\overline{E}\overline{F}} + \overline{\overline{AB}} \cdot \overline{\overline{CD}}$$

$$AB + \left(\overline{C} + \overline{D}\right)\left(E + \overline{F}\right) + ABCD$$

Q. Reduce the Expression $F = \overline{AB} + \overline{A} + AB$

Sol.

The given expression is

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

$$F = \overline{\overline{AB}} \cdot \overline{\overline{A}} \cdot \overline{AB}$$

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

$$F = 0$$

Q. If a function is given $f = AB + \overline{A}\,\overline{B}$ find its dual.

Sol.

Given function $f = AB + \overline{A}\,\overline{B}$

For dual function

$x \leftrightarrow +$

$$f = (A + B) \cdot \left(\overline{A} + \overline{B}\right)$$

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

Dural of $f = A\overline{B} + \overline{A}B$

Q. If a function is given as $f = AB + \overline{A}\,\overline{B}$ then find its complement.

Sol.

Given $f = \left(AB + \overline{A}\,\overline{B}\right)$

Complement of $\bar{f} = \overline{A\bar{B} + \bar{A}\bar{B}}$

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

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

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

Q. Show that

$f = \left(\overline{A + \bar{B}C}\right) \cdot (A\bar{B} + ABC) = 0$

Sol.

$f = \left(\overline{A + \bar{B}C}\right)(A\bar{B} + ABC)$

$= \left(\bar{A} \cdot \overline{\bar{B}C}\right)(A\bar{B} + ABC)$

$= (\bar{A}BC)(A\bar{B} + ABC)$

$= \bar{A}BC \cdot A\bar{B} + \bar{A}BC \cdot ABC$

$= 0 + 0$

$= 0$

Q. Show that

$AB + B\bar{C} + AC = AC + B\bar{C}$

Sol.

LHS $= AB + B\bar{C} + AC$

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

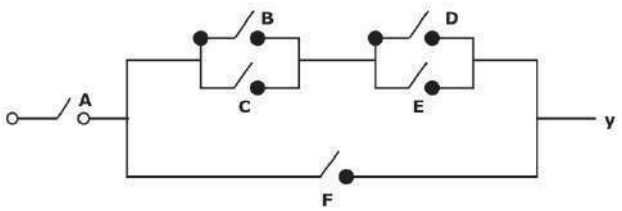
$= ABC + AB\bar{C} + AB\bar{C} + \bar{A}B\bar{C} + ABC + A\bar{B}C$

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

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

$=$ RHS

Q. What is the Boolean function does the following circuit represent?



A. A [F + (B + C) . (D + E)]

B. A + B C + DE + F

C. A (B + C) + A (D + E) + F

D. None of these

Sol.

Switch B & C are in parallel, so it is equivalent to OR operation i.e. B + C

Switch D & E are in parallel, so it is equivalent to OR operation i.e., D + E

Switch in series will become AND operation so

F = A[F + (B + C) . (D + E)]

So, option A will be correct.

Q. The complement of the function

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

A. $\bar{A}B + C\bar{D} + B\bar{C}$

B. $\bar{A}B + \bar{C}D + \bar{B}C$

C. $A\bar{B} + C\bar{D} + BC$

D. AB + BC + CD

Sol. Given function $F = (A + \bar{B})(\bar{C} + D)(\bar{B} + C)$ using Demorgan's law

Complement of F $= \bar{F} = \overline{(A + \bar{B})(\bar{C} + D)(\bar{B} + C)}$

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

$\bar{F} = \bar{A}B + C\bar{D} + B\bar{C}$

So option 'A' will be correct.

Q. If is given that $\overline{AB} + \bar{A}B = C$ then find the value of $\overline{AC} + \bar{A}C$

A. $\bar{A} + B$

B. $A + \bar{B}$

C. $\bar{A} + \bar{B}$

D. A + B

Sol. $\overline{AC} + \bar{A}C = A\overline{(\overline{AB} + \bar{A}\bar{B})} + \bar{A}\left(\overline{AB} + \bar{A}B\right)$

$= A\overline{(\bar{A} + \bar{B} + \bar{A}B)} = \bar{A}\left(\bar{A} + \bar{B} + \bar{A}B\right)$

$= \overline{AB} + \bar{A}\left[\bar{A}[1 + B] + \bar{B}\right]$

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

$= \bar{A} + \bar{A}\bar{B} + B = \bar{A}(1 + \bar{B}) + B = \bar{A} + B$

So option (A) will be correct.

Q. What is the value of $A + \bar{A}B$

A. A B. B

C. O D. A + B

Sol. Given $(A + \bar{A}B) = (A + \bar{A})(A + B)$

= (A + B)

So option (D) will be correct.

Example : Prove that:

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

Solution: Taking L.H.S:

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

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

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

$= AC + BC + A\bar{B}$

Now, R.H.S $= (A + B)(\bar{B} + C) = A\bar{B} + AC + BC$

= L.H.S

COMBINATIONAL CIRCUITS

The combinational circuit has 'n' input variables and 'm' output variables. Since, the number of input variables is n, there are 2^n possible combinations of bits at the input. Each output can be expressed in terms of input variables by a Boolean expression.

Block diagram of a Combinational Circuit

Figure : Block diagram of a Combinational Circuit

ADDERS

The most basic arithmetic operation is the addition of two binary digits. A combinational circuit that performs the addition of two 1-bit numbers is called as half adder, and the logic circuit that adds three 1-bit numbers is called as full adder.

Half Adder:

The logic circuit that performs the addition of two 1-bit numbers is called as half adder. It is the basic building block for addition of two single bit numbers. This circuit has two outputs namely carry (C) and sum (S).

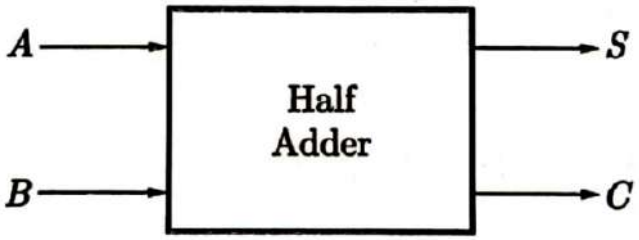


Figure : Block Diagram of a 2-bit Half Adder

The truth table of half adder, where A and B are the inputs and sum and carry are the outputs.

Inputs		Outputs	
A	**B**	**Sum (S)**	**Carry (C)**
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

Table : Truth Table of Half Adder

K–map simplification for Carry and Sum:

Boolean expressions for the sum (S) and carry (C) outputs from K – maps:

K-map for sum output **K-map for carry output**

Sum, $S = \bar{A}B + A\bar{B} = (A \oplus B)$

Carry, $C = AB$

Logic Diagram:

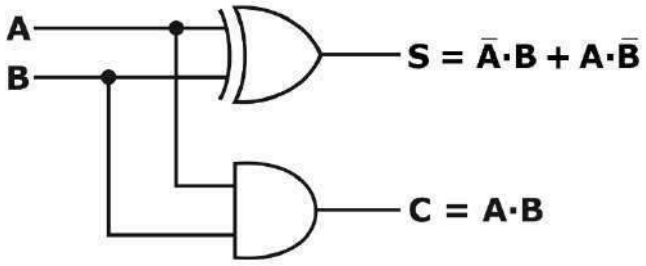


Figure : Logic Diagram of Half Adder

Full Adder:

A full adder circuit is an arithmetic circuit block that can be used to add three bits to produce a sum and a carry output. Let us consider A and B as two 1-bit inputs & C_{in} is a carry generated from the previous order bit additions. Let S (sum) and C_{out} (carry) are the outputs of the full adder.

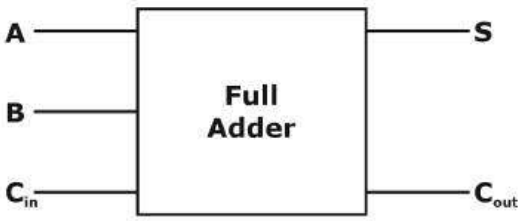


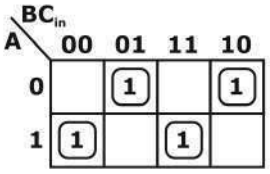
Figure : Block Diagram of a Full Adder

The Truth Table for Full Adder is given as:

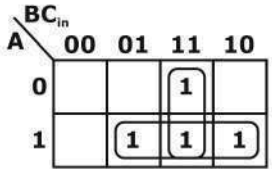
Inputs			Outputs	
A	B	C_{in}	Sum (S)	Carry C_{out}
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	1	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Table : Truth Table for Full Adder

K – map Simplification for Carry and Sum:



K-map for S



K-map for C_{out}

Simplified Boolean expressions for the sum (S) and carry (C_{out}) output from K-maps is

$$\text{Sum,} \quad S = \bar{A}\bar{B}C_{in} + \bar{A}B\bar{C}_{in} + AB C_{in} + A\bar{B}\bar{C}_{in} = C_{in}\left(\bar{A}\bar{B} + AB\right) + \bar{C}_{in}\left(\bar{A}B + A\bar{B}\right)$$

$$= C_{in}\left(A \square B\right) + C_{in}\left(A \oplus B\right)$$

$$= C_{in}\left(\overline{A \oplus B}\right) + C_{in}\left(A \oplus B\right)$$

$$\text{Sum,} \quad S = C_{in} \oplus A \oplus B$$

$$\text{Carry,} \quad C_{out} = AB + AC_{in} + BC_{in}$$

Logic Diagram:

We can realize logic diagram of a full adder using gates as shown in below figure:

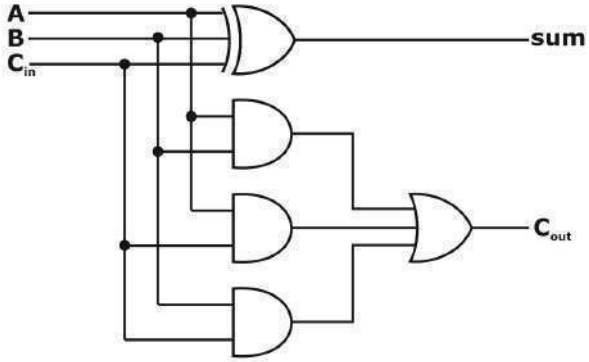
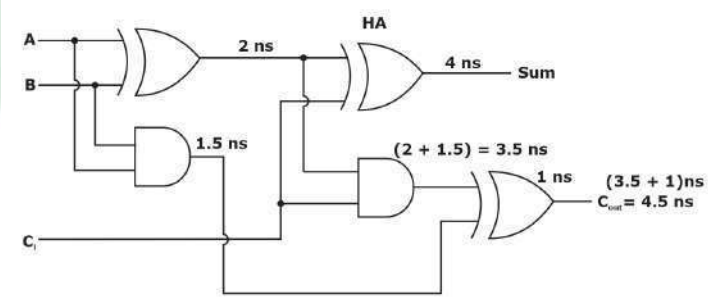


Figure : Logic Diagram of Full Adder

Example 1:

A full adder is implemented using two input OR gate and two half adders. Half adder is implemented using two input XOR and two input AND gate. The propagation delays of XOR gate, AND gate and OR gate respectively are 2ns, 1.5ns. and 1ns. The propagation delay of full adder is ..... ns.

Solution:



Figure

Example 2:

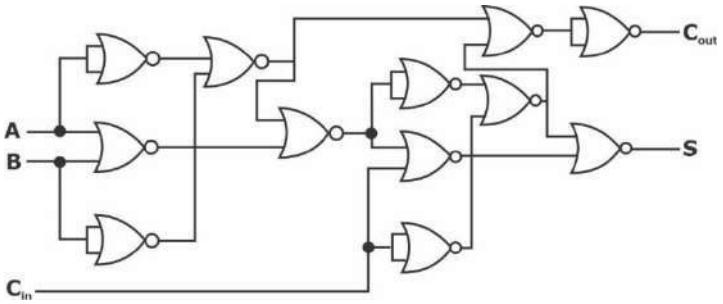
A full adder is realized using only 2 input NOR gates. The minimum number of NOR gates required to realize sum and carry out is —?

Solution: The Sum (S) and carry out (cut) for a full adder is given by

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

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

The realization circuit for the above expression is:



Figure

Here, the minimum no. of NOR gates required to realize sum and carry out is 12.

SUBTRACTORS

Half subtractor:

A half subtractor is a combinational logic circuit, which performs the subtraction of two 1-bit numbers. It subtracts one binary digit from another to produce a DIFFERENCE output and a BORROW output.

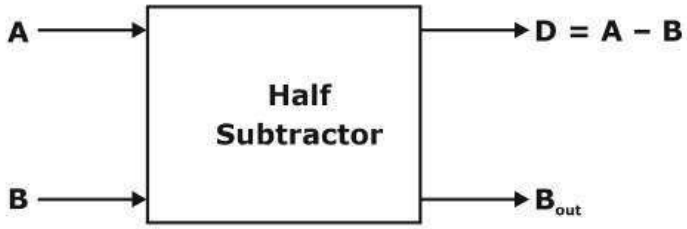


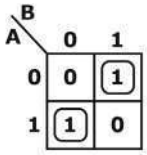
Figure : Block diagram of a half subtractor

The truth table of half – subtractor, where A, B are the inputs, and difference (D) and borrow (B) are the outputs.

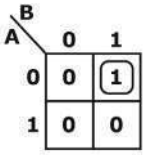
Inputs		Outputs	
A	**B**	**D**	**B_{out}**
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0

Table : Truth table of Half – Subtractor,

K – map Simplification for Difference and Borrow:



K-map for difference output



K-map for borrow output

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

Borrow, $B_{out} = \bar{A}B$

Logic Diagram:

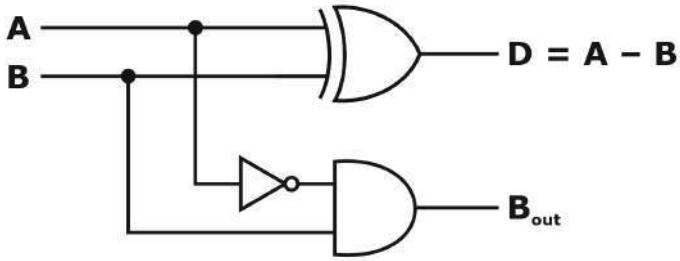


Figure : Logic Diagram of a Half subtractor

Full Subtractor:

A full subtractor performs subtraction operation on two bits, a minuend and a subtrahend.

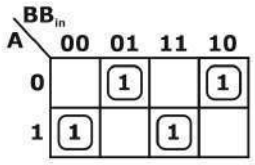
Figure : Block Diagram of a Full subtractor

Truth table of Full subtractor:

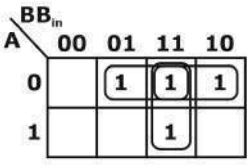
Inputs			Outputs	
A	**B**	**B_{in}**	**D**	**B_{out}**
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1

Table : Truth table of Full subtractor:

K – map simplification for Difference and Borrow:



K-map for difference output



K-map for borrow output

Difference, $D = \bar{A}\bar{B}B_{in} + \bar{A}B\bar{B}_{in} + A\bar{B}\bar{B}_{in} + ABB_{in}$

$= B_{in}\left(AB + \bar{A}\bar{B}\right) + \bar{B}_{in}\left(A\bar{B} + \bar{A}B\right)$

$= B_{in}\left(\overline{A \oplus B}\right) + B_{in}\left(A \oplus B\right)$

$D = A \oplus B \oplus B_{in}$

Borrow, $B_{in} = \bar{A}B + \bar{A}B_{in} + BB_{in}$

Logic Diagram

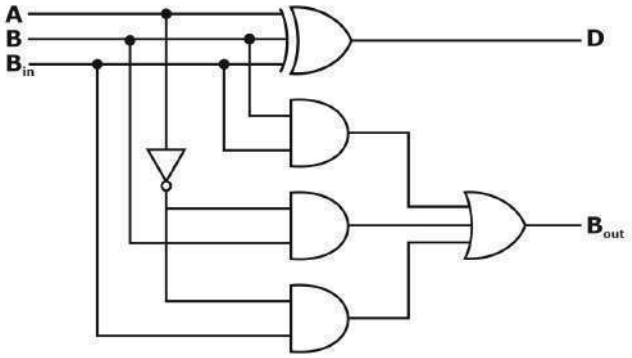


Figure : Logic Diagram of a Full Subtractor

BINARY PARALLEL ADDER

An n-bit parallel adder can be constructed using n number of full adders are connected in parallel and hence; it is also known as parallel adder such that the previous carry or carry input for adder 0 is set to zero. The carry output of each adder is connected to the carry input of the next higher order adder. Hence, it is also known as carry propagate adder.

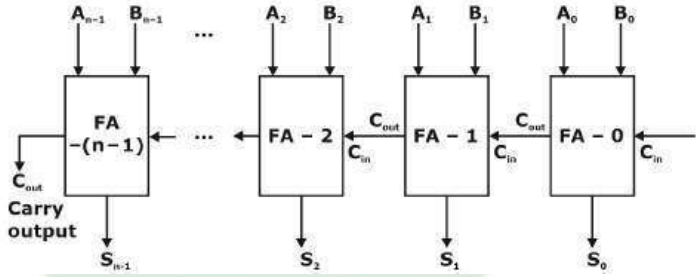


Figure: n-bit Binary Adder

Worst case delay

T = (n-1) T_{carry} + $Max\ \{T_{sum}, T_{carry}\}$

n = number of Adder (no. of bits)

$T_{carry} = Time\ required\ to\ generate\ carry$

$T_{sum} = Time\ required\ to\ generate\ carry$

Addition per second = Addition Rate = $\frac{1}{T}$

Propagation Delay in Parallel Adder:

Parallel adders suffer from propagation delay problem because higher bit additions depend on the carry generated from lower bit addition. In effect, carry bits must propagate or ripple through all stages before the most significant sum bit is valid. Thus, the total

sum (the parallel output) is not valid until after the cumulative delay of all the adder.

CARRY LOOK AHEAD ADDER

The look ahead carry adder speeds up the operation by eliminating this ripple carry delay. It examines all the input bits simultaneously and also generates the carry in bits for all the stages simultaneously. The method of speeding up the addition process is based on the two additional functions of the full adder called the carry generate and carry propagate functions.

Carry Generation:

Carry is generated only if both the input bits are 1, that is, if both the bits A and B are 1's, a carry has to be generated in this stage regardless of whether the input carry C_{in} is a 0 or a 1. Let G as the carry generation function,

$G = A \cdot B$

Consider the present bit as the n^{th}, then

$G_n = A_n \cdot B_n$

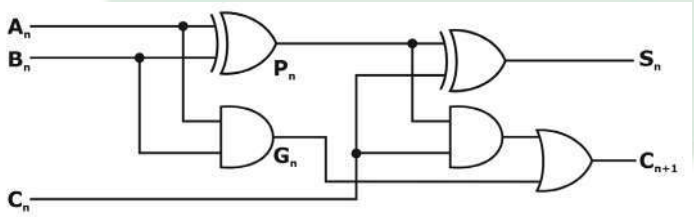


Figure : Carry Look – ahead Generator Circuit

Carry Propagation:

A carry is propagated if any one of the two input bits A or B is 1. If both A and B are 0, a carry will never be propagated. On the other hand, if both A and B are 1, then will not propagate the carry but will generate the carry. Let P as the carry – propagation function, then

$P_n = A_n \oplus B_n$

Look ahead Expressions:

Let n^{th} bit adder, the sum (S) and the carry out (C) for the n^{th} bit may be expressed in terms of the carry generation function (G) and the carry propagation function (P) as

$S_n = P_n \oplus C_n$

$C_{n+1} = G_n + P_n \cdot C_n$

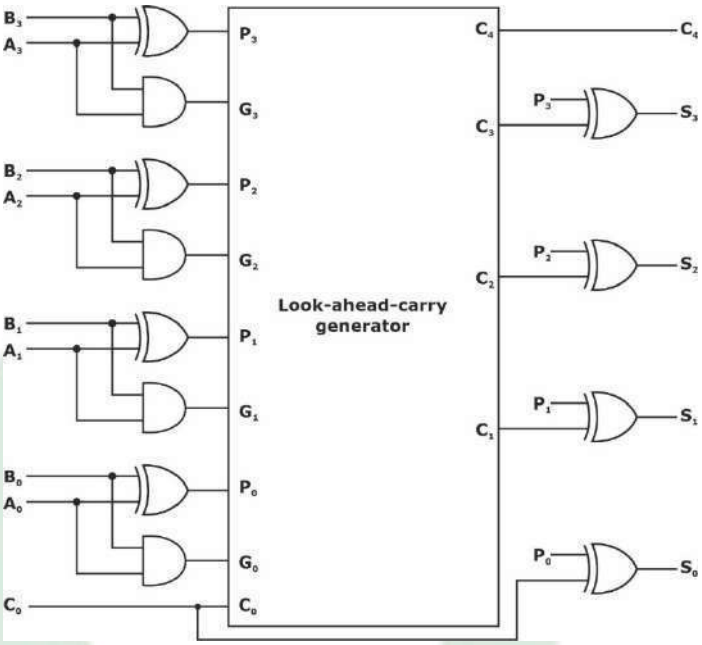


Figure: 4-bit Full Adder with a look Ahead Carry Generator

Example 3:

What is the main drawback of carry lookahead adder?

Solution: Its hardware is more complex than the hardware for a ripple carry adder.

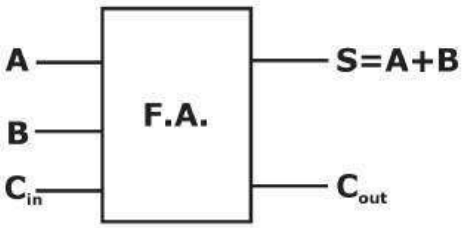
Example 4:

A 1-bit full adder takes 20ns to generate carry-out bit and 40ns for the sum bit.

The maximum rate of addition per second, when four 1-bit full adders are cascaded, is

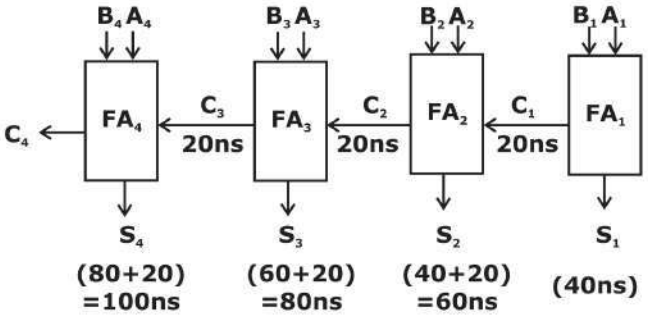
—— — × 10^6?

Solution: Given that, 1-bit full adder takes 20ns to generate carry out and 40ns for the sum bit.



Figure

Now, four 1-bit full adders are cascaded for addition of 4-bits numbers as shown below:



Figure

From the circuit, we may conclude that complete addition of 4-bits number is generated after 100ns of applying inputs. Therefore, minimum time required for addition is:

T = 100ns

Hence, Maximum rate of addition is:

$$\text{Maximum rate} = \frac{1}{100\text{ns}} = \frac{1}{100 \times 10^{-9}}$$
$$= 10^7 \text{ / sec}$$
$$= 10 \times 10^6 \text{ / sec}$$

Example 5:

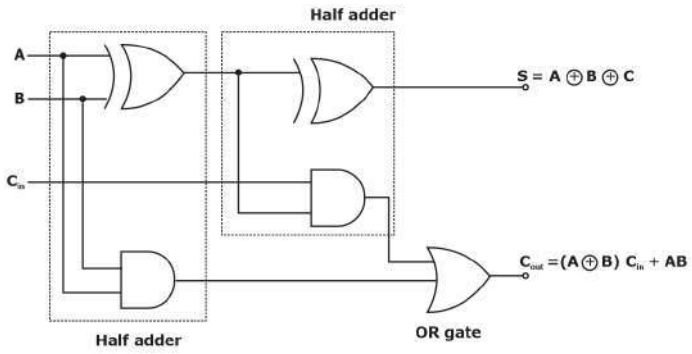
A carry look-ahead adder is frequently used for addition, because ————?

Solution: The look-ahead carry adder speeds up the process by eliminating the ripple carry delay.

Example 6:

A full adder can be realized using half adder. Explain it is detail?

Solution: A full adder realization using half adder is given by:



Figure

COMPARATOR

The comparator is a combinational logic circuit. It compares the magnitude of two n-bit numbers and provides the relative result as the output. Let A and B are the two n-bit inputs. The comparator has three outputs namely A > B, A = B and A < B. Depending upon the result of comparison, one of these outputs will go high.

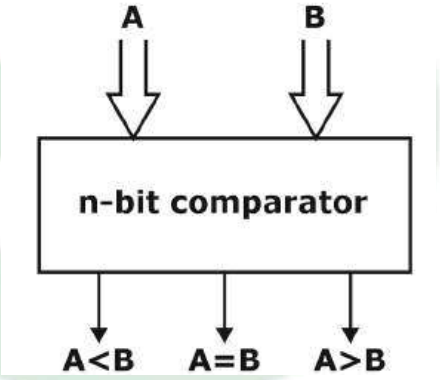


Figure Block diagram of digital comparator

1-bit Magnitude Comparator:

The 1-bit comparator is a combinational logic circuit with two inputs A and B and three outputs namely A < B, A= B and A < B.

Inputs		Outputs		
A	B	X (A < B)	Y (A = B)	Z (A > B)
0	0	0	1	0
0	1	1	0	0
1	0	0	0	1
1	1	0	1	0

Table : Truth Table of a 1-bit Comparator:

Design of 1-bit Magnitude Comparator

We can write the expressions for the three outputs as under:

$$\text{For } (A < B), \quad X = \overline{A}_0 B_0$$

$$\text{For } (A = B), \quad Y = \overline{A}_0 \overline{B}_0 + A_0 B_0 = \overline{A_0 \oplus B_0}$$

$$\text{For } (A > B), \quad Z = A_0 \overline{B}_0$$

Logic Diagram of 1-bit Comparator:

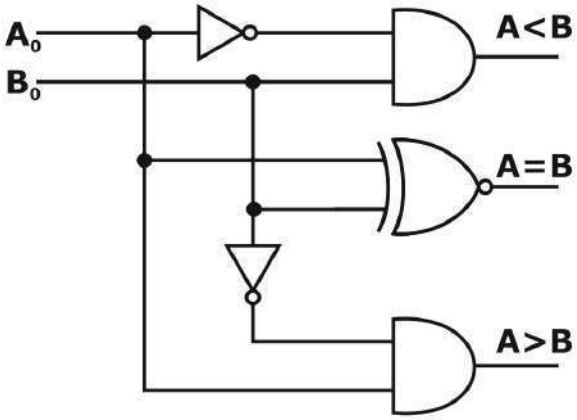


Figure Logic Diagram of 1-bit Comparator

2-bit Magnitude Comparator:

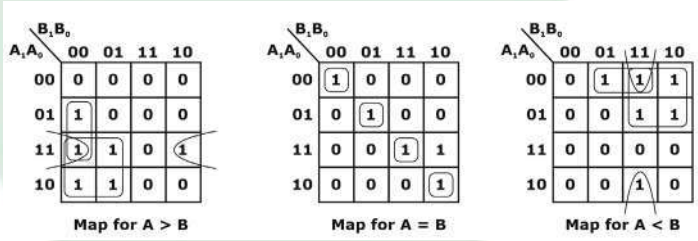
Let the two 2-bit numbers be A = A_1A_0 and B = B_1B_0.

Inputs				Outputs		
A_1	A_0	B_1	B_0	X(A<B)	Y(A=B)	Z(A > B)
0	0	0	0	0	1	0
0	0	0	1	1	0	0
0	0	1	0	1	0	0
0	0	1	1	1	0	0
0	1	0	0	0	1	0
0	1	0	1	0	0	1
0	1	1	0	1	0	0
0	1	1	1	1	0	0

Table : Truth Table for a 2-bit Comparator:

Design of 2-bit Magnitude Comparator:

The K-maps for the three outputs are shown as below:



The simplified expressions can be obtained from the K-map as

For A < B, $X = \overline{A}_1 \overline{A}_0 B_0 + \overline{A}_1 B_1 + \overline{A}_0 B_1 B_0$

For A = B, $Y = (A_0 \odot B_0)\,(A_1 \odot B_1)$

For A > B, $Z = A_0 \overline{B}_1 \overline{B}_0 + A_1 A_0 \overline{B}_0 + A_1 \overline{B}_1$

Logic Diagram of 2-bit Comparator:

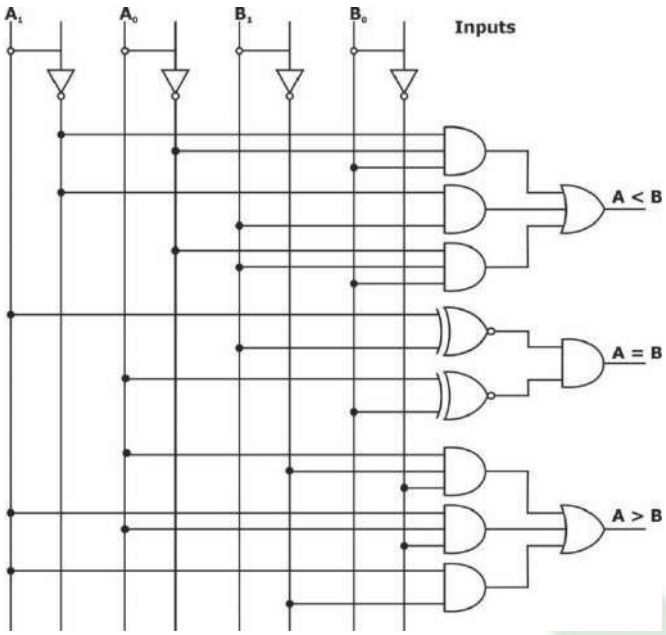
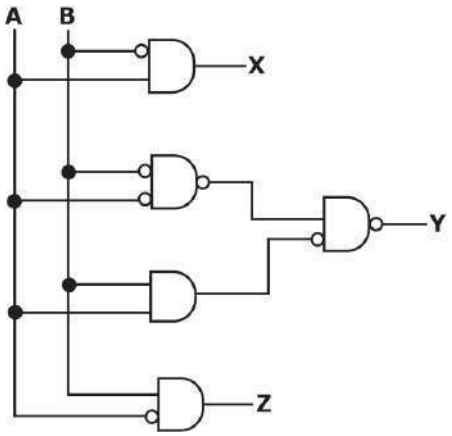


Figure Logic diagram of 2-bit Comparator

Example 7:
The circuit shown in given figure is:



Figure

Solution: From the logic circuit shown in the figure, we obtain the following results,

$X = A\bar{B}$

$Y = \overline{(\bar{A}\bar{B})(\overline{AB})} = \bar{A}\bar{B} + AB = A \odot B$

$Z = \bar{A}B$

So, we obtain the truth table for the above function as shown below.

A	B	X	Y	Z
0	0	0	1	0
0	1	0	0	1
1	0	1	0	0
1	1	0	1	0

From truth table, we deduce the following results.

It A > B, then x = 1

It A = B, the y = 1

It A < B, then z = 1

Therefore, it is a comparator circuit.

MULTIPLEXER

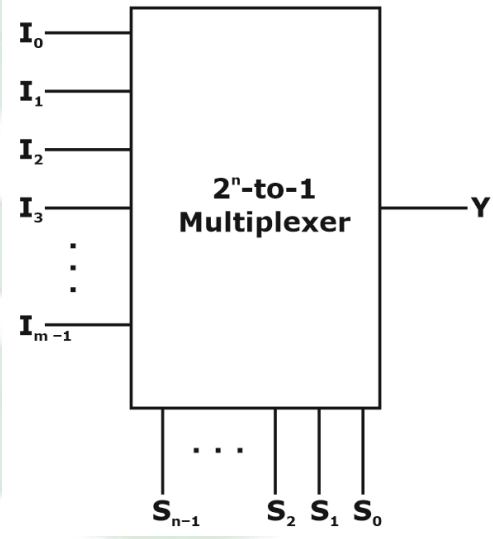


Figure : Block diagram of a 2^n-to-1 multiplexer

2 x 1 MUX:

A 2 to 1 multiplexer has 2 inputs. Since $2 = 2^1$, this multiplexer will have one control (select) line. It has two data inputs I_0 and I_1, one select input S, and one output.

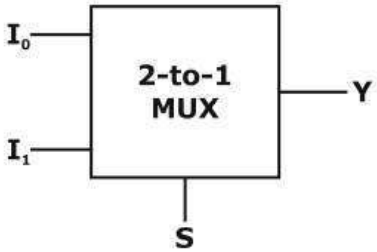


Figure: Schematic block diagram of 2:1 Multiplexer

The truth table of this MUX is given below,

Select Line (S)	Output Y
0	I_0
1	I_1

Figure

Thus, the SOP expression for the output Y is,

$Y = I_0\bar{S}_0 + I_1S_0$

Realization of a 2:1 MUX using Logic Gates:

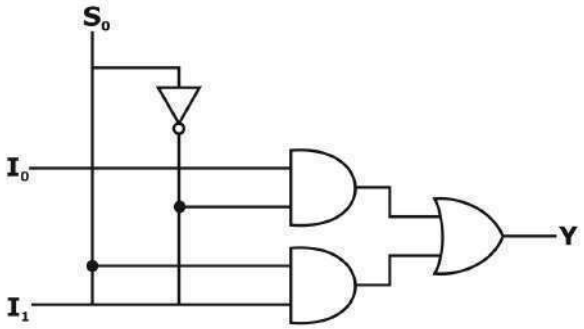


Figure : Logic Diagram of a 2 x 1 Multiplexer

4 x 1 MUX:

A 4-to-1 multiplexer has 4 inputs and two select lines, where I_0 to I_3 are the four inputs to the multiplexer, and S_0 and S_1 are the select lines.

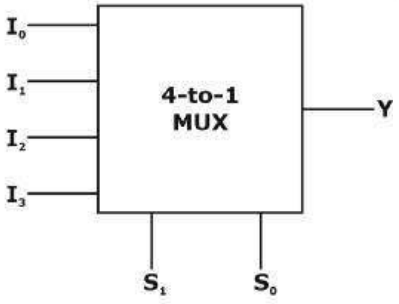


Figure: Schematic block diagram of 4 x 1 MUX

Truth Table of a 4-to-1 Multiplexer

Select Inputs S_1	Select Inputs S_0	Output Y
0	0	I_0
0	1	I_1
1	0	I_2
1	1	I_3

Output Y for a 4-input multiplexer is

$Y = I_0\bar{S}_1\bar{S}_0 + I_1\bar{S}_1S_0 + I_2S_1\bar{S}_0 + I_3S_1S_0$

Realization of a 4:1 MUX Using Logic Gates:

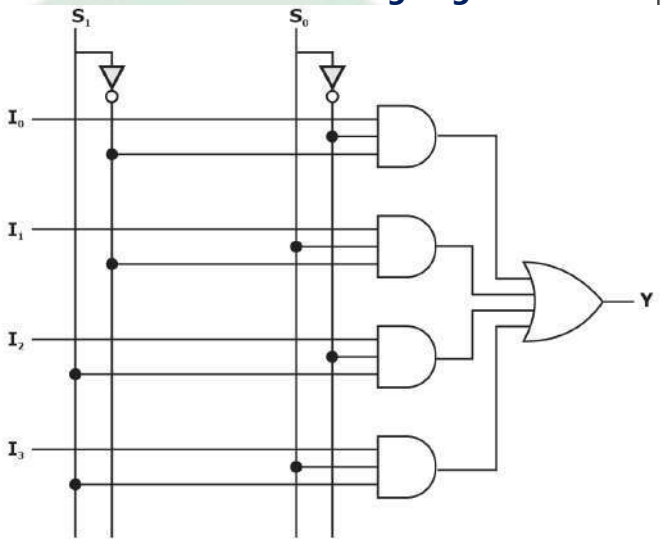


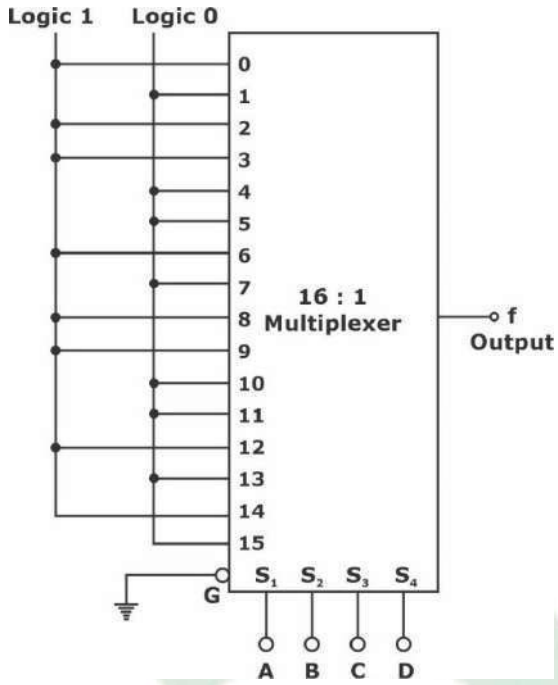
Figure : Logic Diagram of 4 x 1 MUX

Example 8:

Implement the expression using a multiplexer

$f(A,B,C,D) = \Sigma m(0,2,3,6,8,9,12,14)$

Solution: Since, there are four variables, therefore, a multiplexer with four select inputs is required. The circuit of 16 :1 multiplexer connected to implement the above expression is shown is below figure. In case the output of the multiplexer is active-low, the logic 0 and logic 1 inputs are to be interchanged.



Figure

IMPLEMENTATION OF HIGHER ORDER MUX USING LOWER ORDER MUX

The methods for implementing higher order MUX using lower order MUX are

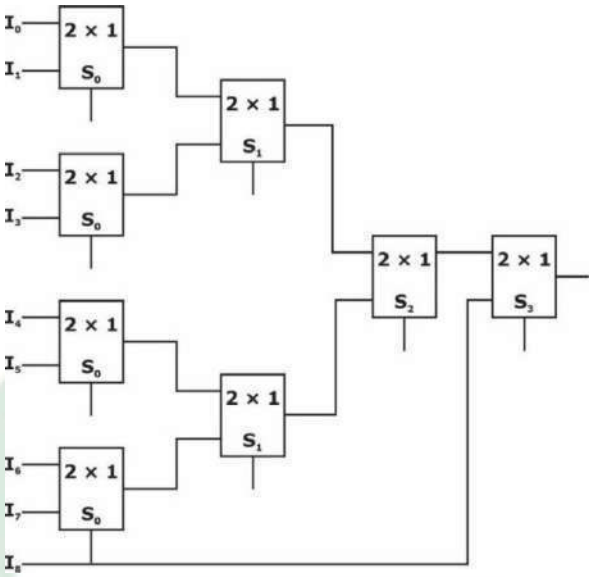
Step 1: If 2^n is the number of input lines in the available lower order multiplexer and 2^N is the number of input lines in the desired multiplexer, then the number of lower order multiplexers required to construct the desired multiplexer circuit would be 2^{N-n}.

Step 2: From the knowledge of the number of selection inputs of the available multiplexer and that of the desired multiplexer, connect the less significant bits of the selection inputs of the desired multiplexer to the selection inputs of the available multiplexer.

Step 3: The most significant bits of the selection inputs of the desired multiplexer circuit are used to enable or disable the individual multiplexers so that their outputs when OR produce the final output.

Example 9:
Minimum no. of 2 × 1 MUX required to implement 9 × 1 MUX.

Solution:



Figure

Example 10:
In realization of 32 : 1 MUX using 2 : 1 MUX, the required number of 2 : 1 MUX is ?

Solution: In realization of 2^n : 1 MUX using 2 : 1 MUX, the required number of 2 : 1 MUX is 2^n -1, since, we have to realize 32 : 1 MUX, so we have
n = 5
Hence, the required number of 2 : 1 MUX is
$2^n - 1 = 2^5 - 1 = 31$

DEMULTIPLEXER

The demultiplexer has one input line and m output lines. Again $m = 2^n$, so it requires n select lines. A demultiplexer with one input and m output is called a 1-to-m demultiplexer.

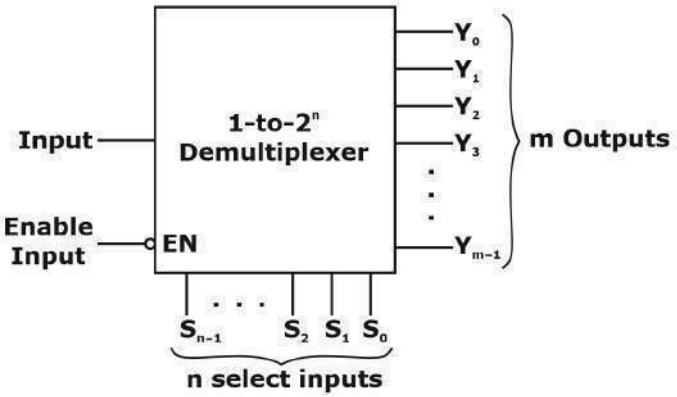


Figure : Block Diagram of m-output Demultiplexer

The demultiplexer has one input line and m output lines. Again m = 2^n, so it requires n select lines. A demultiplexer with one input and m outputs is called a 1-to-m demultiplexer.

1 x 2 Demultiplexer:

A 1 to 2 demultiplexer has one input and two outputs. Since 2 = 21, it requires only one control (select) line.

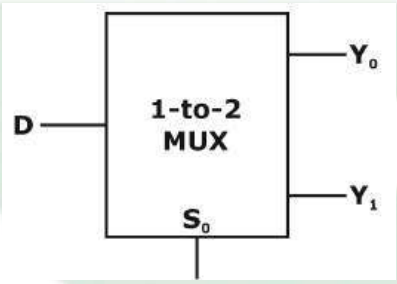
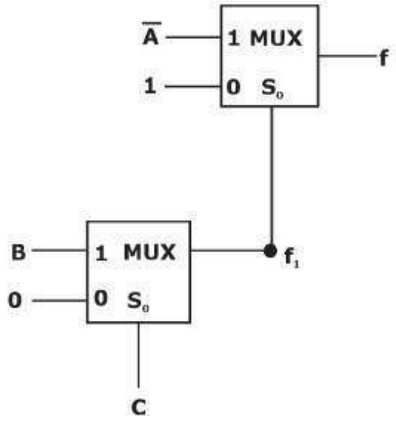


Figure : Logic Diagram of 1 x 2 De-MUX

Example:

The network shown below implements which gate?



Figure

Truth table of a 1-to-2 demultiplexer

Input Select input S	S_0	Output Y_1	Y_0
D	0	0	D
D	1	D	0

Table : Truth table of a 1-to-2 demultiplexer

Thus, the Boolean expressions for the outputs can be written as

$Y_0 = D\bar{S}_0$ & $Y_1 = DS_0$

Realization of a 1 x 2 Demultiplexer using Logic Gates:

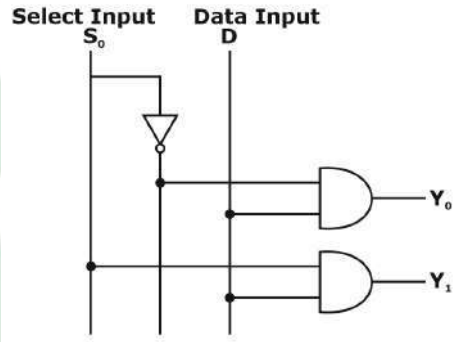


Figure: Logic Diagram of 1 x 2 Demultiplexer

Solution: The function f of the network is given by

$f = 1 \cdot \bar{S}_0 + \bar{A} \cdot S_0$

Output of MUX 1 is given to S_0 of the MUX-2, i.e.

$f_1 = 0 \cdot \bar{C} + B \cdot C$

$f_1 = B \cdot C$

Here, we get the output of the network as.

$f = 1 \cdot \bar{f}_1 + \bar{A} \cdot f_1$

$= \overline{BC} + \bar{A}BC$

$= (\overline{BC} + \bar{A})(\overline{BC} + BC)$

$= (\overline{BC} + \bar{A})$

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

$= \overline{ABC}$

i.e f is the output of a 3 input (A, B, C) NAND gate.

COMPARISON BETWEEN MULTIPLEXER AND DEMULTIPLEXER

S.No.	Parameter of comparison	Multiplexer	Demultiplexer
1.	Type of logic circuit	Combinational	Combinational
2.	Number of data inputs	m	1
3.	Number of select inputs	n	N
4.	Number of data output	1	M
5.	Relation between input/output lines and select lines	$m = 2^n$	$m = 2^n$
6.	Operation principle	Many to 1 or as data selector	1 to many or data distributor

Table : Comparison between Multiplexer and Demultiplexer

DECODER

A decoder is a combinational circuit that converts an n-bit binary input data into 2^n output lines, such that each output line will be activated for only one of the possible combinations of inputs. Decoders are usually represented as n-to-2^n line decoders, where n is the number of input lines and 2^n is the number of maximum possible output lines.

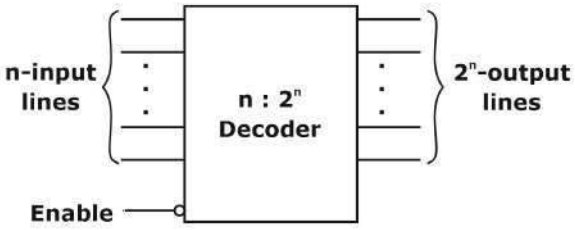


Figure: Block Diagram of n-to-2^n Decoder

If there are some unused or 'don't care' combinations in the n-bit code, then there will be less than 2n output lines. In general, if n and m are respectively the numbers of input and output lines, then m ≤ 2n.

2 to 4 Line Decoder:

Consider a 2 to 4-line decoder, where A and B are two inputs whereas Y_0 through Y_3 are the four outputs.

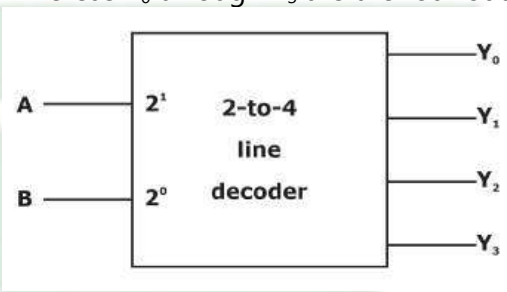


Figure : Block Diagram of a 2 to 4 Line Decoder

Truth Table of a 2 to 4 Line Decoder:

Inputs		Outputs			
A	B	Y_0	Y_1	Y_2	Y_3
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Table: Truth Table of a 2 to 4 Line Decoder:

The Boolean expressions for the four outputs is given as:

$Y_0 = \overline{A}\overline{B}$ and $Y_1 = \overline{A}B$

$Y_2 = A\overline{B}$ and $Y_3 = AB$

Realization of a 2 to 4 Line Decoder using Logic Gates:

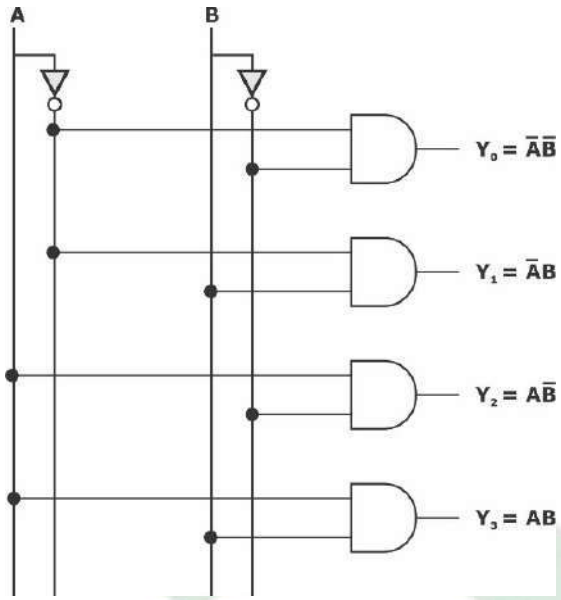


Figure: Logic Diagram of a 2 to 4 Line Decoder

Example 12:

Implement the following multi-output combinational logic circuit using a 4-to-16 line decoder.

$f_1 = \Sigma m(1, 2, 4, 7, 8, 11, 12, 13)$

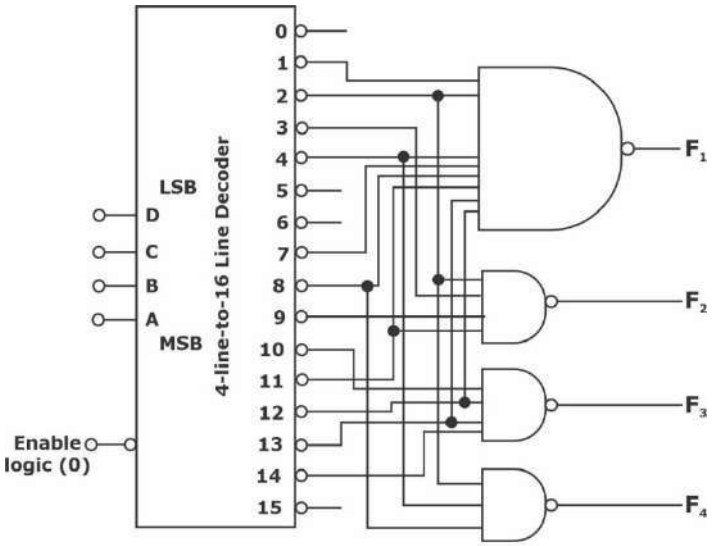
$f_2 = \Sigma m(2, 3, 9, 11)$

$f_3 = \Sigma m(10, 12, 13, 14)$

$f_4 = \Sigma m(2, 4, 8)$

Solution:

A 4-bit input ABCDABCD feeds a decoder, with F1=1F_1=1 for specific minterms, implemented using a NAND due to active-low outputs. Each output needs its own NAND. Multiplexer designs avoid extra gates, but decoders can be cheaper for multiple complex outputs, as one decoder with a few gates can replace several multiplexers.

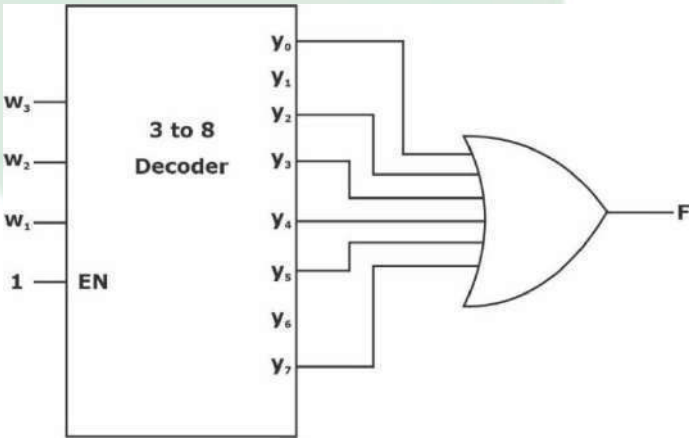


Implementation of Combinational Logic Circuit

Figure

Example 13:

The circuit shown below represents a Boolean expression. How many minterms are there in the Boolean expression?



Figure

Solution: In the given logic circuit, the enable input (EN) is 1. So, 3 to 8 decoder is activated.

So, 3-to-8 decoder is activated.

Therefore, the output function f is given by

$F = y_0 + y_2 + y_3 + y_4 + y_5 + y_7$

Or $F(w_1, w_2, w_3)$

$= \bar{w}_1 \bar{w}_2 \bar{w}_3 + \bar{w}_1 w_2 \bar{w}_3 + \bar{w}_1 w_2 w_3 +$

$w_1 \bar{w}_2 \bar{w}_3 + w_1 \bar{w}_2 w_3 + w_1 w_2 w_3$

$= 000 + 010 + 011 + 100 + 101 + 111$

$= \Sigma m(0, 2, 3, 4, 5, 7)$

i.e. Boolean expression with 6 min-terms.

ENCODERS

An encoder is a combinational logic circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines.

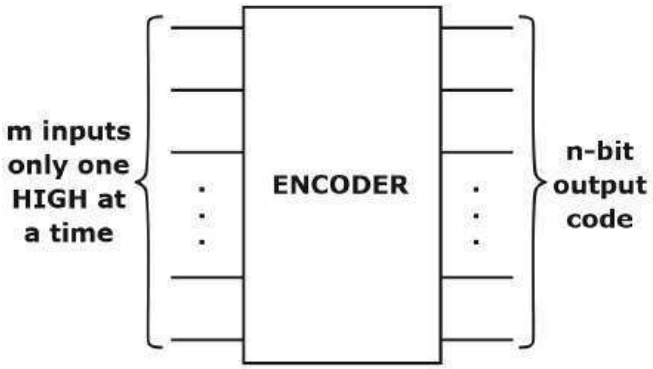


Figure : Block Diagram of Encoder

Truth Table of an Octal to Binary Encoder:

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	A	B	C
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	0
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	0

Octal to Binary Encoder:

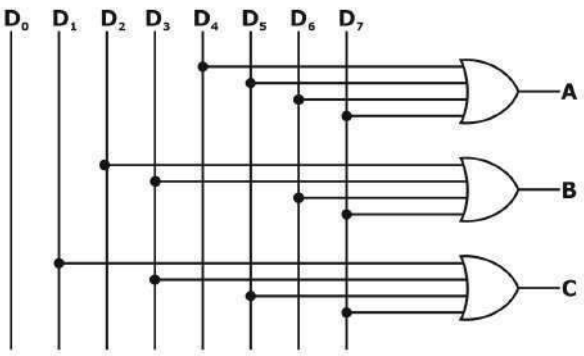


Figure: Logic Diagram of Octal-to Binary Encoder

Decimal to BCD Encoder: This type of encoder has 10 inputs one for each decimal digit and 4 outputs corresponding to the BCD code.

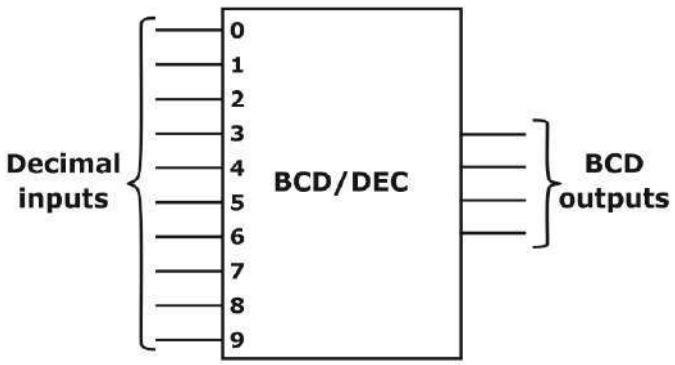


Figure: Block Diagram of a Decimal-to-BCD Encoder

Truth Table of a Decimal to Binary Encoder:

Input										Output			
0	1	2	3	4	5	6	7	8	9				
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	D_8	D_9	A	B	C	D
1	0	0	0	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	0	0	0	1	0	0
0	0	0	0	0	1	0	0	0	0	0	1	0	1
0	0	0	0	0	0	1	0	0	0	0	1	1	0
0	0	0	0	0	0	0	1	0	0	0	1	1	1
0	0	0	0	0	0	0	0	1	0	1	0	0	0
0	0	0	0	0	0	0	0	0	1	1	0	0	1

Table: Truth Table of a Decimal to Binary Encoder

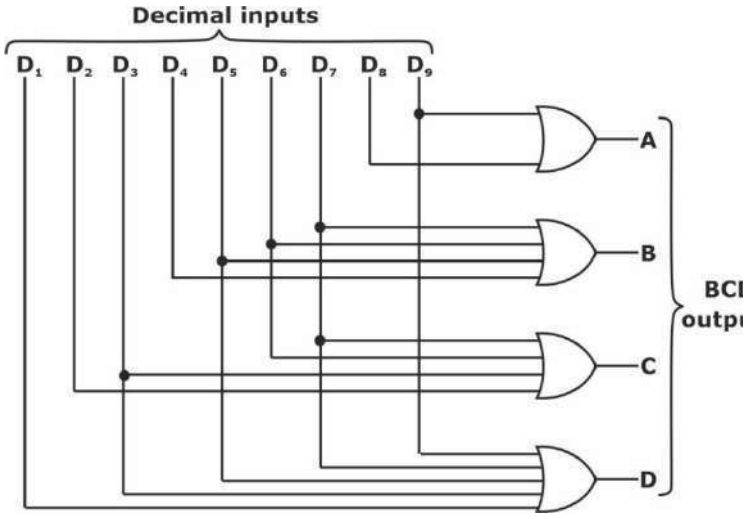


Figure: Logic Diagram of Decimal-to-BCD encoder

The outputs of a decimal-to-BCD encoder:

$A = D_8 + D_9$

$B = D_4 + D_5 + D_6 + D_7$

$C = D_2 + D_3 + D_6 + D_7$

$D = D_1 + D_3 + D_5 + D_7 + D_9$

PRIORITY ENCODER

Truth Table of a Four Input Priority

Encoder: (Taking LSB as priority)

Inputs				Outputs	
D_0	D_1	D_2	D_3	A	B
0	0	0	0	X	X
1	0	0	0	0	0
X	1	0	0	0	1
X	X	1	0	1	0
X	X	X	1	1	1

Table: Truth Table of a Four Input Priority Encoder

According to the truth table, the higher the subscript number, the higher the priority of the input.
The X's are don't care conditions indicating that the binary values they represent may be equal to 0 or 1.

CODE CONVERTERS

A code converter is a combinational logic circuit which accepts the input information in one binary code, converts it and produces an output into another binary code.

The truth table for 4-bit Binary and its Equivalent BCD:

Decimal	Binary Input				BCD Output				
	A	B	C	D	B_4	B_3	B_2	B_1	B_0
0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	0	1
2	0	0	1	0	0	0	0	1	0
3	0	0	1	1	0	0	0	1	1
4	0	1	0	0	0	0	1	0	0
5	0	1	0	1	0	0	1	0	1
6	0	1	1	0	0	0	1	1	0
7	0	1	1	1	0	0	1	1	1
8	1	0	0	0	0	1	0	0	0
9	1	0	0	1	0	1	0	0	1
10	1	0	1	0	1	0	0	0	0
11	1	0	1	1	1	0	0	1	0
12	1	1	0	1	1	0	0	1	0
13	1	1	0	1	1	0	0	1	1
14	1	1	1	0	1	0	1	0	0
15	1	1	1	1	1	0	1	0	1

Table : Truth table for 4-bit Binary and its Equivalent BCD

The minimized expression of outputs are as follows:

$B_4 = AB + AC$

$B_1 = \overline{A}C + AB\overline{C}$

$B_2 = \overline{A}B + BC$

$B_3 = A\overline{B}\overline{C}$

$B_0 = D$

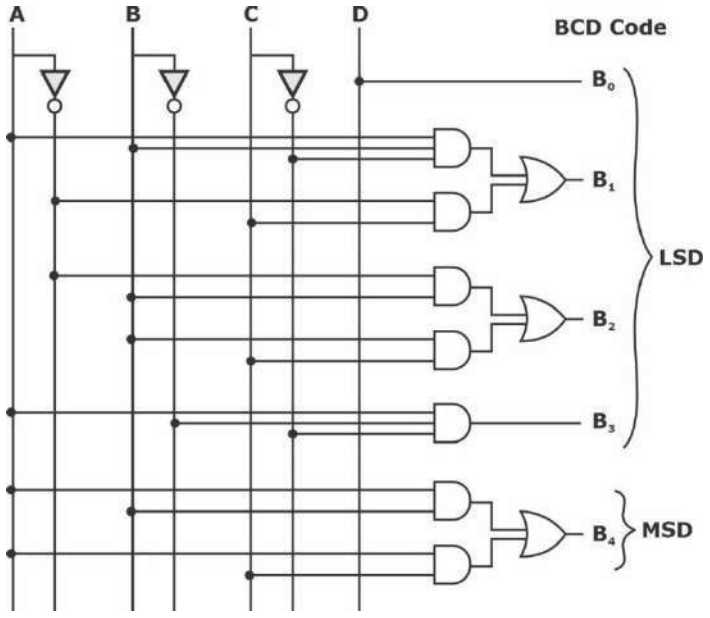


Figure: Logic Diagram of a Binary-to-BCD Code Converter

PARITY GENERATOR

Parity generators are circuits that accept an (n-1) bit data stream and generate an extra bit that is transmitted with the bit stream. This extra bit is referred to as the parity bit. The parity added in binary message is such that the total number of 1's in the message can be either odd or even according to the type of parity used.

Even Parity Generator:

The even parity generator is a combinational logic circuit that generates the parity bit such that the number of 1's in the message becomes even. The parity bit is '1' if there are odd number of 1's in the data stream and the parity bit is '0' if there are even number of 1's in the data stream.

Truth table for 4-bit data with Even Parity:

4-bit data				Even Parity
A	B	C	D	P
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	0
1	0	1	0	0
1	0	1	1	1
1	1	0	0	0
1	1	0	1	1
1	1	1	0	1
1	1	1	1	0

Table: Truth table for 4-bit data with Even Parity

The minimized expression for even parity generator is

$P = A \oplus B \oplus C \oplus D$

The logic diagram for the even parity generator is given as

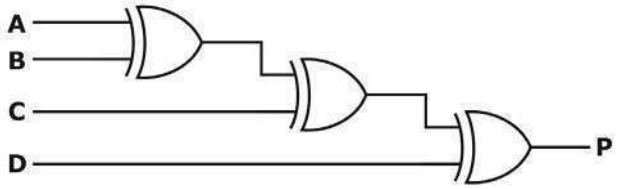


Figure: Logic diagram of even parity generator

Odd Parity Generator:

The odd parity generator is a combinational logic that generates the parity bit such that the number of 1's in the message becomes odd. The parity bit is '0' for odd number of 1's and '1' for even number of 1's in the bit stream.

Truth table for 4-bit data with Odd Parity:

4-bit data				Odd Parity
A	B	C	D	P
0	0	0	0	1
0	0	0	1	0
0	0	1	0	0
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	1
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	1
1	0	1	1	0
1	1	0	0	1
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

Table : Truth table for 4-bit data with Odd Parity

The minimized expression for odd parity generator is
$P = (A \oplus C) \odot (B \oplus D)$
The logic diagram of odd parity generator is given as

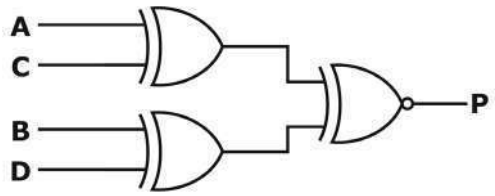
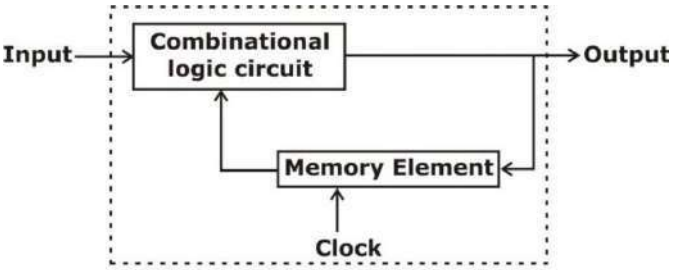


Figure: Logic Diagram of Odd Parity Generator

SEQUENTIAL LOGIC CIRCUITS

In sequential logic circuits, the output is a function of the present inputs as well as previous state of inputs and outputs. Sequential circuit include memory elements to store past data.



There are two types of sequential circuits:

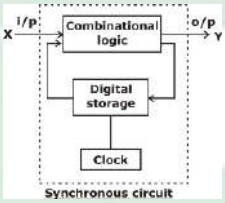
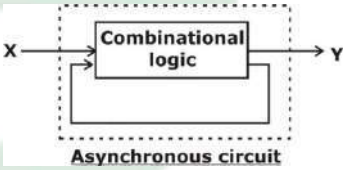
1. Synchronous Circuits:

The sequential circuits which are controlled by a clock are called synchronous sequential circuits. These circuits get activated only when clock signal is present.

2. Asynchronous Circuits:

The sequential circuits which are not controlled by a clock are called asynchronous sequential circuits, i.e. the sequential circuits in which events can take place any time the inputs are applied are called asynchronous sequential circuits.

DIFFERNCE BETWEEN SYNCHRONOUS AND ASYNCHRONUS SEQUENTIAL CIRCUITS

S.No.	Synchronous Sequential Circuits	Asynchronous Sequential Circuits
1.	In synchronous circuits, the change in input signals can affect memory elements upon activation of clock signal.	In asynchronous circuits, change in input signals can affect memory elements at any instant of time.
2.	In synchronous circuits, memory elements are clocked FF's.	In asynchronous circuits, memory elements are either unclocked FF's or time delay elements.
3.	The maximum operating speed of the clock depends on time delays involved.	Since the clock is not present, asynchronous circuits can operate faster than synchronous circuits.
4.	They are easier to design.	More difficult to design.
5.		

LATCHES

Flip-Flop is a one-bit memory device and it can store either 1 or 0.

General Block Diagram of a Latch or Flip-flop:

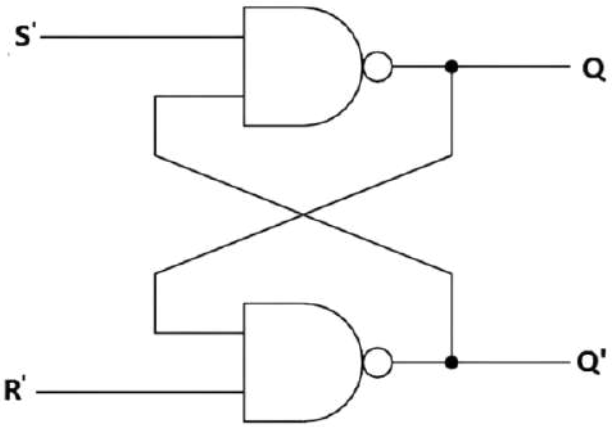
Figure shown below is the general type of symbol used for a latch. In case of a flip-flop, a clock signal must be shown at input side. It has many inputs and two outputs, labelled Q and $\overline{Q}$. The Q output is the normal output of the latch and $\overline{Q}$ is the inverted output.

Note: A flip-flop is said to be in HIGH state or logic 1 state or SET state when Q = 1, and in LOW state or logic 0 state or RESET state or CLEAR state when Q = 0.

Difference between Latches and Flip-flops:

S.No.	Latch	Flip-flop
1.	A latch is an electronic sequential logic circuit used to store information in an asynchronous arrangement.	A flip-flop is an electronic sequential logic circuit used to store information in a synchronous arrangement. It has two stable states and maintains its states for an indefinite period until a clock pulse is applied.
2.	One latch can store one-bit information, but output state changes only in response to data input.	One flip-flop can store one-bit data, but output state changes with clock pulse only.
3.	Latch is an asynchronous device and it has no clock input.	Flip-flop has clock input and its output is synchronised with clock pulse.
4.	Latch holds a bit value and it remains constant until new inputs force it to change.	Flip-flop holds a bit value and it remains constant until a clock pulse is received.
5.	Latches are level-sensitive, and the output tracks the input when the level is high. Therefore, as long as the level is logic level 1, the output can change if the input changes.	Flip-flops are edge sensitive. They can store the input only when there is either a rising or falling edge of the clock.

LATCH

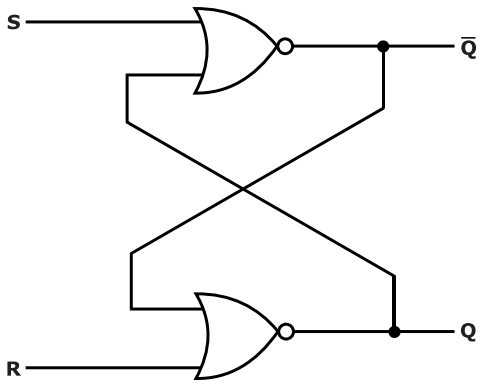


S	R	Qn+1
0	0	0 (Hold)
0	1	0 (Reset)
1	0	1 (Set)
1	1	Forbidden

A latch is a type of bistable multivibrator. The main characteristics of latch is that the output is not dependent solely on the on the present state of the input but also on the proceeding output state.
Latches are sometimes used for multiplexing data onto a bus.

SR Latch:

For the SR latch (S stands for set and R for reset). The logic circuit for SR latch is shown in figure below:

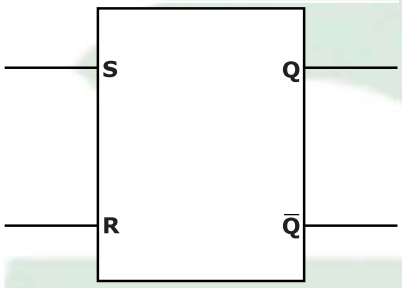


Logic circuit of SR latch.

The state table for the SR latch is:

S	R	Q	Q^+	$\overline{Q^+}$
0	0	0	0	1
0	0	1	1	0
0	1	0	0	1
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	0	0
1	1	1	0	0

The symbol for SR Latch is:



Obtaining the characteristic equations of the NOR gate based latch are; we get

$Q^+ = \overline{R} \cdot S + \overline{R} \cdot Q = \overline{R} \cdot (S + Q)$

and

$\overline{Q}^+ = \overline{S} \cdot R + \overline{S} \cdot \overline{Q} = \overline{S} \cdot (S + \overline{Q})$

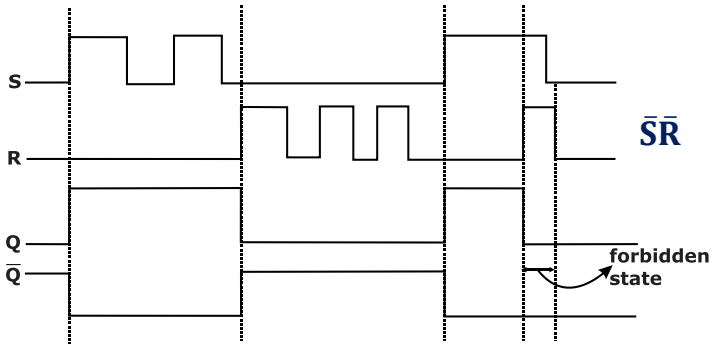
Note: It must be noted that the complementing Q^+ does not yield $\overline{Q}^+$.

S	Q	Q^+	$\overline{Q^+}$	
0	0	Q	$\overline{Q}$	⇒ **No change**
1	1	0	1	⇒ **Reset Q^+ to 0**
1	0	1	0	⇒ **Set Q^+ to 1**
1	1	0	0	⇒ **Forbidden state**

Hence, the truth table for SR latch is

However, the forbidden state (S = R = 1) is considered a don't care state.

Consider a Timing diagram for SR latch



Latch:

An $\overline{S}\overline{R}$ latch can be implemented using NAND gates, as shown in figure below

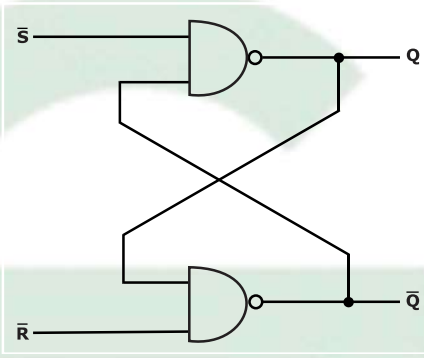
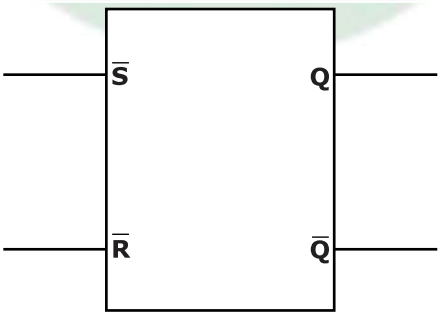


Figure : Logic circuit for $\overline{S}\overline{R}$ Latch

The $\overline{S}\overline{R}$ latch is said to be set-dominant 1,
The symbol for $\overline{S}\overline{R}$ latch is shown below:



The truth table for $\overline{S}\overline{R}$ latch is given as:

$\overline{S}$	$\overline{R}$	Q^+	$\overline{Q^+}$	
1	1	Q	$\overline{Q}$	⇒ **No change**
1	0	0	1	⇒ **Reset Q^+ to 0**
0	1	1	0	⇒ **Set Q^+ to 1**
1	1	1	1	⇒ **Forbidden state**

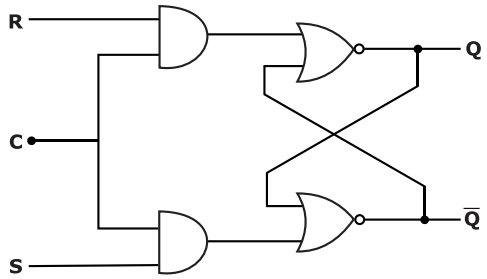


Figure : Logic circuit of clocked SR latch.

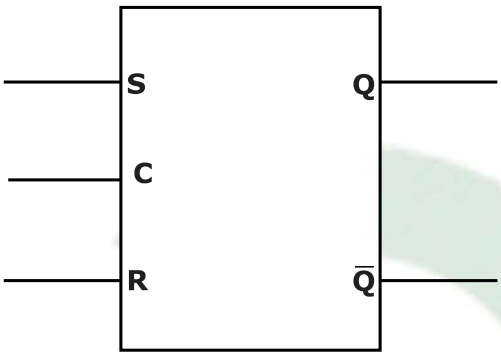


Figure : Symbol for clocked SR latch

C	S	R	Q^+	$\overline{Q^+}$	
0	X	X	Q	$\overline{Q}$	} No change state
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	0	0	⇒ Forbidden state

Gated $\overline{S}\overline{R}$ Latch or enable $\overline{S}\overline{R}$ Latch or clocked $\overline{S}\overline{R}$ Latch:

Gated $\overline{S}\overline{R}$ latch is implemented using two NAND gates and an $\overline{S}\overline{R}$ latch.

The logic circuit diagram, symbol and truth is given as

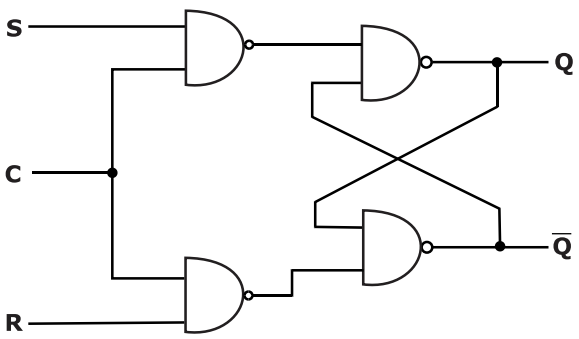
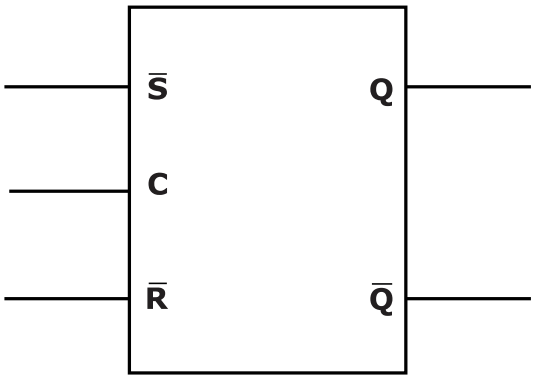


Figure : logic diagram of clocked $\overline{S}\overline{R}$ latch.



The truth table of the gated SR latch based on a $\overline{S}\overline{R}$ latch:

C	S	R	Q^+	$\overline{Q^+}$
0	X	X	Q	$\overline{Q}$
1	0	0	Q	$\overline{Q}$
1	0	1	0	1
1	1	0	1	0
1	1	1	1	1

The characteristic equation for SR flip-flop is given as

$Q^+ = Q_{n+1} = S + \overline{R}Q_n = S + \overline{R}Q$

FLIP-FLOPS

Flip-flops are edge-triggered or edge-sensitive whereas gated latches are level-sensitive.

Edge-triggered flip-flop:

An edge-triggered flip-flop changes its state either at positive edge (rising edge) or at negative edge (falling edge) of the check pulse.

Positive edge triggered has no bubble at input C whereas negative edge triggered has bubble at input C.

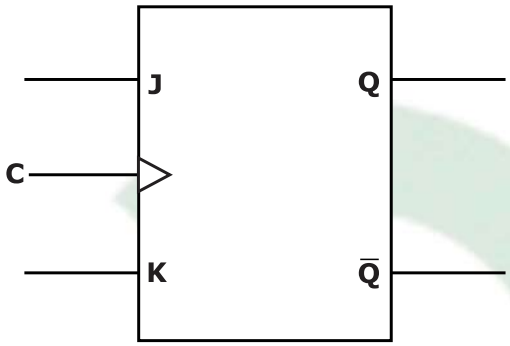


Figure : Positive edge triggered flip-flop.

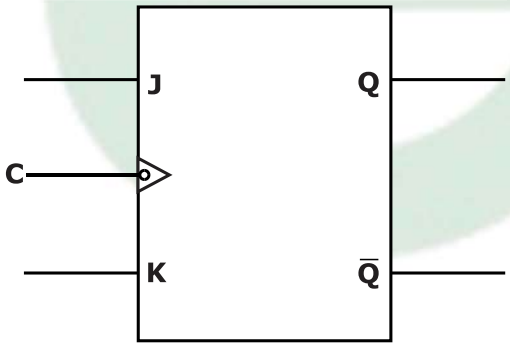


Figure : Negative edge-triggered flip-flop.

Basic JK flip-flop:

JK Flip-flop (J as a set input and K and K as a reset input) is the most versatile of the basic flip-flops.

The logic circuit of the gated JK flip-flop is shown in **figure below**:

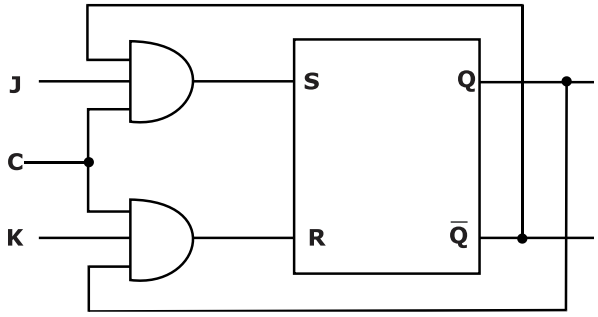


Figure : Logic circuit diagram of clocked JK flip-flop.

The state table for the JK flip-flop is given as

C	J	K	Q	Q^+
O	X	X	X	Q
1	0	0	0	0
1	0	0	1	1
1	0	1	0	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	1	0
1	1	1	1	0

Hence, the truth table becomes,

C	J	K	Q^+	$\overline{Q^+}$	
0	X	X	Q	$\overline{Q}$	⇒ No change State
1	0	0	Q	$\overline{Q}$	
1	0	1	0	1	⇒ Reset
1	1	0	1	0	⇒ Set
1	1	1	$\overline{Q}$	Q	⇒ Toggle

Note: The forbidden state, inherent to SR flip-flop is eliminated by adding two feedback loops such that the output becomes 1 only if Q = 0 and reset to only if Q = 1.

It should also be noted that when the inputs (J &K) are set to 1 and clock signal change to 1, then the feedback value of Q & $\overline{Q}$ forced the slip-flop is toggle its value.

(i.e. to switch its state to its logical complement) hence, to ensure this operation in smooth fashion, the pulse width of the clock must be smaller than the propagation delay of the flip-flop.

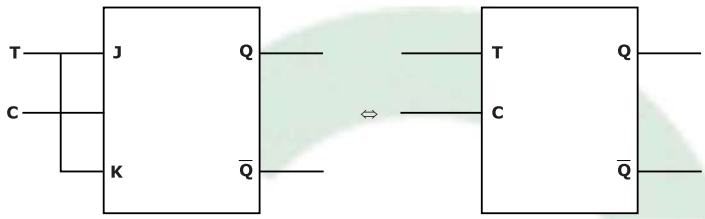
The characteristic equation of the JK flip-flop

$Q_{n+1} = J\bar{Q}_n + \bar{K}Q_n$ or $Q^+ = J\bar{Q} + \bar{K}Q$

T-flip-flop:

A JK flip-flop can be transformed into a T- flip-flop (T stands for Toggle). When T flip-flop is activated, its output changes its state at every time a pulse is applied to the input T.

The logic circuit of the gated T flip-flop is.



The state or characteristic table for T flip-flop is

C	T	Q	Q^+
O	X	X	Q
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

As J = K = T, we obtain the characteristic equation as

$Q^+ = T \cdot \bar{Q} \cdot C + (\bar{T} + \bar{C}) \cdot Q$

If C = 1, the characteristic the equation is reduced to

$Q^+ = T \oplus Q$

If C = 0, $Q^+ = Q$

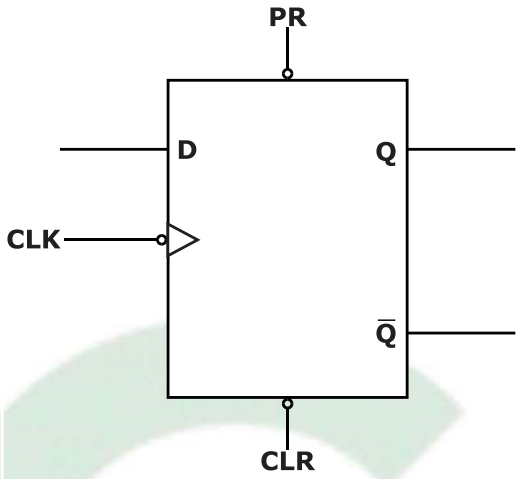
Hence, the truth table of the T-flip flop is given as

C	T	Q^+	$\overline{Q^+}$	
0	X	Q	$\bar{Q}$	} ⇒ No change state
1	0	Q	$\bar{Q}$	
1	1	$\bar{Q}$	Q	⇒ Toggle

D Flip-Flop:

D-flip-flop can be obtained by use of only two combinations of S-R or J-K flip-flop. It has only one input i.e. D-input or data impact.

The logic symbol for D- flip-flop is given as



The truth table for D-flip-flop is

Input	Output
D_n	Q_{n+1}
0	0
1	1

The characteristic equation of D-flip-flop is:

$Q_{n+1} = D$

It is very important and useful design aid for sequential circuit.

The excitation table for flip-flops:

Present state	Next state	SR Flip-flop		JK Flip-flop		T Flip-flop	D Flip-flop
		S	R	J	K	T	D
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	1	0
1	1	X	0	X	0	0	1

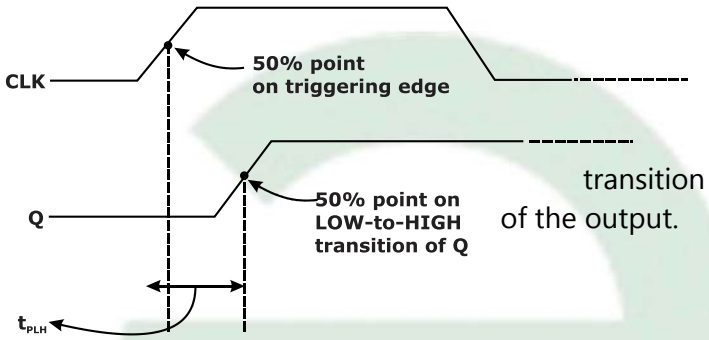
OPERATING CHARACTERISTICS OF FLIP-FLOPS

Propagation Delay Time:

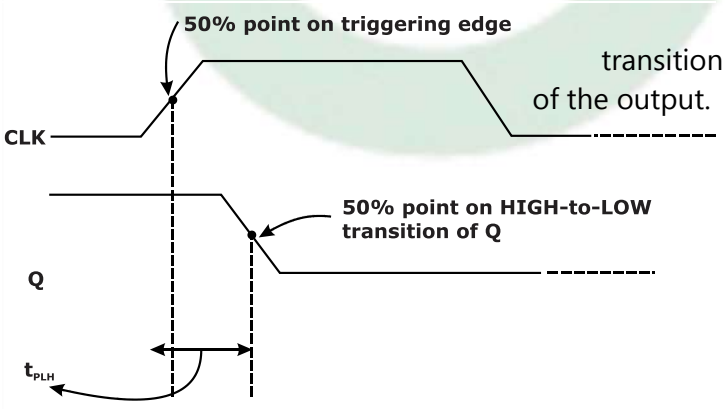
Propagation delay time is the time interval required after an input signal has been applied for the resulting output change to occur.

There are four categories of propagation delay times which are as follows:

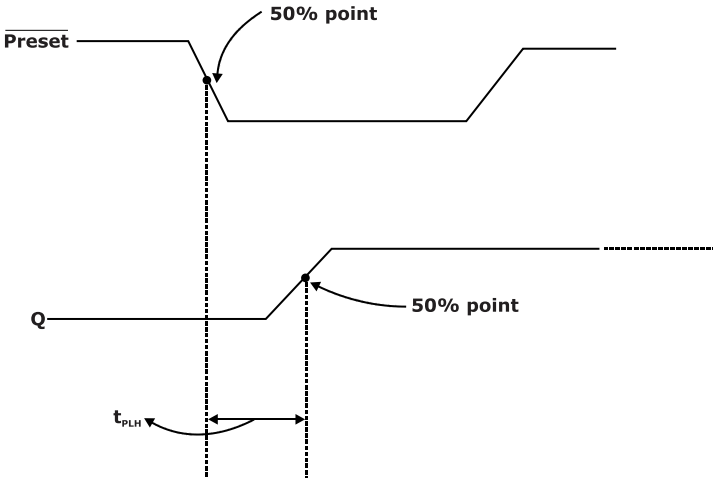
A. Propagation delay t_{PLH}, it is measured from the triggering edge of the clock pulse to Low-to-High transition of the output.



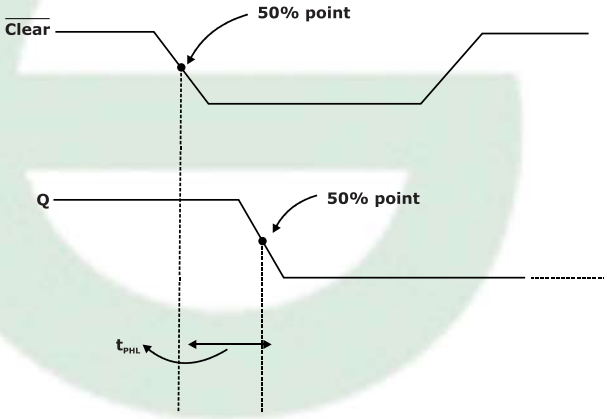
B. Propagation delay t_{PHL}, it is measured from the triggering edge of the clock pulse to HIGH-to-LOW transition of the output.



C. Propagation delay t_{PHL}, it is measured from the leading edge of the PRESET input to LOW-to-HIGH transition of the output.

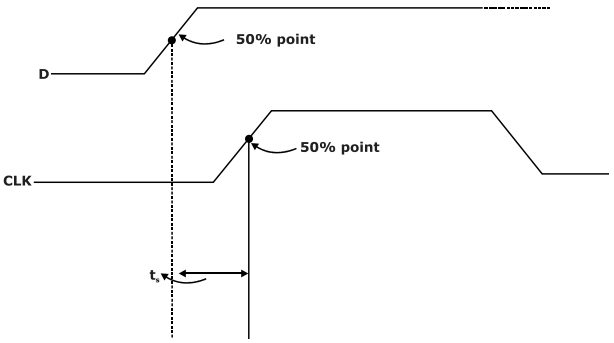


D. Propagation delay t_{PHL}, it is measured from the leading edge of the clear input to the HIGH-to-LOW transition of the output.



Set-up time (t_s):

It is the minimum time interval required for the logic levels (0 or 1) to be maintained constantly on the inputs (J, K or D) prior to the triggering edge of the clock pulse in order for the levels to be reliably clocked into the flip-flop.



Hold time (t_h):
It is the time for which the data must remain stable after the triggering edge of the clock.

Clock-pulse width:
The minimum time duration for which the clock pulse must remain HIGH and LOW which are designed by manufacturers. Failure to clock pulse width results in unreliable triggering.

Maximum clock frequency:
The maximum clock frequency (f_{max}) is the highest rate at which flip-flop can be reliably operated.

RACE AROUND CONDITION

JK flip flop suffers from the problem of race around condition. When J = 1 & K = 1, is applied to the JK flip flop and JK flip flop is level triggered then output of the JK flip flop toggles so many times during the pulse width of the clock and output of the flip flop settled either at 1 or 0 depending upon the pulse width of the clock and propagation delay of the flip flop is called race around condition.

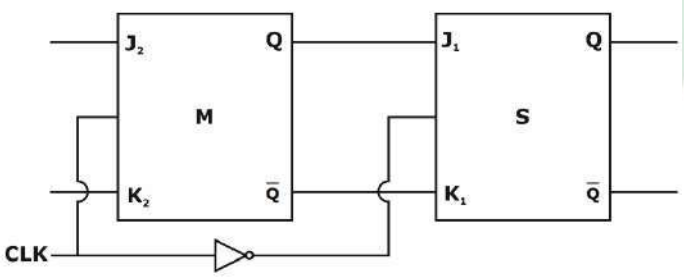


Figure: Race Around Condition in JK Flip Flop

To avoid Race Around condition:

- $T_{pulse\text{-}width} < T_{pd} < T_{clock}$
- Master Slave flip flop

Master Slave Flip flop:

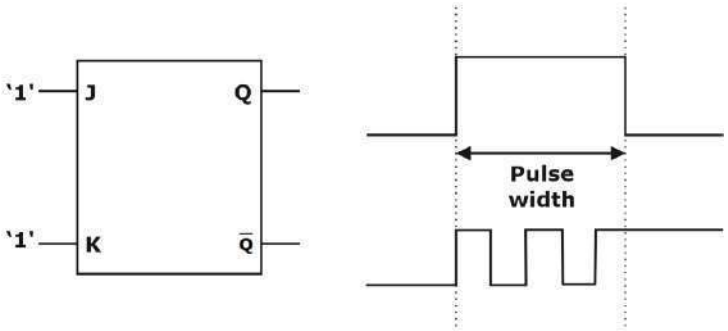
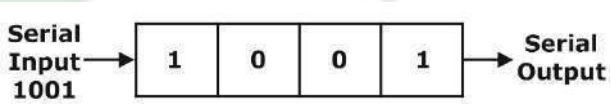


Figure : Logic Diagram for Master Slave JK Flip Flop

a. In master slave flip flop, inverted clock is given to the slave.
b. Master slave flip flop is used to store single bit because output is taken only from slave flip flop.



c. Here, master flip flop is level triggered while slave is negative edged triggered.

Note: JK flip flop is also known as Universal flip flop.

DESIGNING OF ONE FLIP FLOP BY OTHER FLIP FLOP

The steps for designing of one flip flop or new flip flop using existing or same existing flip flop.
Step 1: Write the characteristic table for the desired flip flop.
Step 2: Write the excitation table for the available flip flop.
Step 3: Write the logical expression.
Step 4: Minimize the logical expression.
Step 5: Circuit Implementation.

SHIFT REGISTERS

An array of flip-flops in required to store binary information, and the number of flip-flops required is equal to the number of bits used to store is referred as registers.

Examples of registers are general purpose registers flags, etc.
Now, the information or data can be stored or entered in serial form (one-bit at a time) or in parallel form (all the bits simultaneously) and can be retrieved like this manner too. The data will be entered or retrieved in serial form is known as temporal code and which is in parallel form is called spacial code.
Hence, registers can be classified into four categories depending upon the data being entered or retrieved.

SISO (serial-in, serial-out) Shift Register:

In serial-in, serial-out shift register, data input is in serial form and common clock pulse is applied to each of the flip-flop. After each clock pulse, data moves by one position. The output can be obtained in serial form, as shown in figure below:

Figure: SISO Shift Register

- It is the slowest shift register among all the shift registers.
- To store n-bits in a n-bit SISO register, then the minimum "n" clock pulses are required.
- To retrieve n-bits from a n-bit SISO register, then the minimum "(n-1)" clock pulses are required.

SIPO (serial-in, parallel-out) Shift Register:

In serial-in, parallel-out shift register, data is applied at the input of register in serial form and the output can be obtained in parallel form after the completely shifting of data in register. Figure below shows the serial input data, and then parallel output.

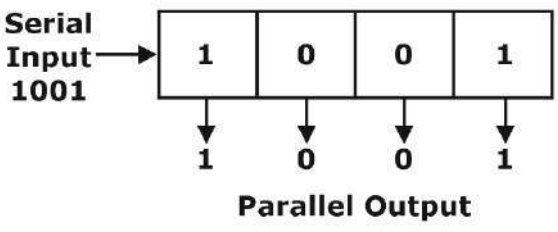


Figure : SIPO Shift Register

- To store n-bits in a n-bit SIPO register, the minimum "n" clock pulses are required.
- To retrieve n-bits from a n-bit SIPO register, there is no pulse required.

PISO (parallel-in, serial out) Shift Register:

In parallel-in, serial-out shift register, data is loaded into shift register in parallel form and the data output obtained will be serial form as shown in figure below:

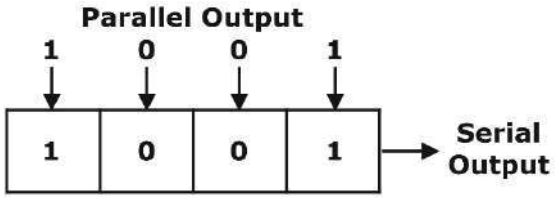


Figure : PISO shift register

- To store n-bit in a n-bit PISO register, a single clock pulse is required.
- To retrieve n-bit from n-bit PISO register, the minimum "(n-1)" clock pulses are required.

PIPO (parallel in, parallel out) Shift Register:

In parallel-in, parallel-out shift register, data is loaded in parallel form and the data output obtained will be in parallel, as shown in figure below:

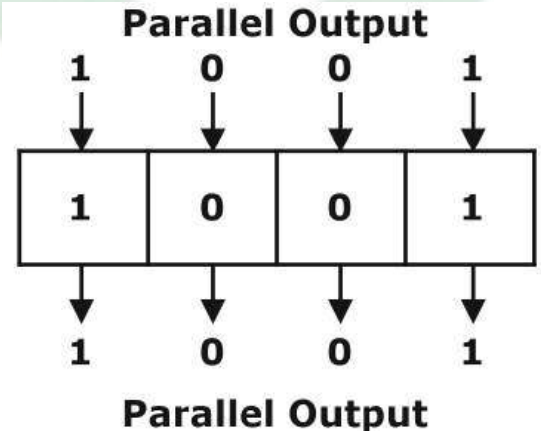
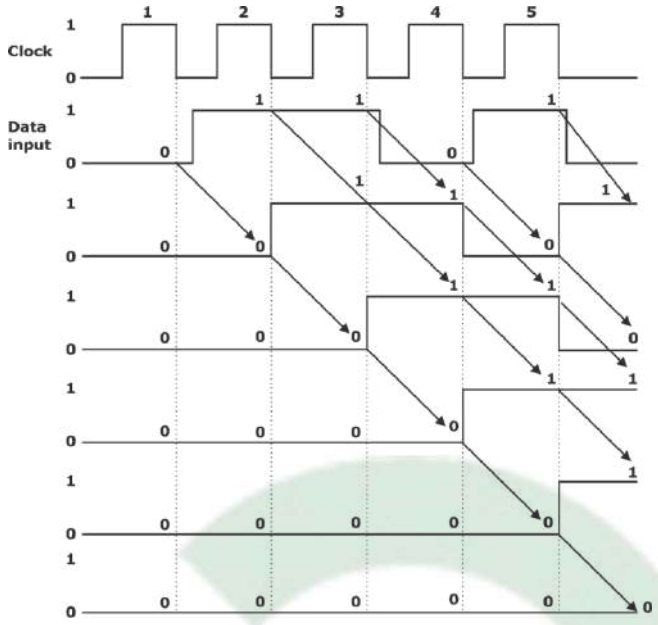


Figure : PIPO Shift Register

- To store n-bit in n-bit PIPO register, only a single clock pulse is required.
- To retrieve n-bits from n-bit PIPO register, no clock pulse is required.

Serial Input: The data in the serial form is applied at the serial input after clearing the flip-flops using CLR.



The waveform of serial input shift register is shown below:

Parallel Input: Data can be entered in the parallel form making use of the pre-set inputs. Then after clearing the flip-flops, if the data lines are connected to the parallel lines and '1' is applied to the PRESET input.

UNIVERSAL SHIFT REGISTER:

If the flip-flop outputs of a shift register are accessible, then information entered serially by shifting can be taking out in parallel from the outputs of the flip-flops. If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register. Some shift registers provide the necessary input and output terminals for parallel transfer. They may also have both shift-right and shift-left capabilities.

The most general shift register has the following capabilities:

1. A clear control to clear the register to 0.
2. A clock input to synchronize the operations.
3. A shift-right control to enable the shift-right operation and the serial input and output lines associated with the shift right.
4. A shift-left control to enable the shift-left operation and the serial input and output lines associated with the shift left.
5. A parallel-load control to enable a parallel transfer and the n input lines associated with the parallel transfer.
6. "n" parallel output lines.
7. A control state that leaves the information in the register unchanged in response to the clock.

   Other shift registers may have only some of the preceding functions, with at least one. shift operation. A register capable of shifting in one direction only is a unidirectional shift register. One that can shift in both directions is a bidirectional shift register. If the register can shift in both directions and has parallel-load capabilities, it is referred to as a universal shift register.

   The block diagram symbol and the circuit diagram of a four-bit universal shift register is shown in figure below:

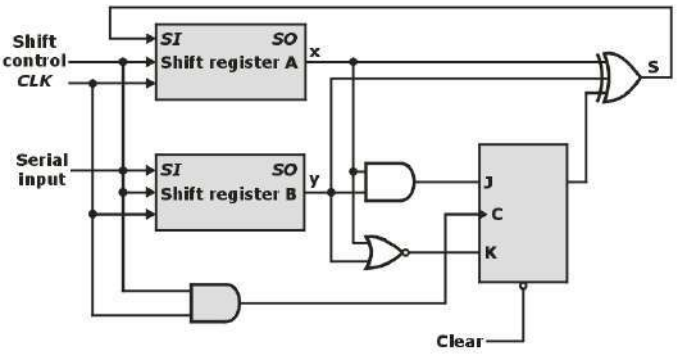


Figure : Second form of serial adder

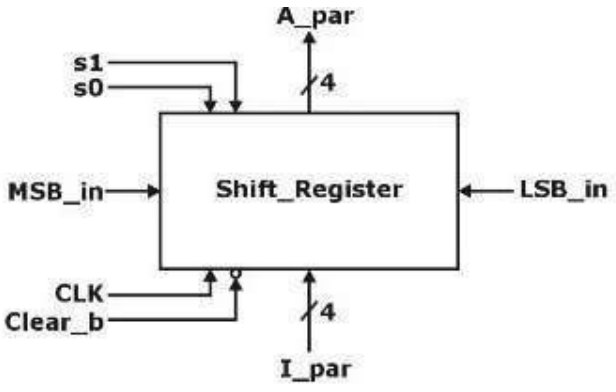


Figure : 4-bit Universal Shift Register

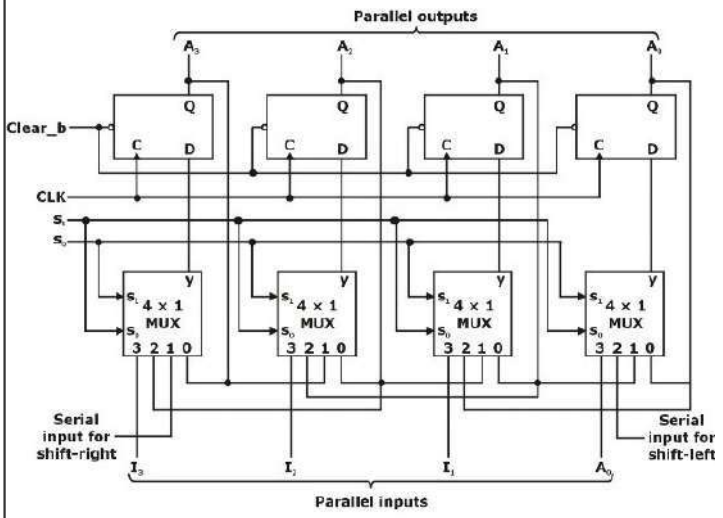


Figure : Logic Diagram of 4-bit Universal shift register

The function for the Universal Shift Register is as follows:

Mode Control		Register Operation
S_1	S_0	
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

Shift registers are often used to interface digital systems situated remotely from each other. For example, suppose it is necessary to transmit an n-bit quantity between two points. If the distance is far, it will be expensive to use n lines to transmit the its bits in parallel. It is more economical to use a single line and transmit the information serially, one bit at a time. The transmitter accepts the n-bit data in parallel into a shift register and then transmits the data serially along the common line. The receiver accepts the data serially into a shift register. When all n bits are received, they can be taken from the outputs of the register in parallel. Thus, the transmitter performs a parallel-to-serial conversion of data and the receiver does a serial-to-parallel conversion.

APPLICATIONS OF SHIFT REGISTERS

(a) Delay line: A shift register can be used to introduce a delay (Δt) in signals

$$\Delta t = N \times \frac{1}{f_c}$$

Where N is number of stages & f_c is the clock frequency.

(b) Serial-to-parallel converter

(c) Parallel-to-serial converter

(d) Ring counter

(e) Twisted ring counter

(f) Sequence counter

ASYNCHRONOUS COUNTER OR RIPPLE COUNTER

A circuit which is used for counting the numbers or pulses is known as counter. Counter is referred to as modulo-N (or divide by N), where the word modulo indicates the number of states in the counter.

3-BIT BINARY COUNTER:

Consider a 3-bit binary counter which has total '8' number of states which require three flip-flops and Q_2, Q_1 and R_0 are the outputs of those flip-flops.

The circuit diagram or logic circuit diagram for 3-bit binary counter,

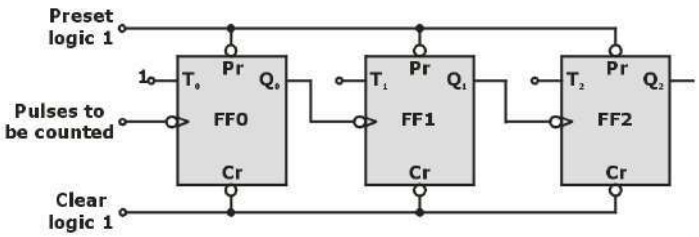
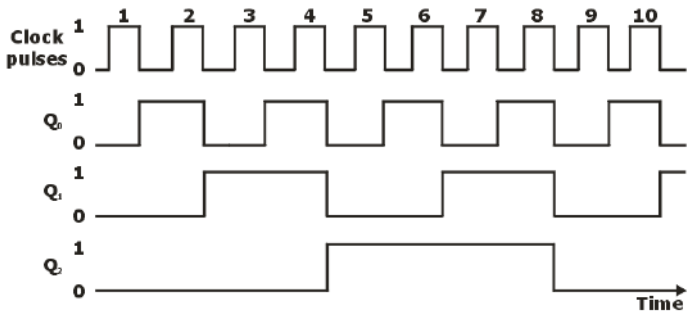


Figure : A 3-bit Binary Counter

The truth table for 3-bit binary counter is given as:

Counter state	Count		
	Q_2	Q_1	Q_0
0	0	0	0
1	0	0	1
2	0	1	0
3	0	1	1
4	1	0	0
5	1	0	1
6	1	1	0
7	1	1	1

Output waveforms of the above counter is:



The frequency 'f' of clock pulses for reliable operation of the counter is given as

$$\frac{1}{f} \geq N.t_{pd} + T_s$$

Where, N = number of flip-flops

t_{pd} = propagation delay of one flip-flop.

T_s = strobe pulse width.

If during the operation of counter, if some pulses are falsely operated for short duration, known as spikes, which change the state of the flip-flop. It may happen when the propagation delay of each flip-flop may vary and may happen that, all the flip-flops may not change their states or may be only one flip-flop changes its state during the pulse time.

This problem of spikes can be eliminated by using a strobe pulse with the help of strobe pulse, the state will charge only when flip-flops of the counter are in steady state.

Example 1:

In a 5-stage ripple counter, the propagation delay of a flip-flop is 100nsec. If the pulse width of the strobe is 50nsec. Find the maximum frequency at which the counter operator reliably.

Solution:

The maximum frequency is

$$f_{max} = \frac{1}{nt_{pd} + t_s}$$

n = number of flip-flops or stage = 5

t_{pd} = propagation delay of each flip-flop = 100 nsec.

t_s = Strobe pulse width = 50 nsec

$$f_{max} = \frac{1}{(5 \times 100 + 50) \times 10^{-9}} = \frac{1000}{(550)} \text{MHz}$$
$$= 1.818 \text{ MHz}$$

MOD-8 UP/DOWN COUNTER:

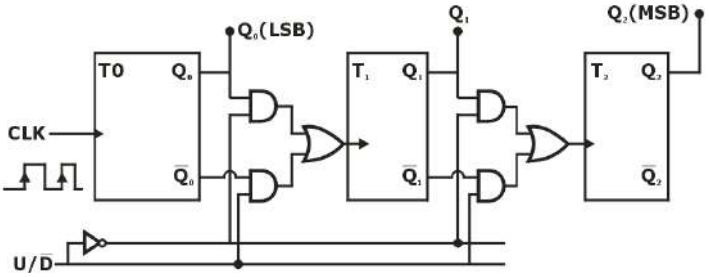


Figure : MOD-8 UP/DOWN Counter

DEMERITS OF ASYNCHRONUS COUNTER:

1. Only sequential counter can be designed. Random counter cannot be designed.
2. Glitch (undesirable state) would appear in case of asynchronous counter.
3. Speed of asynchronous counter is not fast.

BCD RIPPLE COUNTER

A decimal counter follows a sequence of 10 states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, since a decimal digit is represented by a binary code with at least four bits. The sequence of states in a decimal counter is dictated by the binary code used to represent a decimal digit. If the BCD code is used, the sequence of states is as shown in the state diagram. A decimal counter is similar to a binary counter, except that the state after 1001 (the code for decimal digit 9) is 0000 (the code for decimal digit 0).

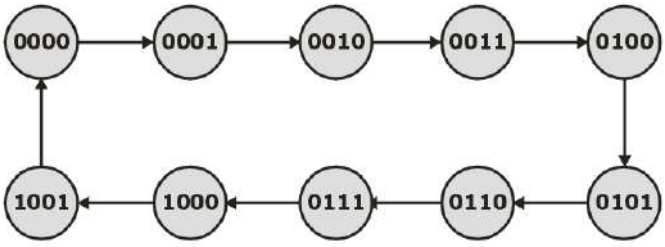


Figure State diagram of a decimal BCD counter.

The logic diagram of a BCD ripple counter using JK flip-flops is shown in figure below. The four outputs arc designated by the letter symbol Q, with a numeric subscript equal to the binary weight of the corresponding bit in the BCD code. Note that the output of Q_1 is applied to the C inputs of both Q_2 and Q_8 and the output of Q_2 is applied to the

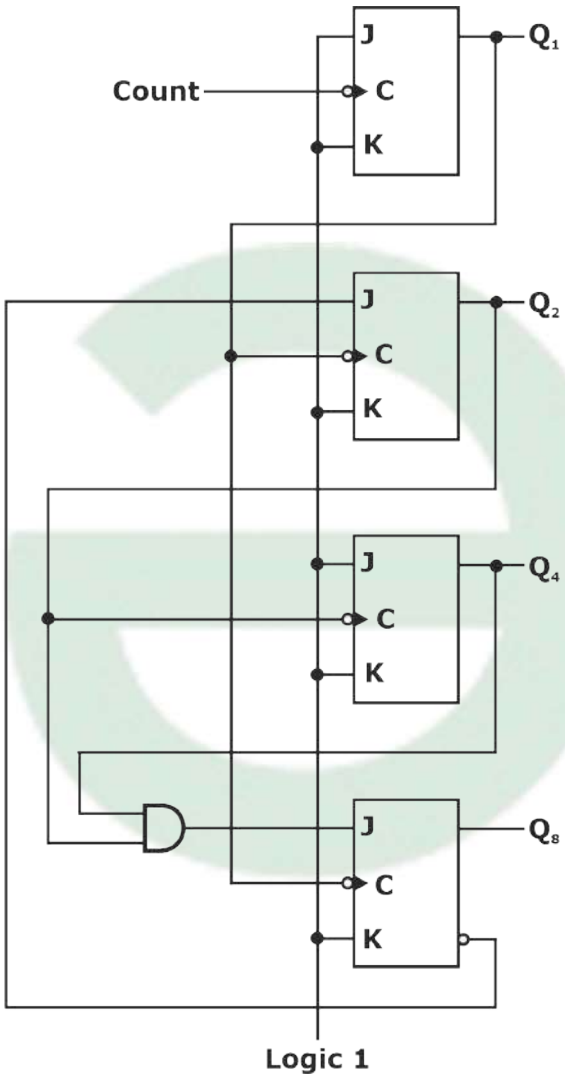


Figure : BCD ripple counter

SYNCHRONOUS COUNTERS

The ripple counters have the advantage of simplicity (only FLIP-FLOP's are required) but their speed is low because of ripple action. The maximum time is required when the output changes from 111....1 to 00....0 and this limits the frequency of operation of ripple counters.

The speed of operation improves significantly if all the FLIP-FLOPS are clocked simultaneously. The resulting circuit is known as a synchronous counter. Synchronous counters can be designed for any count sequence (need not be straight binary).

The output Q_0 of the least-significant FLIP-FLOP changes for every clock pulse. This can be achieved by using a T-type FLIP-FLOP with $T_0 = 1$. The output Q_0 changes whenever Q_0 changes from 1 to 0. Therefore, if Q_0 is connected to T input (T_1) of the next FLIP-FLOP, Q_1 will change from 1 to 0 (or 0 to 1) when $Q_0 = 1$ ($T_1 = 1$) and will remain unaffected when $Q_0 = T_1 = 0$. Similarly, Q_2 changes whenever Q_1 and Q_0 are both "1". This can be achieved by making the T-input (T_2) of the most-significant FLIP-FLOP equal to $Q_1.Q_0$.

In addition to FF's, synchronous counters require some gates also. JK FLIP-FLOPS are the most commonly used FLIP-FLOP's for the design of synchronous counters. In this, each FLIP-FLOP has two control inputs (J and K) and circuit is required to be designed for each control input. Many programmable logic devices (PLDs) used for the design of digital systems utilise D FLIP-FLOPS for their memory elements, therefore, counter design using D FLIP-FLOPS will be useful for programming inside a PLD. It has only one control input which makes its design simpler than the design using J-K FLIP-FLOPS.

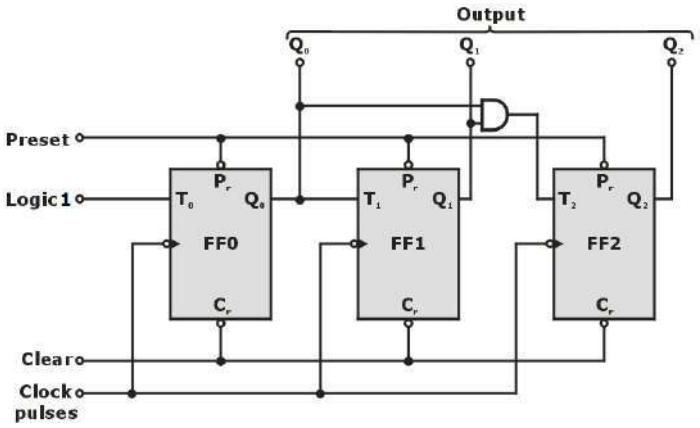


Figure A 3-bit Synchronous Counter

Synchronous Counter Design:

Synchronous counters for any given count sequence and modulus can be designed in the following way:

1. Find the number of FLIP-FLOPs required.
2. Write the count sequence in the tabular form.
3. Determine the FLIP-FLOP inputs which must be present for the desired next state from the present state using the excitation table of the FLIP-FLOPS.
4. Prepare K-map for each FLIP-FLOP input in terms of FLIP-FLOP outputs as the input variables.
5. Simplify the K-maps and obtain the minimized expressions.
6. Connect the circuit using FLIP-FLOPS and other gates corresponding to the minimized expressions.

Example 2:

Design a 3-bit synchronous counter using JK Flip-Flops.

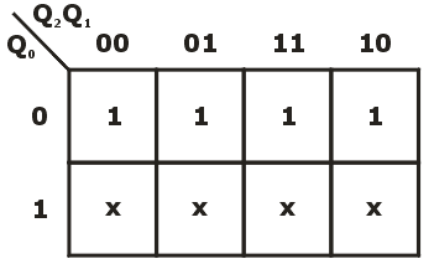
Solution:

The number of FLIP-FLOPs required is 3. Let the FLIP-FLOPs be FF0, FF1, FF2 and their inputs and outputs are given below:

FLIP-FLOP	Inputs	Outputs
FF0	J_0, K_0	Q_0
FF1	J_1, K_1	Q_1
FF2	J_2, K_2	Q_2

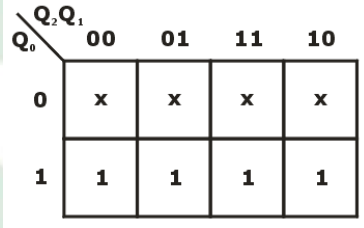
The count sequence and the required inputs of FLIP-FLOPs is shown below.

			FLIP-FLOP INPUTS					
Counter state			FF0		FF1		FF2	
Q_2	Q_1	Q_0	J_0	K_0	J_1	K_1	J_2	K_2
0	0	0	I	X	0	X	0	X
0	0	1	X	I	1	X	0	X
0	1	0	I	X	X	0	0	X
0	1	1	X	I	x	1	1	X
1	0	0	I	X	0	X	X	0
1	0	1	X	I	1	X	X	0
1	1	0	I	X	X	0	X	0
1	1	1	x	I	x	1	x	1
0	0	0						



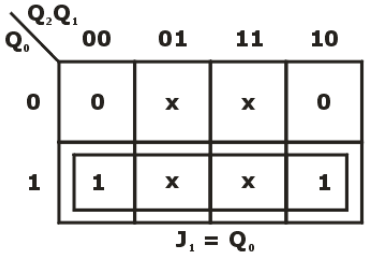
$J_0 = 1$

(a)



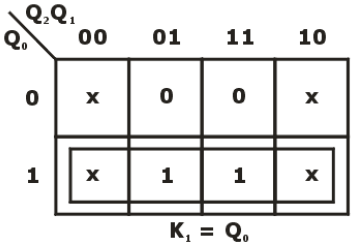
$K_0 = 1$

(b)



$J_1 = Q_0$

(c)



$K_1 = Q_0$

(d)

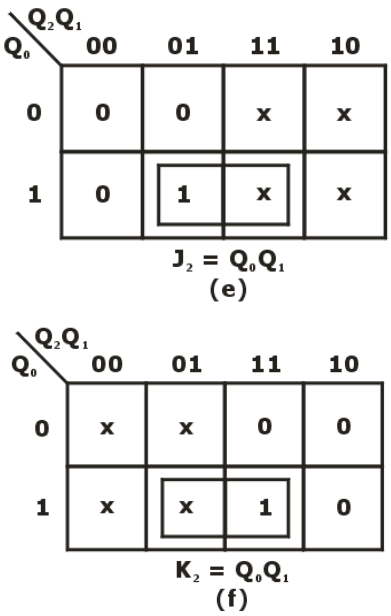


Figure K-Maps of 3-bit Synchronous Counter

Example 3:

Design a natural binary sequence mod-8 synchronous counter using D FLIP—FLOPS.

Solution:

The number of FLIP-FLOPS required is 3. Let the FLIP—FLOPS be FF0, FF1 and FF2 with inputs D_0, D_1 and D_2, respectively. Their outputs are Q_0, Q_1, and Q_2 respectively.

Counter state			FLIP-FLOP inputs		
Q_2	Q_1	Q_0	D_0	D_1	D_2
0	0	0	1	0	0
0	0	1	0	1	0
0	1	0	1	1	0
0	1	1	0	0	1
1	0	0	1	0	1
1	0	1	0	1	1
1	1	0	1	1	1
1	1	1	0	0	0
0	0	0			

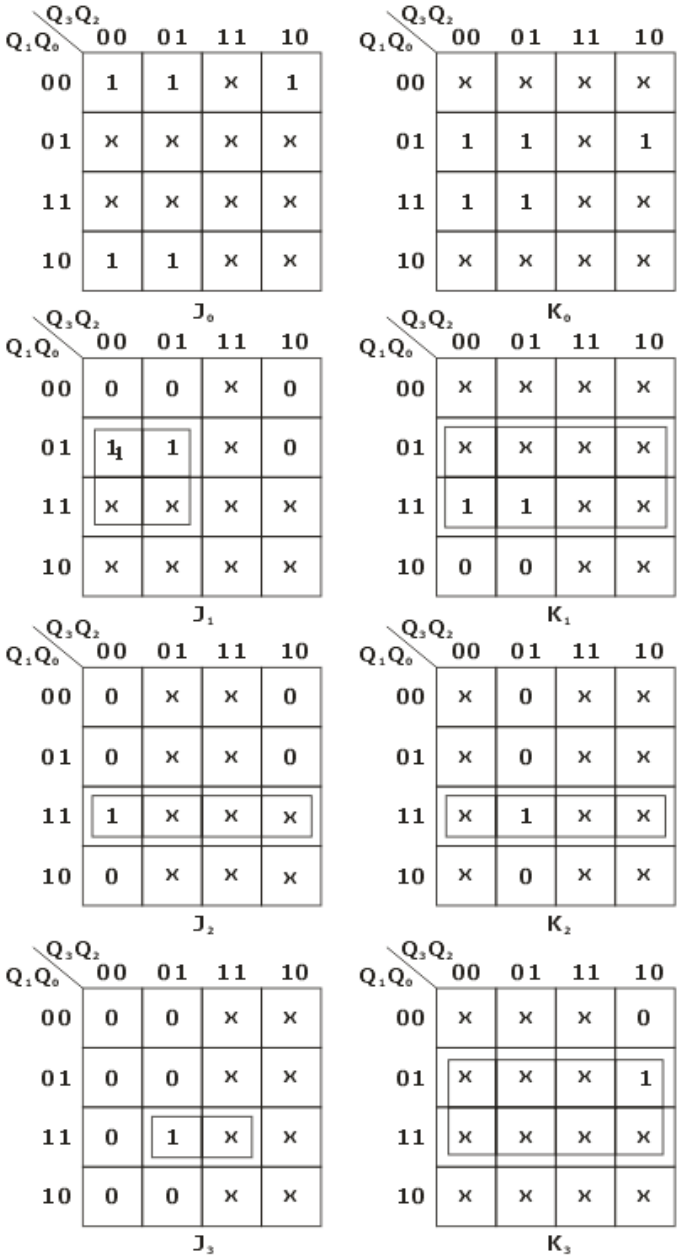
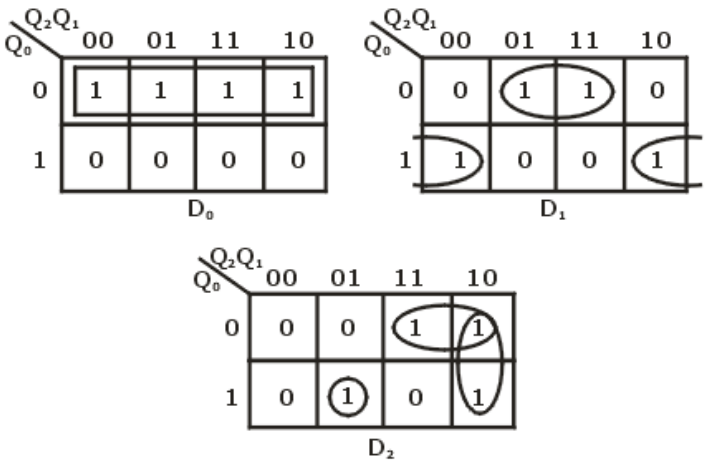


Figure : K-Maps for 8-bit Synchronous counter

The K-maps for D_0, D_1, and D_2 are given as,



The minimised expressions for D_0, D_1 and D_2 are:

$$D_0 = \bar{Q}_0$$

$$D_1 = Q_1\bar{Q}_0 + \bar{Q}_1Q_0$$

$$D_2 = Q_2\bar{Q}_0 + Q_2\bar{Q}_1 + Q_2Q_1Q_0$$

$$= Q_2(\bar{Q}_0 + \bar{Q}_1) + \bar{Q}_2Q_1Q_0$$

$$= Q_2(\overline{Q_0.Q_1}) + \bar{Q}_2(Q_1Q_0) = Q_2 \oplus Q_1.Q_0$$

The complete circuit of the synchronous counter using positive edge triggered D FLIP-FLOPs is shown in figure below as

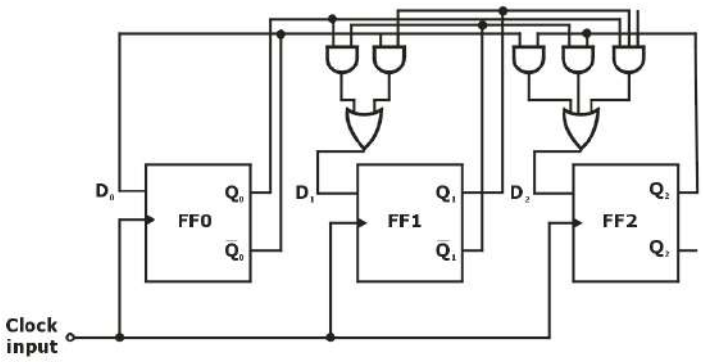


Figure: 8-bit Synchronous Counter Circuit

SYNCHRONOUS SEQUENTIAL CIRCUIT MODELS

A general block diagram of clocked sequential circuit is also known as finite stable machine (FSM). Depending upon the external outputs, there are two types of models of sequential circuits.

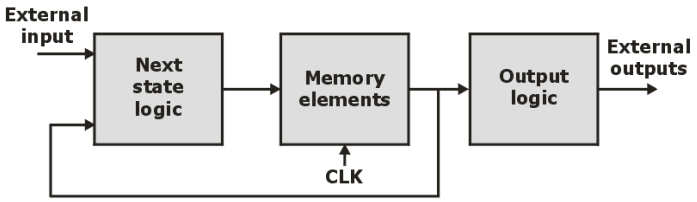
Mealy model:

In mealy model, the next of the function depends on present state as well as present inputs.

Moore Model:

In more model, the next state depends on the present state and present inputs but also on the outputs of more model.

The block diagram of a Moore model is given as



The systematic procedure for designing of clocked sequential circuit is based on the concept of 'state'. Hence the sequence of inputs, present & next states and output is represented by a state table or state diagram & if the procedure follows in the form of flow chart, it is known as algorithms state machine (ASM).

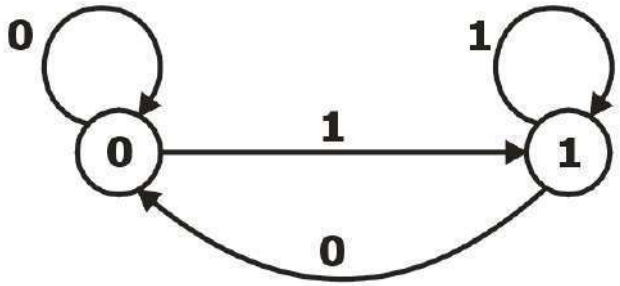
STATE DIAGRAM:

It is a directed graph, consisting of vertices (or nodes) and directed are between the nodes. Every state of the circuit is represented by a node in the graph. A node is represented by a circle with the name of the state written inside the circle. The directed are represents the state transitions.

With the circuit in may one state, at the occurrence of a clock pulse, there will be a state transition to the next state and there will be an output, corresponding to the requirement of the circuit. This state transition is represented by a directed line and we use each (/) for representing present state and the next state.

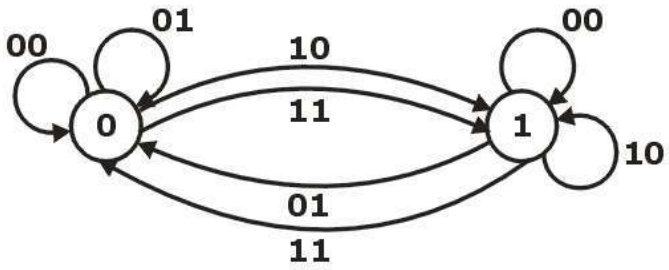
Example 4:
Draw the state diagram of D-flip-flop.

Solution:
The D flip-flop has only input (D) & two output states (Q = 0 & Q = 1).
Using the state table or characteristic table of the D-flip flop. The state diagram is given as



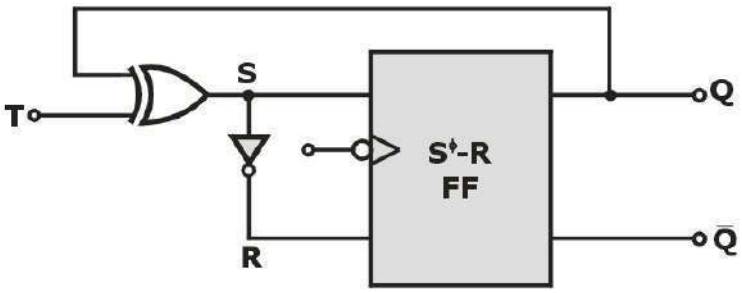
Example 5:
Draw the state diagram of a JK flip-flop.

Solution:
A JK flip-flop has inputs (J & K) and one clock input (CLK) and the two output states (Q = 0 & Q = 1). Using the state table or characteristic table of the JK flip-flop, the state diagram is given as

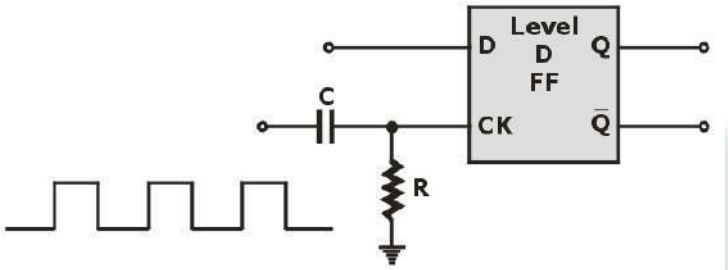


PRACTICE QUESTIONS

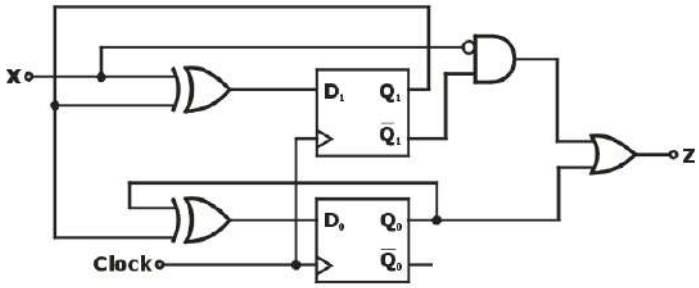
1. Prepare the truth table for the circuit shown in figure below and show that it acts as a T-type FLIP-FLOP.



2. In the circuit shown in figure below, the time constant (RC) is very small. Explain the operation of this circuit.



3. For a clocked sequential circuit shown in figure below, Obtain
   1. Excitation and output equations,
   2. The output sequence for an input sequence of 101001 assuming initial state to be $Q_1Q_0 = 00$.



4. A clocked synchronous sequential circuit using positive-edge-triggered D FFs has an input and output Y.
   The excitation equations are:

$$D_1 = Q_1 \cdot \bar{X} + \bar{Q}_1 \cdot X + Q_1 \cdot \bar{Q}_0 \cdot X$$

$$D_0 = Q_0 \cdot \bar{X} + \bar{Q}_0 \cdot X$$

   And the output equation is
   $Y = Q_1 \,.\, Q_0 \,.\, X$
   A. Draw its circuit diagram,
   B. Obtain its state diagram,
   C. Redesign the circuit using J-K FF's.

5. For the stage diagram shown in figure below, obtain the state table and design the circuit using minimum number of J-K FF's.