UNIT-I : Fundamentals of Digital Electronics

1. 1. Introduction

Digital electronics is a type of electronics that deals with the digital systems which processes the data/information in the form of binary(0s and 1s) numbers, whereas analog electronics deals with the analog systems which processes the data/information in the form of continuous signals.

Continuous signals
A Continuous signal is function f(t), whose value is defined for all time 't'.
in other words
Continuous signal a varying quantity with respect to independent variable time.
Example: Figure 1.1(a) shows the continuous signal.

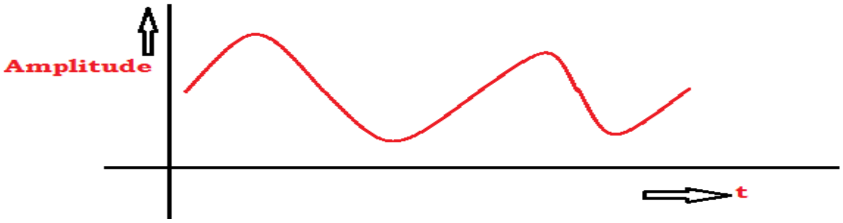


Figure 1.1(a): Continuous signals.

Digital signals
A digital signal is a quantized discrete time signals.
Example: Figure 1.1(b) shows the discrete and digital signals.

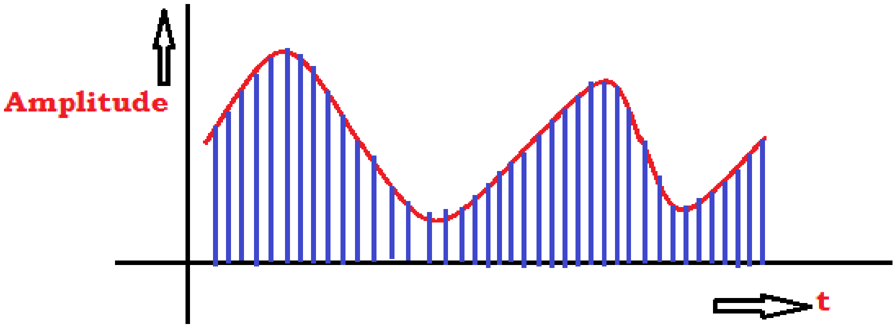


Figure 1.1(b1): Discrete signal.

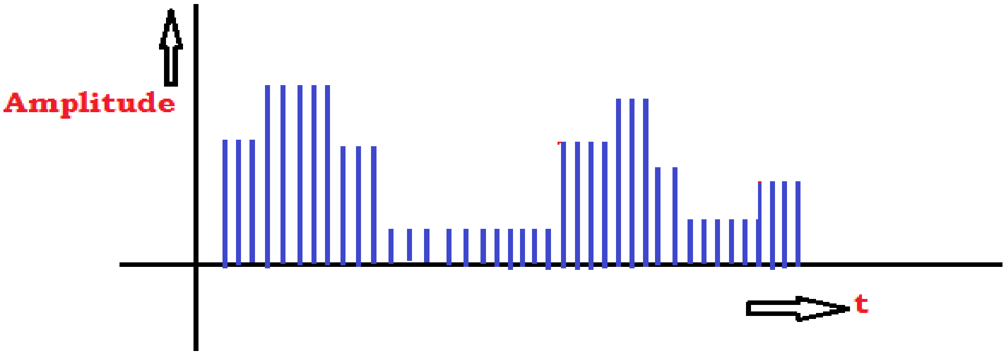


Figure 1.1(b2): Digital signal.

1.2. Boolean Algebra

Boolean algebra is a branch of Algebra (Mathematics) that deals with operations on logical values with Boolean variables, Boolean variables are represented as binary numbers which takes logic 1 and logic 0 values. Hence, the Boolean algebra is also called two-valued logic, Binary Algebra or Logical Algebra. The Boolean algebra was introduced by great mathematician George Boole in 1847. The Boolean algebra is a fundamental for the development of digital electronic systems, and is provided for in all programming languages. Set theory and statistics fields also uses Boolean algebra for the representation, simplification and analysis of mathematical quantities.

Logic levels are classified into two types

1. Positive logic

Logic 0 = False, 0V, Open Switch, OFF
Logic 1= True, +5V, Closed Switch, ON

2. Negative logic

Logic 0 = True, +5V, Closed Switch, ON
Logic 1= False, 0V, Open Switch, OFF

Boolean algebra differs from normal or elementary algebra. Latter deals with numerical operations such as, addition, subtraction, multiplication and division on decimal numbers. And former deals with the logical operations such as conjunction (OR), disjunction(AND) and negation(NOT).

In present context, positive logic has been used for the entire discussion, representation and simplification of Boolean variables.

1.2.1. Rules and properties of Boolean Algebra

1. Boolean variables takes only two values, logic 1 and logic 0, called binary numbers.
2. Basic operations of Boolean algebra are complement of a variable, ORing and ANDing of two or more variables.
3. Mathematical description of Boolean operations using variables is called Boolean expression.
4. Complement of variable is represented by an over-bar (-).
   Example: $Y = \bar{A}$, Y is the output variable
5. ORing of variables is represented by a plus symbol (+)
   Example:$Y = A + B$, Y is the output variable
6. ANDing of variables is represented by a dot symbol (.)
   Example:$Y = A.B$, Y is the output variable
7. Boolean operations are different from binary operations.
   Example : 1+1=10 in Binary Addition
   1+1=1 in Boolean algebra.

Table 1.1, shows the complement operation of a variable, table 1.2 summarized the OR operation and table 1.3, summarized the AND operation of two variables.

A	$\boldsymbol{Y = \bar{A}}$
0	1
1	0

Table 1.1: Complement of variable A

A	B	Y=A+B
0	0	0
0	1	1
1	0	1
1	1	1

Table 1.2: OR operation on A and B

A	B	Y=A.B
0	0	0
0	1	0
1	0	0
1	1	1

Table 1.3: AND operation on A and B

The present chapter deals with the simplification of Boolean expressions and representation using sum of product form and product of sum forms.

1.2.2. Boolean Laws:

Law-1: Commutative law

The sequence of changing the variables does not effect on the result even after changing their sequence while performing OR/AND operations of Boolean expression.

$$\mathrm{i.e., A.B = B.A \text{ and } A + B = B + A}$$

Law-2: Associative law

The order of operations on variables is independent.

$$\mathrm{A.(B.C) = (A.B).C \text{ and } A + (B + C) = (A + B) + C}$$

Law-3: Distributive Laws

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

Law-4: AND Laws

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

Law-5: OR Laws

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

Law-6: Inversion/Complement/NOT Laws

$$\bar{0} = 1$$

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

Law-7: Absorption Law

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

Law-8: Demargon's Laws

De-Morgan's First Law

Statement: Sum of complement of two or more variables is equal to the product of the complement of their variables.

$$\text{i.e.,}\ \overline{\mathrm{A} + \mathrm{B} + \mathrm{C} + \cdots ..} = \bar{\mathrm{A}}.\bar{\mathrm{B}}.\overline{\mathrm{C} \ldots ..}$$

Proof:

consider three variables for the proof shown in figure 1.4

A	**B**	**C**	$\boldsymbol{\overline{A + B + C}}$	$\boldsymbol{\bar{A}.\bar{B}.\bar{C}}$
0	0	0	**1**	**1**
0	0	1	**0**	**0**
0	1	0	**0**	**0**
0	1	1	**0**	**0**
1	0	0	**0**	**0**
1	0	1	**0**	**0**
1	1	0	**0**	**0**
1	1	1	**0**	**0**

Table 1.4: De-Morgan's First Law

De-Morgan's Second Law

Statement: Product of complement of two or more variables is equal to the sum of the complement of their variables.

$$\text{i.e.,}\ \overline{\mathrm{ABC} + \cdots ..} = \cdots \bar{A} + \bar{B} + \bar{C} \ldots .$$

Proof:

consider three variables for the proof shown in figure 1.5

A	**B**	**C**	$\boldsymbol{\overline{A.B.C}}$	$\bar{\mathrm{A}} + \bar{\mathrm{B}} + \bar{\mathrm{C}}$
0	0	0	**1**	**1**
0	0	1	**1**	**1**
0	1	0	**1**	**1**
0	1	1	**1**	**1**
1	0	0	**1**	**1**
1	0	1	**1**	**1**
1	1	0	**1**	**1**
1	1	1	**0**	**0**

Table 1.5: De-Morgan's Second Law

Boolean expressions must be simplified and evaluated using the order of operator precedence shown in table 1.6

Operator	Precedence
Parenthesis	1
NOT	2
AND	3
OR	4

Table 1.6: Operator precedence

Example:

$$\boldsymbol{Y} = \underbrace{(\underbrace{\underbrace{\boldsymbol{A}\left(\underbrace{\overline{\boldsymbol{C}+\underbrace{\underbrace{\overline{\boldsymbol{B}}}\,\boldsymbol{D}}}}\right)}+\underbrace{\overline{\boldsymbol{B}\,\underbrace{\overline{\boldsymbol{C}}}}})}\,\overline{\boldsymbol{E}}}$$

1.2.3. Simplify the following expressions

1. $\mathbf{Y = BC + B\overline{C} + BA}$

solution:

$Y = BC + B\overline{C} + BA$

$Y = B(C + \overline{C}) + BA$

$Y = B + BA \qquad (\because C + \overline{C} = 1)$

$Y = B(1 + A) \qquad (\because 1 + A = 1)$

$Y = B.1$

$\mathbf{Y = B}$

2. $\mathbf{Y = A + \overline{A}B + \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}E}$

Solution:

$Y = A + \overline{A}B + \overline{A}\overline{B}C + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}\overline{C}\overline{D}E$

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

$Y = A + (B + \overline{B}C + \overline{B}\overline{C}D + \overline{B}\overline{C}\overline{D}E) \qquad (\because A + \overline{A}B = A + B)$

$Y = A + (B + \overline{B}(C + \overline{C}D + \overline{C}\overline{D}E)) \qquad (\because A + \overline{A}B = A + B)$

$Y = A + (B + (C + \overline{C}D + \overline{C}\overline{D}E)) \qquad (\because A + \overline{A}B = A + B)$

$Y = A + \left(B + \left(C + \overline{C}(D + \overline{D}E)\right)\right) \qquad (\because A + \overline{A}B = A + B)$

$Y = A + (B + (C + (D + \overline{D}E))) \qquad (\because A + \overline{A}B = A + B)$

$\mathbf{Y = A + B + C + D + E}$

3. $\mathbf{Y = C + \overline{BC}}$

Solution:

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

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

$Y = 1 + \overline{B}$

$\mathbf{Y = 1}$

4. $\mathbf{Y = \overline{AB}(\overline{A} + B)(\overline{B} + B)}$

Solution:

$Y = (\overline{A} + \overline{B})(\overline{A} + B)(1)$

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

$Y = 0 + \overline{A}(B + \overline{B}) + 0$

$Y = \overline{A}(1)$

$\mathbf{Y = \overline{A}}$

5. $\mathbf{Y = (A + C)(AD + A\overline{D}) + AC + C}$

Solution:

$\mathrm{Y = (A + C)(A(D + \overline{D})) + AC + C}$
$\mathrm{Y = (A + C)(A)(1) + AC + C}$
$\mathrm{Y = AA + AC + AC + C}$
$\mathrm{Y = A + AC + C}$
$\mathrm{Y = A(1 + C) + C}$
$\mathbf{Y = A + C}$

6. $\mathbf{Y = \overline{A}(A + B) + (B + AA)(A + \overline{B})}$
Solution:
$\mathrm{Y = \overline{A}A + \overline{A}B + BA + B\overline{B} + AAA + AA\overline{B}}$
$\mathrm{Y = 0 + \overline{A}B + AB + 0 + A + A\overline{B}}$
$\mathrm{Y = B(\overline{A} + A) + A + A\overline{B}}$
$\mathrm{Y = B + A(1 + \overline{B})}$
$\mathbf{Y = A + B}$

7. $\mathbf{Y = \overline{\overline{A + B\overline{C}} + D(\overline{E + \overline{F}})}}$
Solution:
$\mathrm{Y = \overline{(\overline{A + B\overline{C}})}\ \ (\overline{D(\overline{E + \overline{F}})})}$
$\mathrm{Y = (A + B\overline{C})(\ \overline{D} + \overline{\overline{(E + \overline{F}}})}$
$\mathbf{Y = (A + B\overline{C})(\ \overline{D} + E + \overline{F})}$

8. $\mathbf{Y = AB + A(B + C) + B(B + C)}$
Solution:
$\mathrm{Y = AB + AB + AC + BB + BC}$
$\mathrm{Y = AB + AC + B + BC}$
$\mathrm{Y = AB + AC + B(1 + C)}$
$\mathrm{Y = AB + AC + B}$
$\mathrm{Y = B(A + 1) + AC}$
$\mathbf{Y = B + AC}$

9. $\mathbf{Y = A\overline{B} + A(\overline{B + C}) + B(\overline{B + C})}$
Solution:
$\mathrm{Y = A\overline{B} + A(\overline{B}\overline{C}) + B\overline{B}\overline{C}}$
$\mathrm{Y = A\overline{B}(1 + \overline{C})}$
$\mathbf{Y = A\overline{B}}$

10. $\mathbf{Y = \overline{A}BC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}C + ABC}$
Solution:
$\mathrm{Y = \overline{A}BC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + A\overline{B}C + ABC}$
$\mathrm{Y = \overline{A}BC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + AC(\overline{B} + B)}$
$\mathrm{Y = \overline{A}BC + A\overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + AC}$
$\mathrm{Y = \overline{A}BC + \overline{B}\overline{C}(A + \overline{A}) + AC}$
$\mathrm{Y = \overline{A}BC + \overline{B}\overline{C} + AC}$
$\mathrm{Y = (\overline{A}B + A)C + \overline{B}\overline{C}}$
$\mathrm{Y = (A + B)C + \overline{B}\overline{C}}$
$\mathbf{Y = AC + BC + \overline{B}\overline{C}}$

11. If $\mathbf{F = \overline{A} + B}$**,** $\mathbf{A = Y + X}$ **and** $\mathbf{B = \overline{X} + Y}$**, then** $\mathbf{F} =?$
Solution:

$\mathrm{F} = (\overline{\mathrm{Y} + \mathrm{X}}) + (\overline{\mathrm{X}} + \mathrm{Y})$
$\mathrm{F} = \overline{\mathrm{Y}}.\overline{\mathrm{X}} + \overline{\mathrm{X}} + \mathrm{Y}$
$\mathbf{F} = \mathbf{Y} + \overline{\mathbf{X}}$

12. If $\mathbf{f}(\mathbf{A}, \mathbf{B}) = \mathbf{A} + \mathbf{B}$, then show that $\mathbf{f}(\mathbf{f}(\mathbf{X}, \mathbf{YZ}), \overline{\mathbf{X}}\mathbf{Y}) = \mathbf{X} + \mathbf{Y}$
Solution:
$\mathrm{f}(\mathrm{A}, \mathrm{B}) = \mathrm{f}(\mathrm{X}, \mathrm{YZ})$
$\therefore \mathrm{A} = \mathrm{X}$ and $\mathrm{B} = \mathrm{YZ}$
$\mathrm{f}(\mathrm{X}, \mathrm{YZ}) = \mathrm{X} + \mathrm{YZ}$
$\mathrm{f}(\mathrm{f}(\mathrm{X}, \mathrm{YZ}), \overline{\mathrm{X}}\mathrm{Y}) = \mathrm{f}(\mathrm{A} + \mathrm{B})$
$\mathrm{A} = \mathrm{f}(\mathrm{X}, \mathrm{YZ})$and $\mathrm{B} = \overline{\mathrm{X}}\mathrm{Y}$
$\mathrm{f}(\mathrm{f}(\mathrm{X}, \mathrm{YZ}), \overline{\mathrm{X}}\mathrm{Y}) = (\mathrm{X} + \mathrm{YZ}) + \overline{\mathrm{X}}\mathrm{Y}$
$\mathrm{f}(\mathrm{f}(\mathrm{X}, \mathrm{YZ}), \overline{\mathrm{X}}\mathrm{Y}) = (\mathrm{X} + \overline{\mathrm{X}}\mathrm{Y}) + \mathrm{YZ}$
$\mathrm{f}(\mathrm{f}(\mathrm{X}, \mathrm{YZ}), \overline{\mathrm{X}}\mathrm{Y}) = (\mathrm{X} + \mathrm{Y}) + \mathrm{YZ}$
$\mathrm{f}(\mathrm{f}(\mathrm{X}, \mathrm{YZ}), \overline{\mathrm{X}}\mathrm{Y}) = \big(\mathrm{X} + \mathrm{Y}(1 + \mathrm{Z})\big)$
$\mathbf{f}(\mathbf{f}(\mathbf{X}, \mathbf{YZ}), \overline{\mathbf{X}}\mathbf{Y}) = (\mathbf{X} + \mathbf{Y})$

13. Let $\mathbf{A} * \mathbf{B} = \overline{\mathbf{A}} + \mathbf{B}$ and $\mathbf{C} = \mathbf{A} * \mathbf{B}$ then $\mathbf{C} * \mathbf{A}$ is ________
Solution:
$\mathrm{C} * \mathrm{A} = \overline{\mathrm{C}} + \mathrm{A}$
$\mathrm{C} * \mathrm{A} = (\overline{\mathrm{A} * \mathrm{B}}) + \mathrm{A}$
$\mathrm{C} * \mathrm{A} = \overline{(\overline{\mathrm{A}} + \mathrm{B})} + \mathrm{A}$
$\mathrm{C} * \mathrm{A} = \mathrm{A}.\overline{\mathrm{B}} + \mathrm{A}$
$\mathbf{C} * \mathbf{A} = \mathbf{A}$

1.3. Realization of Boolean expressions using logic gates.

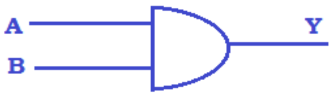
Logic gate is the basic building block of any digital circuits. The logic gates may have one or more inputs and only one output. The relationship between input and output is based on a certain logic, which is same as Boolean operations, such as AND, OR and NOT.

Based on the Boolean operations, the gates are named as AND gate, OR gate and NOT gate. These three gates are called basic gates, and some more gates can be derived by using the basic gates, they are named as NAND gate, NOR gate, EXOR gate and XNOR gate. NAND and NOR gates are called universal gates, because by using only the NAND gates /NOR gates we can realize all basic gates even all Boolean expression.

Logic gates, its truth table, expression and symbols are summarized in the table 1.7 as follows.

Sl. No. **Gate name and Logic Symbol** **Truth table and Logical Expression**

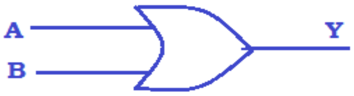
1 AND Gate



Inputs		**Output**
A	**B**	$\boldsymbol{Y = A.B}$
0	0	0
0	1	0
1	0	0
1	1	1

2 OR Gate

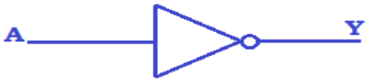
Inputs		**Output**
A	**B**	$\boldsymbol{Y = A + B}$



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

NOT Gate

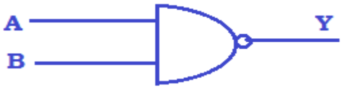
3



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

NAND Gate

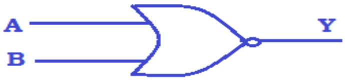
4



Inputs		**Output**
A	**B**	**Y**
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

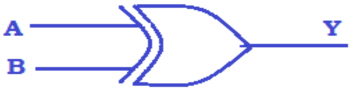
5



Inputs		**Output**
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	0

EX-OR Gate

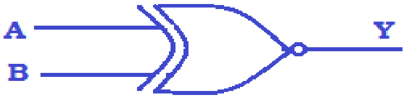
6



Inputs		**Output**
A	**B**	**Y**
0	0	0
0	1	1
1	0	1
1	1	0

EX-NOR Gate

7

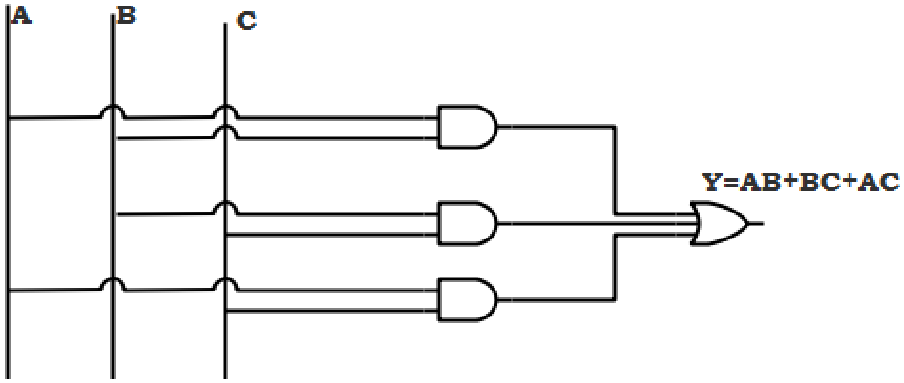


Inputs		**Output**
A	**B**	**Y**
0	0	1
0	1	0
1	0	0
1	1	1

1.3.1. Realize the following Boolean expression using basic gates.

Y = AB + BC + AC

Logic diagram

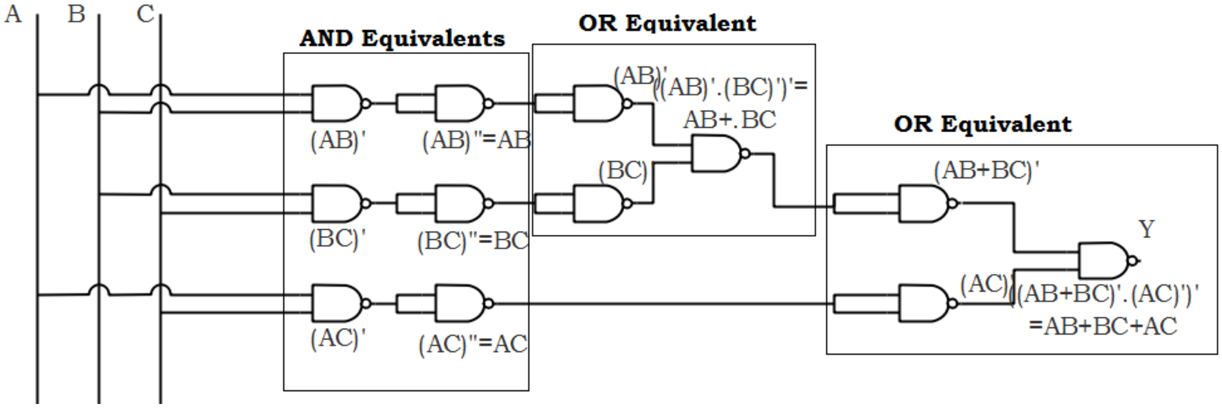


1.3.2. Realize the following Boolean expression using only NAND gates.

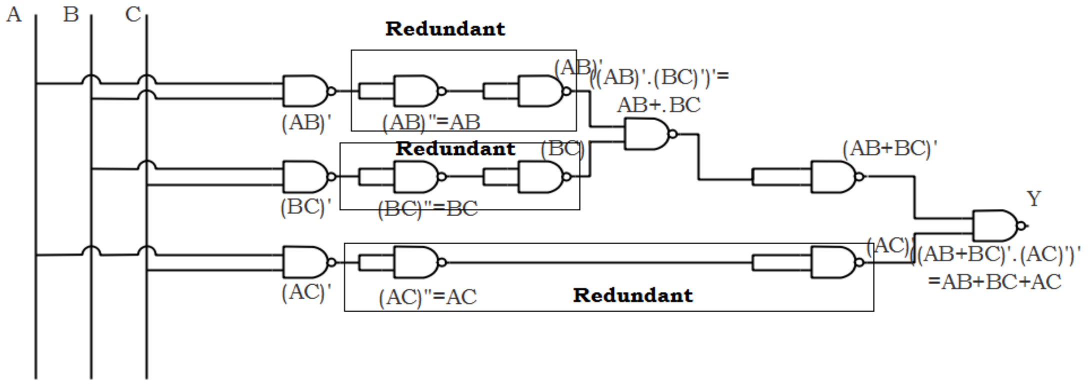
Y = AB + BC + AC

Logic diagram

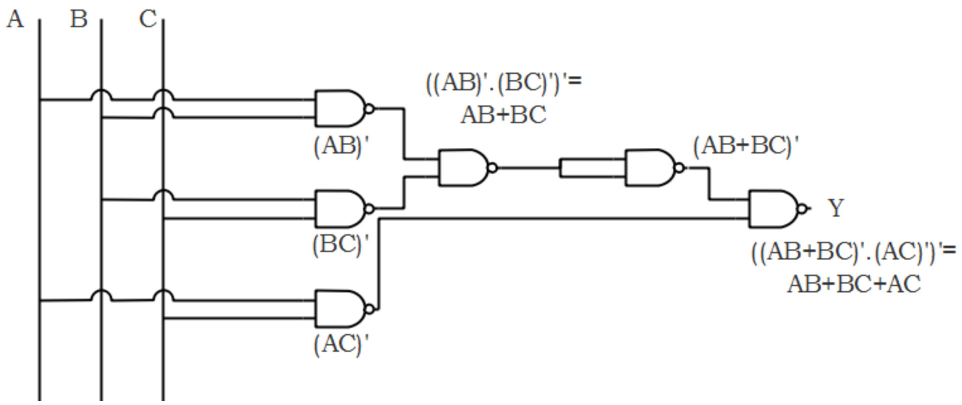
Step-1: Replace basic gates by NAND equivalents



Step-2: Eliminate two single input NAND gates are connected in series.



Step-3: Draw the resultant logic circuit.



1.4. Representation of Boolean Expressions

The relationship between Boolean variables and output variable is called Boolean expression, the Boolean expressions can be represented in two different forms, they are,
1. Sum of Products (SOP) form and
2. Product of Sums (POS) form

1.4.1. Sum of Product (SOP) form

The Boolean Expressions in which the product of input variables are summed together for output high.
Example: Consider a truth table shown in table 1.8.

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

Table 1.8: Truth table

The Boolean expression for Y is

$$\mathrm{Y} = \overline{\mathrm{A}}\overline{\mathrm{B}}\mathrm{C} + \overline{\mathrm{A}}\mathrm{B}\overline{\mathrm{C}} + \mathrm{A}\overline{\mathrm{B}}\overline{\mathrm{C}} + \mathrm{A}\overline{\mathrm{B}}\mathrm{C} + \mathrm{ABC} - - - (1)$$

NOTE: $\mathrm{A} = 0$ is represented as $\overline{\mathrm{A}}$ and $\mathrm{A} = 1$ will be represented as A

Expression (1) is a standard or canonical sum of product form, which is directly derived from the truth table.

Sum variable can be minimized using Boolean algebra rules.

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

$$Y = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B} + AC$$
$$Y = \overline{A}\overline{B}C + A\overline{B} + \overline{A}B\overline{C} + AC$$
$$Y = (\overline{A}C + A)\overline{B} + \overline{A}B\overline{C} + AC$$
$$Y = (C + A)\overline{B} + \overline{A}B\overline{C} + AC$$
$$Y = A\overline{B} + C\overline{B} + \overline{A}B\overline{C} + AC --- (2)$$

Expression (2) is the simplified form of canonical SOP form called, **minimal SOP form**.

NOTE:
1. Canonical SOP form to minimal SOP form and vice versa can also be derived using truth table.
2. Each product terms of SOP form is called **minterms**.
3. Canonical SOP form of Boolean expressions can also be written using decimal equivalent of input variables for the output high.

Example: for the Boolean expression (1), the output is high for ABC=001, ABC=010 ABC=100, ABC=101 and ABC=111.
The decimal equivalent of ABC=001 is '1', ABC=010 is '2', ABC=100 is '4', ABC=101 is '5' and ABC=111 is '7'
Therefore, Y can also be expressed as

$$Y(A, B, C) = (m_1, m_2, m_4, m_5, m_7)$$

OR

$$Y(A, B, C) = \sum m(1,2,4,5,7)$$

Problem: Refer the truth table shown in table 1.9., write the Boolean expression in canonical SOP form and minimal SOP form. Also write the different ways of writing canonical SOP form.

A	**B**	**C**	**Y**
0	0	0	0
0	**0**	**1**	**1**
0	**1**	**0**	**1**
0	1	1	0
1	0	0	0
1	**0**	**1**	**1**
1	**1**	**0**	**1**
1	**1**	**1**	**1**

Standard or canonical form representation

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

other representations

$$Y(A, B, C) = (m_1, m_2, m_5, m_6, m_7)$$

OR

$$Y(A, B, C) = \sum m((1,2,5,6,7)$$

Simplification of Boolean expression using Boolean algebra rules

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

$$Y = \overline{B}C + \overline{A}B\overline{C} + AB$$
$$Y = \overline{B}C + (\overline{A}\overline{C} + A)B$$
$$Y = \overline{B}C + (\overline{C} + A)B$$
$$Y = \overline{B}C + B\overline{C} + AB - \text{minimal SOP form}$$

1.4.2. Product of Sum (POS) form

The Boolean Expressions in which the Sum of input variables are multiplied together for output low.
Example: Consider a truth table shown in table 1.9.

A	B	C	Y
0	**0**	**0**	**0**
0	0	1	1
0	1	0	1
0	**1**	**1**	**0**
1	0	0	1
1	0	1	1
1	**1**	**0**	**0**
1	1	1	1

Table 1.9: Truth table

The Boolean expression for Y is

$$Y = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C) - - - (1)$$

NOTE: $A = 0$ is represented as A and $A = 1$ will be represented as $\overline{A}$

Expression (1) is a standard or canonical Products of sum form, which is directly derived from the truth table.

Sum variable can be minimized using Boolean algebra rules.

$$Y = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$$
$$Y = \big(A + (B + C)(\overline{B} + C)\big)(\overline{A} + \overline{B} + C) \qquad (\because (A + X)(A + Y) = A + XY) \text{ Distributive law}$$
$$Y = (A + C + B\overline{B})(\overline{A} + \overline{B} + C)$$
$$Y = (A + C)(\overline{A} + \overline{B} + C)$$
$$Y = (C + A)(\overline{A} + \overline{B})) - - - (2)$$

Expression (2) is the simplified form of canonical POS form called, **minimal POS form**.

NOTE:
1. Canonical POS form to minimal POS form and vice versa can also be derived using truth table.
2. Each Sum terms of POS form is called **maxterm**.
3. Canonical POS form of Boolean expressions can also be written using decimal equivalent of input variables for the output high.

Example: for the Boolean expression (1), the output is low for ABC=000, ABC=011 and ABC=110.
The decimal equivalent of ABC=000 is '0', ABC=011 is '3', and ABC=110 is '6'
Therefore, Y can also be expressed as

$$Y(A,B,C) = (M_0, M_3, M_6)$$

OR

$$Y(A,B,C) = \prod M(0,3,6)$$

Problem: Refer the truth table shown in table 1.9., write the Boolean expression in canonical POS form and minimal POS form. Also write the different ways of writing canonical SOP form.

A	B	C	Y
0	**0**	**0**	**0**
0	0	1	1
0	1	0	1
0	**1**	**1**	**0**
1	**0**	**0**	**0**
1	0	1	1
1	1	0	1
1	1	1	1

Standard or canonical form representation

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

other representations

$$Y(A,B,C) = (M_0, M_3, M_4)$$

OR

$$Y(A,B,C) = \prod M(0,3,4)$$

Simplification of Boolean expression using Boolean algebra rules

$$Y = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + B + C)$$
$$Y = (A + B + C)(\overline{A} + B + C)(A + \overline{B} + \overline{C})$$
$$Y = (A\overline{A} + B + C)(A + \overline{B} + \overline{C})$$
$$Y = (B + C)(A + \overline{B} + \overline{C}) - - - \text{Minimal POS form}$$

1.5. K-Map simplification (Karnaugh Map simplification)

The simplification of Boolean expressions using Boolean algebraic rules is not unique and most of the cases, the resultant expression is not in minimal form. In order to get the uniqueness and final minimal form, K-map technique will be used. In the following section, the introduction to K-maps, grouping of variables and simplification procedures are discussed with examples.

NOTE:
Number of cells in K-map = number of possible cases
No. of possible cases=2^N
N is the number of input variables.

Example: Number of input variables=2, then the number of cells in K-map is 4.

NOTE: K-maps can take wither POS form or SOP form, in SOP form 1's are need to be grouped and in POS form 0's are need to be grouped.

NOTE: Only adjacent cells will be considered for grouping, diagonal cells should not be grouped.

NOTE: grouping can be done using 2 variables, 4 variables, 8 variables, 16 variables etc.., highest priority for grouping maximum variables in the above denomination. Variables are 0's for POS form and 1's for SOP form.

Procedure:

1. Select the number of cells according to the number of input variables.
2. Identify whether the given problem is SOP or POS form, minterms for SOP form and maxterms for POS form.

   NOTE: In SOP form, fill the cells by 1's at corresponding minterms and otherwise fill with 0's.

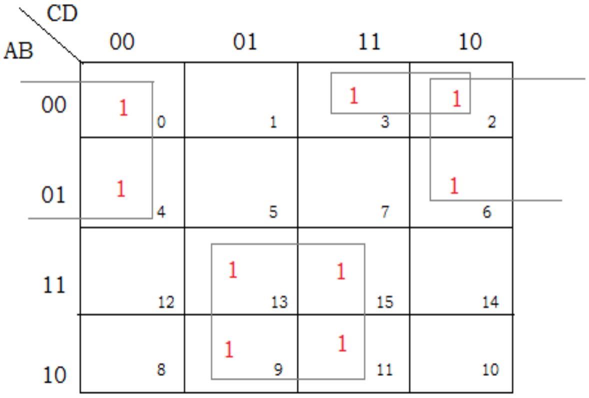
   NOTE: In POS form, fill the cells by 0's at corresponding maxterms and otherwise fill with 1's.

   NOTE: In POS form, take the complement of the output variable to get the resultant expression.
3. group the terms in the form of rectangular, the total number of terms is 2, 4, 8, etc.., try to cover as many elements as you can in one group.
4. from the groups, find the Product terms for SOP from and sum terms for POS form.

Example:

Simplify the following canonical SOP form of Boolean expression using K-map technique.

$$Y(A,B,C,D) = \sum m(0,2,3,4,6,9,11,13,15)$$



AB \ CD	00	01	11	10
00	1 0	1	1 3	1 2
01	1 4	5	7	1 6
11	12	1 13	1 15	14
10	8	1 9	1 11	10

Simplified Boolean Expression $Y = \overline{A}\overline{D} + \overline{A}\overline{B}C + AD$

******

UNIT-III: Flip-Flops and Registers

I. Introduction

Digital circuits are interconnection of various logic gates, which processes the data in the form of digital signals, which are designed to perform arithmetic as well as logical operations. Digital circuits are classified into two types

1. Combinational circuits and
2. Sequential circuits.

The following section is discussed the definition and differences between combinational and sequential circuits.

I.1. Combinational Circuits:

Combinational circuits are time independent circuits, which depends on the present input and do not dependent on previous inputs to generate any output. Figure (1) shows the block diagram of combinational circuit.

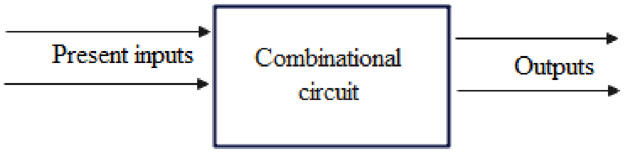


Figure 1: Block diagram of combinational circuit

Combinational circuit outputs are dependent only on present inputs.

$$i.e., Outputs = f(Present\ inputs) --- (1)$$

Features of combinational circuits:

- Simple in construction
- Speed is fast, only propagation delay occurs.
- No feedback
- Basic building blocks are AND gate, OR gate and NOT gate.
- Time independent

Examples: Adders, Subtractors, Decoders, Encoders, Multiplexers, Demultiplexers etc.

I.2. Sequential circuits:

Sequential circuits are those which are time dependent (Clock cycles) and depends on present as well previous outputs to generate any output. Figure (2) shows the block diagram of sequential circuit.

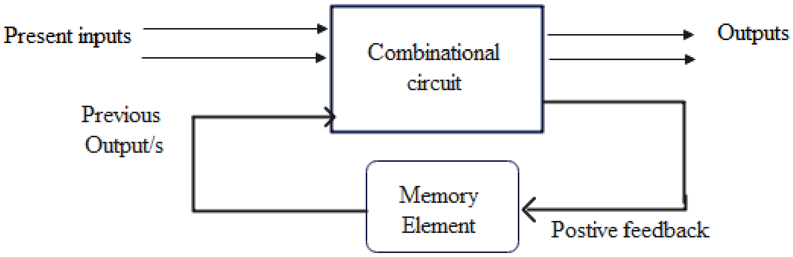


Figure 2: Block diagram of sequential circuit.

From the figure (2), the output, not only depends on present inputs but also previous outputs.

$$i.e., Output = f(Present\ inputs, Previous\ outputs) - - - (2)$$

The sequential circuits send the previous output to the input of the circuit with a positive feedback through memory element. Memory element is used to store the previous data and send it to the combinational circuit as the additional inputs.

Memory element plays a very important role in the design of sequential circuits. Flip-flop/Latch is a basic building block/ elementary building block which acts as a memory element for the design of any sequential circuits. The present section discusses the concept of Flip-flops.

Sequential circuits are classified into two types,

a. Synchronous sequential circuits – These circuits generate the output only at discrete time instants. These circuits require clock pulse (Master clock generator) to sample the inputs and determine the system behavior. These circuits are also called *clocked* sequential circuits.

b. Asynchronous sequential circuits – These circuits generate the output instantly, once the input variables change. These circuits are also called *unclocked* sequential circuits.

Features of Sequential circuits:

- Output depends on present as well as past data.
- Positive feedback is present to make dependent on past/previous output.
- Speed is slow.
- Complex design.
- Time dependent, hence need clock for triggering.
- Designed to storing the data.
- Basic element is flip-flop.

Example: Counters, Registers etc...

II. Concept of Flip-Flops

The elementary building block of any sequential circuits is Flip-flop, the flip-flop itself is a simple sequential circuit. Basically sequential circuits have feedback, and it is seen that feedback is present in flip-flops also.

Flip-flops have two stable conditions; each stable condition is called a state. The state represents the storage of binary symbols. The state of a system is also called as content or internal state or secondary state.

Flip-flop has two states; hence flip-flops are called Bi-stable devices/elements.

II.1. Basic Bistable element:

Figure (3) shows the basic bistable element, which is a cross-coupling of two NOT gates. The output of first NOT gate is connected to the input of second NOT gate and output of second NOT gate is connected to the input of the first NOT gate.

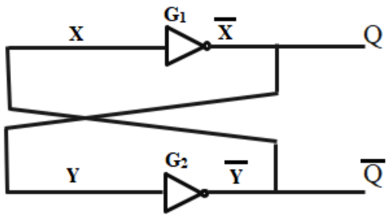


Figure 3: Basic Bistable element

In the figure (3), G1 and G2 are NOT gates, this bistable element can be analyzed with the following cases.

Case(i): Initially, assume X=0.

If X=0, output of the G1 is complement of X, i.e., $Q = \bar{X} = 1$. Q serves as input to the G2, i.e., $Y = Q = 1$, the output of the G2 is complement of Y, i.e., $\bar{Q} = \bar{Y} = X = 0$. However, the output of the G2 serving as input to G1, i.e., $\bar{Q} = X = 0$ and hence output of G1 complement of X, i.e., $Q = \bar{X} = Y = 1$. Thus the bistable element is stable with $Q = \bar{X} = Y = 1$ and $\bar{Q} = \bar{Y} = X = 0$

Case(ii): Assume Y=0.

If Y=0, output of the G2 is complement of Y, i.e., $\bar{Q} = \bar{Y} = 1$. Q serves as input to the G1, i.e., $X = \bar{Q} = 1$, the output of the G1 is complement of X, i.e., $Q = \bar{X} = Y = 0$. However, the output of the G1 serving as input to G2, i.e., $Q = Y = 1$ and output of G2 complement of Y, i.e., $\bar{Q} = \bar{Y} = X = 1$. Thus the bistable element is stable with $Q = \bar{X} = Y = 0$ and $\bar{Q} = \bar{Y} = X = 1$

The above discussion has proved that, the bistable element has two stable states, hence, the bistable element can be used to store binary symbols. When the output Q=1, indicates storing of binary symbol '1', and when the output Q=0, indicates, storing of binary symbol '0'. The binary symbol that is stored in the bistable element is called content or state. The state of the bistable element is given by the signal value at the Q output terminal. Hence the Q output terminal is called normal output, while $\bar{Q}$ output is termed as complementary output. When the device is storing a '1', it is said to be set or preset. On the other hand, when the device is storing a '0' is said to be reset or clear.

The bistable element may enter into another equilibrium condition called metastable state, which occurs when the two outputs are about halfway between those associated with logi-0 and logic-1, this is invalid state. The bistable element itself quickly move from metastable state to stable state when internal circuit signal values change say, due to circuit noise. But the amount of time the bistable element stay in its metastable is unpredictable. Hence, need to avoid the element entering into the metastable state.

Disadvantages of Bistable element

1. This bistable element has no inputs, so, difficult to fed the data.
2. When the power is applied, it becomes stable in one of its two stable states and remains in this state until power is removed.
3. No provision to force the device into a particular state.

To overcome these disadvantages, other class of flip-flops need to be designed, one such class of flip-flop is a latch, which is discussed in the next section.

II.2. Latches:

Latches are one class of flip-flops; the timing of the output changes is not controlled in latches. i.e., the output responds immediately to change in the input lines. Hence input lines are continuously need to be interrogated. In this section SR latch using NOR realization and NAND realization are discussed.

1. SR Latch:

In flip-flops, storing of '1' is called Set and storing of '0' is called Reset, hence the name SR latch. i.e., Set and Reset Latch. Figure (4) Shows the NOR gate realization of SR latch.

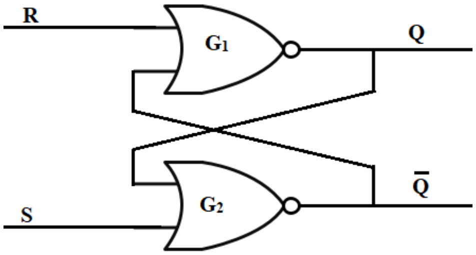


Figure 4: NOR gate realization of SR latch.

In the Figure (4), S and R are Set and Reset inputs respectively, G1 and G2 are NOR gates, Q and $\bar{Q}$ are output lines.

The circuit can be analyzed with the following cases.

Case (i): If S=0, and R=1.

As we know that, output of NOR gate is always '0' if any one input is '1', So if R=1, output of G1 is '0' irrespective of other input i.e., $Q = 0$ and output of G1 is one of the input for G2. Now, G2 takes the values S=0 and $Q = 0$, therefore, $\bar{Q} = 1$. This operation is called reset.

With the same reset state, let us consider S=0 and R=0.

R=0 and previous output $\bar{Q} = 1$ are the inputs for G1 and the output of G1 is 0, i.e., $Q = 0$ similarly, S=0 and Q=0 are the input for G2 and the output of G2 is 1, i.e., $\bar{Q} = 1$. From the discussion, it has been observed that, for the values of R=S=0, the SR latch retains the previous data. This operation is memory (storing of binary symbol '0').

Case (ii): If S=1, and R=0.

If S=1, output of G2 is '0' irrespective of other input i.e., $\bar{Q} = 0$ and output of G2 is one of the input for G1. Now, G1 takes the values S=0 and $\bar{Q} = 0$, therefore, $Q = 1$. This operation is called Set.

With the same set state, let us consider S=0 and R=0.

S=0 and previous output $Q = 1$ are the inputs for G2 and the output of G2 is 0, i.e., $\bar{Q} = 0$ similarly, S=0 and $\bar{Q}$=0 are the input for G1 and the output of G1 is 1, i.e., $Q = 1$. From the discussion, it has been observed that, for the values of R=S=0, the SR latch retains the previous data. This operation is memory (storing of binary symbol '1').

Case (iii): if S=1, and R=1.

If S=1, irrespective of the other input, the output of G2 is ‘0’, i.e., $\bar{Q} = 0$, G1 takes the inputs R=1 and $\bar{Q} = 0$, so the output of G1 is ‘0’, i.e., Q=0. This is invalid /Not used / forbidden case of SR latch.

Figure (5) shows the two different logic symbols of SR latch.

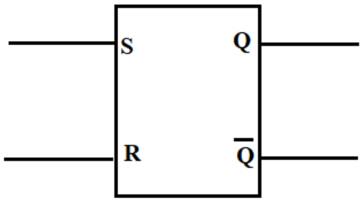


Figure 5: Logic Symbol of SR latch

Table (1) shows the truth table of SR latch, which summarizes the operations of SR latch in two different methods. Table (2) shows the characteristic table of SR latch and Table (3) shows the excitation table.

Characteristic table: Tabular form of generation of output from present input and previous output is called characteristic table.

Excitation table: Tabular form of finding inputs to change present state to required next state is called excitation table. Present state is denoted as Q_n and next state is denoted as Q_{n+1}.

Truth Table-I

Inputs		**Outputs**	
S	**R**	**Q**	$\bar{Q}$
0	0	Memory	
0	1	0	1
1	0	1	0
1	1	Forbidden (NOT USED)	

Table 1: Truth table of SR Latch

Truth Table-II

Inputs		**Output**
S	**R**	Q_{n+1}
0	0	Q_n
0	1	0
1	0	1
1	1	Forbidden (NOT USED)

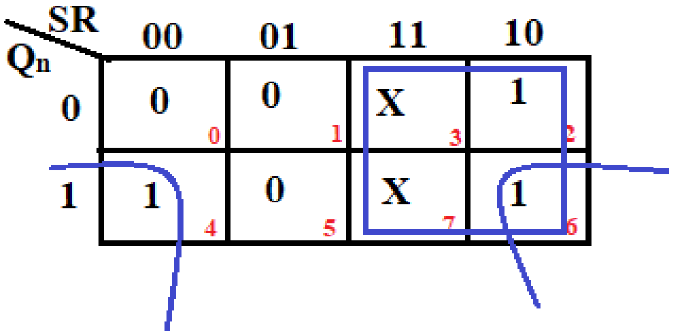
Table 2: Function table of SR Latch

Characteristic table:

Inputs			**Output**
Q_n	**S**	**R**	Q_{n+1}
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Table 3: Characteristic table of SR Latch

K-map simplification



$$Q_{n+1} = Q_n\bar{R} + S - - - (3)$$

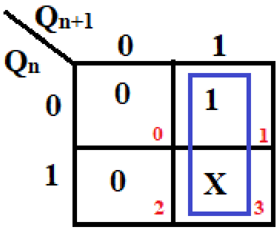
Equation (3) is called **characteristic equations**, which is the Boolean equation or algebraic description of the next state of a flip-flop in terms of present inputs and previous outputs.

Excitation table:

Inputs		**Outputs**	
Q_n	**Q_{n+1}**	**S**	***R***
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

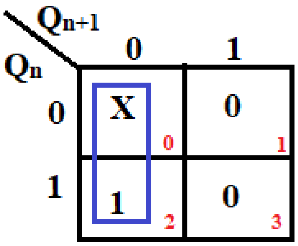
Table 4: Excitation table of SR Latch

K- map simplification for S



$$S = Q_{n+1} - - - (4)$$

K- map simplification for R



$$R = \bar{Q}_{n+1} - - - (5)$$

Equations (4) and (5) represents the Boolean equation or algebraic description of the excitations (input values) in terms in terms of the desired state.

2. $\bar{S}\bar{R}$ Latch

Figure (6) Shows the NAND gate realization of SR latch.

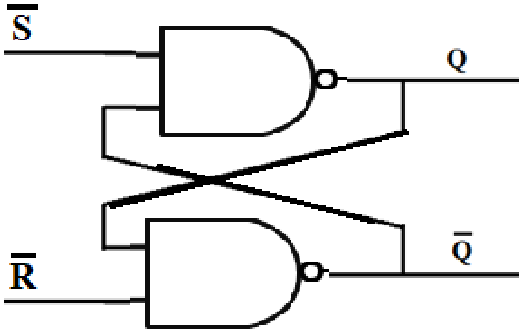


Figure 6: NAND gate realization of $\bar{S}\bar{R}$ latch.

In the Figure (6), S and R are Set and Reset inputs respectively, G1 and G2 are NAND gates, Q and $\bar{Q}$ are output lines.

The circuit can be analyzed with the following cases.

Case (i): If $\bar{S} = 0$*, and* $\bar{R} = 1$.

As we know that, output of NAND gate is always '1' if any one input is '0', So if $\bar{S} = 0$, output of G1 is '1' irrespective of other input i.e., $Q = 1$ and output of G1 is one of the input for G2. Now, G2 takes the values $\bar{R} = 1$. and $Q = 1$, therefore, $\bar{Q} = 0$. This operation is called Set.

With the same reset state, let us consider $\bar{S} = 1$ and $\bar{R} = 1$.

$\bar{S} = 1$ and previous output $\bar{Q} = 0$ are the inputs for G1 and the output of G1 is 1, i.e., $Q = 1$ similarly, $\bar{R} = 1$ and Q=1 are the input for G2 and the output of G2 is 1, i.e., $\bar{Q} = 0$. From the discussion, it has been observed that, for the values of $\bar{R} = \bar{S} = 1$, the $\bar{S}\bar{R}$ latch retains the previous data. This operation is memory (storing of binary symbol '1').

Case (ii): If $\bar{S} = 1$ *and* $\bar{R} = 0$

If $\bar{R} = 0$, output of G2 is '1' irrespective of other input i.e., $\bar{Q} = 1$ and output of G2 is one of the input for G1. Now, G1 takes the values $\bar{S} = 1$ and $\bar{Q} = 1$, therefore, $Q = 0$. This operation is called Reset.

With the same set state, let us consider $\bar{S} = 1$ and $\bar{R} = 1$.

$\bar{S} = 1$and previous output $\bar{Q} = 1$ are the inputs for G1 and the output of G1 is 0, i.e., $Q = 0$ similarly, $\bar{R} = 1$ and Q =0 are the input for G2 and the output of G2 is 1, i.e., $\bar{Q} = 1$. From the discussion, it has been observed that, for the values of $\bar{S} = 1$ and $\bar{R} = 1$, the SR latch retains the previous data. This operation is memory (storing of binary symbol '0').

Case (iii): If $\bar{S} = 0$ *and* $\bar{R} = 0$

If $\bar{S} = 0$ irrespective of the other input, the output of G1 is ‘1’, i.e.,$Q = 1$, G1 takes the inputs $\bar{R} = 0$ and $Q = 1$, so the output of G2 is ‘1’, i.e., $\bar{Q} = 1$. This is invalid /Not used / forbidden case of SR latch.

Figure (7) shows the logic symbols of $\bar{S}\bar{R}$ latch.

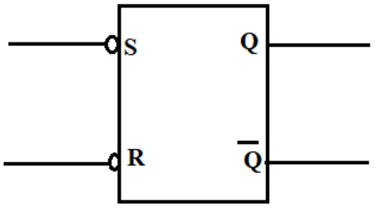


Figure 7: Logic Symbol of $\bar{S}\bar{R}$ latch

Table (5) shows the truth table of $\bar{S}\bar{R}$ latch, which summarizes the operations of $\bar{S}\bar{R}$ latch in two different methods. Table (6) shows the characteristic table of $\bar{S}\bar{R}$ latch and Table (7) shows the excitation table.

Characteristic table: Tabular form of generation of output from present input and previous output is called characteristic table.

Excitation table: Tabular form of finding inputs to change present state to required next state is called excitation table. Present state is denoted as Q_n and next state is denoted as Q_{n+1}.

Truth Table-I

Inputs		**Outputs**	
$\bar{S}$	**$\bar{R}$**	**Q**	**$\bar{Q}$**
0	0	Forbidden (NOT USED)	
0	1	1	0
1	0	0	1
1	1	Memory	

Table 5: Truth table of $\bar{S}\bar{R}$ Latch

Truth Table-II

Inputs		Output
$\bar{S}$	**$\bar{R}$**	**Q_{n+1}**
0	0	Forbidden (NOT USED)
0	1	1
1	0	0
1	1	Q_n

Table 5: Function table of $\bar{S}\bar{R}$ Latch

Characteristic table:

Inputs			**Output**
Q_n	**$\bar{S}$**	**$\bar{R}$**	**Q_{n+1}**
0	0	0	X
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	X
1	0	1	1

1	1	0	0
1	1	1	1

Table 6: Characteristic table of $\bar{S}\bar{R}$ Latch

Excitation table:

Inputs		**Outputs**	
Q_n	**Q_{n+1}**	$\overline{\boldsymbol{S}}$	$\overline{\boldsymbol{R}}$
0	0	1	X
0	1	0	1
1	0	1	0
1	1	X	1

Table 7: Excitaion table of $\bar{S}\bar{R}$ Latch

3. Gated SR Latch

In normal SR and $\bar{S}\bar{R}$ latches outputs will change immediately just after the change in input values. It is frequently desirable to avoid change in input values from affecting the state of the latch immediately. In order to allow the input changes to be effective only during a prescribed period of time, an enable signal is introduced. SR latch with enable is called gated SR latch or clocked latch or controlled latch.

In the following section gated $\bar{S}\bar{R}$ is discussed. An enable input can be introduced by two additional NAND gates with a control input 'C', shown in figure (7).

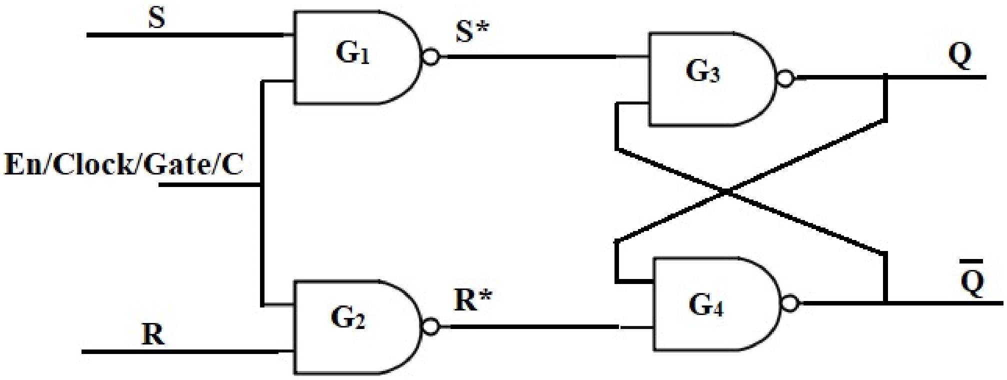


Figure 8: Gated SR latch

In the figure (8), G1 and G2 are two additional NAND gates connected to introduce control input/enable input. Cross coupling of G3 and G4 forms a $\bar{S}\bar{R}$. The inputs to the $\bar{S}\bar{R}$ latch are denoted as S* and R*, and these values depends on SR and enable input signal.

The gated SR latch arrangement shown in figure (7) can be analyzed with the following cases as follows. Before that, make a note of the expressions for S* and R*

$$S^* = \overline{S.En} => \bar{S} + \overline{En} --- (6)$$
$$R^* = \overline{R.En} => \bar{R} + \overline{En} --- (7)$$

Case (i): S=X, R=X, and En=0.

If $En = 0$, irrespective of the S and R inputs, $S^* = R^* = 1$ and for these inputs, $\bar{S}\bar{R}$ latch outputs stay at the previous state called memory operation. This case is the evident for the gated SR latch control input makes insensitive to the change in input values.

Case (ii); S=0, R=0, and En=1.

If S=R=0 and En=1, S*=1 and R*=1, Memory operation.

Case (iii): S=0, R=1 and En=1

If S=0, R=1 and En=1, S*=1 and R*=0, Reset operation (storing binary symbol ‘0’).

Case (iv): S=1, R=0 and En=1

If S=1, R=0 and En=1, S*=0 and R*=1, Set operation (storing binary symbol ‘1’).

Case (v): S=1, R=1 and En=1

If S=1, R=1 and En=1, S*=1 and R*=1, Invalid state (NOT used).

Figure (9) shows the two different logic symbols of gated SR latch.

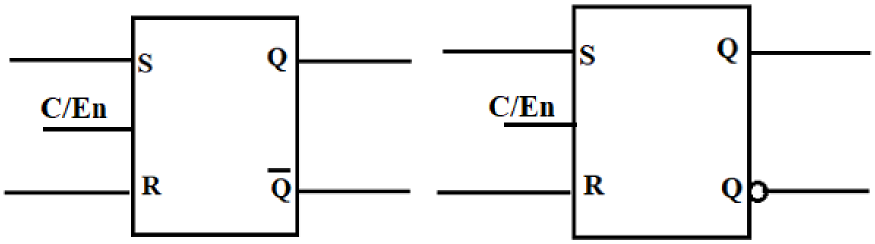


Figure 9: Logic Symbols of gated SR latch

Table (8 & 9) shows the truth table/function table of gated SR which summarizes the operations gated SR latch in two different methods. Table (10) shows the characteristic table of gated SR latch and Table (11) shows the excitation table.

Characteristic table: Tabular form of generation of output from present input and previous output is called characteristic table.

Excitation table: Tabular form of finding inputs to change present state to required next state is called excitation table. Present state is denoted as Q_n and next state is denoted as Q_{n+1}.

Truth Table-I

Inputs			**Outputs**	
En	***S***	***R***	**Q**	$\bar{Q}$
0	**X**	X	Memory	
1	0	0	Memory	

Truth Table-II

Inputs		**Output**
S	***R***	Q_{n+1}
0	0	Q_n
0	1	0

1	0	1	0	1
1	1	0	1	0
1	1	1	Forbidden (NOT USED)	

Table 8: Truth Table of Gated SR latch

1	0	1
1	1	Forbidden (NOT USED)

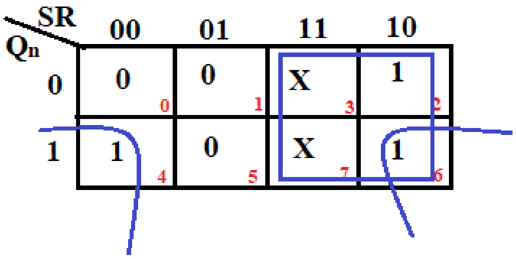
Table 9: Function Table of Gated SR latch

Characteristic table:

Inputs			**Output**
Q_n	***S***	***R***	**Q_{n+1}**
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	X
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	X

Table 10: Characteristic Table of Gated SR latch

K-map simplification



Q_n \ SR	00	01	11	10
0	0 [0]	0 [1]	X [3]	1 [2]
1	1 [4]	0 [5]	X [7]	1 [6]

$$Q_{n+1} = Q_n\bar{R} + S - - - (8)$$

Equation (8) is called **characteristic equations**, which is the Boolean equation or algebraic description of the next state of a flip-flop in terms of present inputs and previous outputs.

Excitation table:

Inputs		**Outputs**	
Q_n	**Q_{n+1}**	***S***	***R***
0	0	0	X
0	1	1	0
1	0	0	1
1	1	X	0

Table 11: Excitation Table of Gated SR latch

4. Gated D Latch

All input combinations of the above latches are not recommended, in gated latch, only specific input combinations are considered. In particular, the non-control inputs S and R are simultaneously active. Hence introducing a NOT gate between S and R inputs to make a single input denoted as 'D' called gated 'Data' latch, shown in figure ().

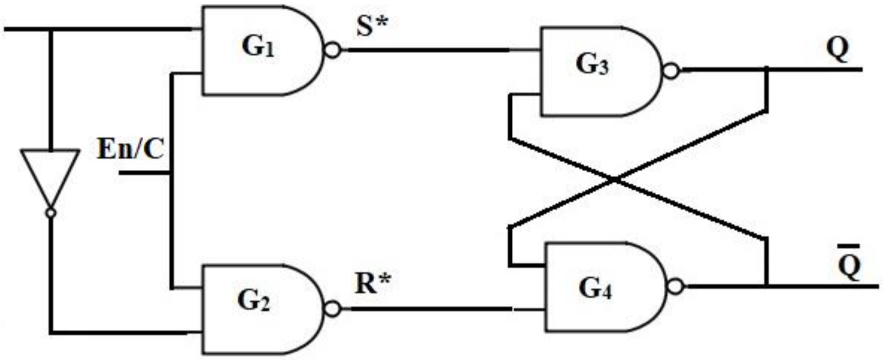


Figure 10 : Gated D latch

The opearation of the gated D latch is discussed as follows.

Case (i): D=0 and En=1

If D = 0 and En=1, the signals S =0 and R=1, S*=1 and R*=0, then Q=0 and $\bar{Q} = 1$ this is the combination for reset operation (Storing binary symbol '0')

Case (ii): D=1 and En=1

If D=1 and En=1, the signals S=1 and R=0, S*=0 and R*=1, then Q=1and $\bar{Q} = 0$ this is the combination for set operation (Storing binary symbol '1')

Case (iii): D=X and En=0

If D=X and En=0, the signals S*=1 and R*=1 irrespective of D, this is the combination for retaining the data called memory operation.

Figure (8) shows the two different logic symbols of gated SR latch.

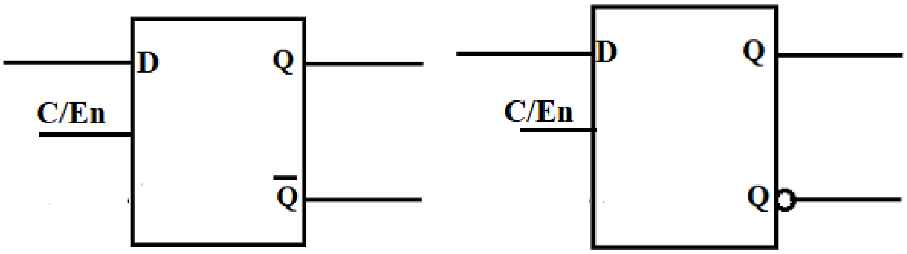


Figure 11: Logic Symbols of gated D latch

Table (12 & 13) shows the truth table of gated D which summarizes the operations gated D latch in two different methods. Table (14) shows the characteristic table of gated D latch and Table (15) shows the excitation table.

Truth Table-I

Inputs		Outputs	
En	***D***	**Q**	$\bar{Q}$
0	**X**	Memory	
1	0	0	1
1	1	1	0

Table 12: Truth Table of Gated D latch

Truth Table-II

Input	Output
D	**Q_{n+1}**
0	0
0	1

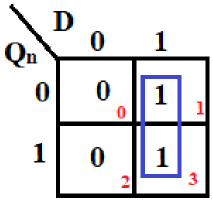
Table 13: Function Table of Gated D latch

Characteristic table:

Inputs		Output
Q_n	**D**	**Q_{n+1}**
0	0	0
0	1	1
1	0	0
1	1	1

Table 14: Characteristic Table of Gated D latch

K-map simplification



$$Q_{n+1} = D - - - (9)$$

Equation (9) is called **characteristic equations**, which is the Boolean equation or algebraic description of the next state of a flip-flop in terms of present inputs and previous outputs.

Excitation table:

Inputs		Outputs
Q_n	**Q_{n+1}**	***D***
0	0	0
0	1	1
1	0	0
1	1	1

Table 15: Excitation Table of Gated D latch

Difference between Latch and Flip-flop

Latch	Flip-Flop
The enable input is not a clock	The enable input is a clock
Sensitive to the level	Sensitive to the edge
Level triggered	Pulse and Edge triggered
Immediate output response within the propagation delays	Output change occurs in accordance with changes in a control line.

Table 16: Difference between Latch and Flip-Flop

Clock signal or clock pulse:

Clock pulse is a continuous, precisely spaced changes in amplitude (voltage) levels. It decides the time of the input applied to the system. Basically clock pulses are in the form of square wave with 50% duty cycle. Figure () shows the 50% duty cycle clock pulse, which is used to synchronize the operations of sequential digital circuits.

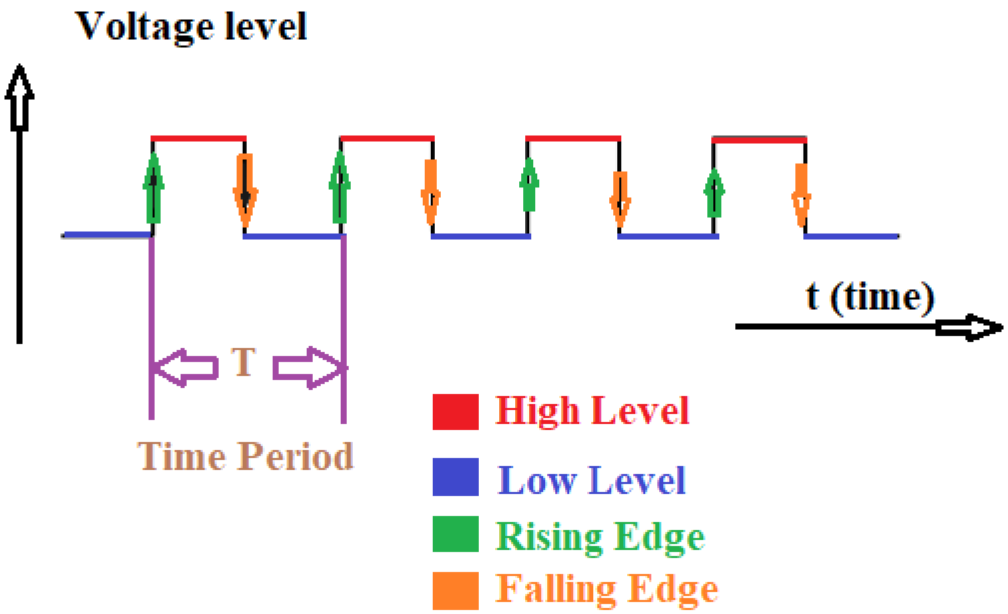


Figure 12: Clock pulse

Characteristic parameters of clock pulse is frequency and Time period. Frequency is the number of cycles output will perform in one second an Time period is the amount of time required to complete one cycle (Positive level as well as negative level). Time period is the reciprocal of frequency.

i.e., $T = \frac{1}{f} - - -$ (10) where, f is the frequency in Hz.

Duty cycle of clock pulse is the ratio of ON time to total time. In the figure () ON time and OFF time are mentioned for reference.

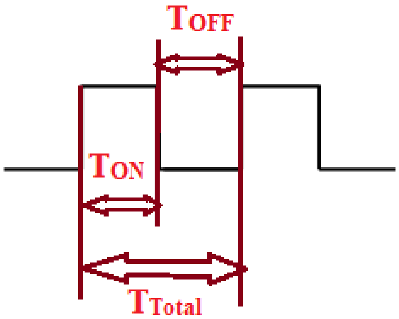


Figure 13: Duty cycle of clock pulse

$$T_{Total} = T_{ON} + T_{OFF} --- (11)$$

$$\%\ of\ Duty\ Cycle = \frac{T_{ON}}{T_{Total}} * 100 --- (12)$$

Figure (13) shows the ideal clock, which has no rise time and fall times, but for practice the rise time and fall time need to be considered. Figure () shows the practical clock pulse.

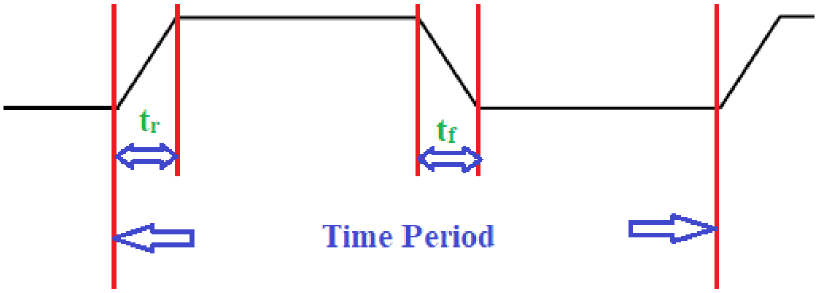


Figure 14: Practical clock with rise and fall times

Where, t_r is the rise time and t_f is the fall time.

Timing considerations:

The response of any system is non-instantaneous with the applied input, i.e., the systems will take certain time to respond for the applied input/s. Means appropriate delay occurs due to time delays associated with the gates themselves. To achieve desired responses, timing constraints must be satisfied.

1. Propagation delay:

The time taken to produce output with respect to change in an input is called propagation delay. The propagation delay between each pair of input and output terminals is different due to change occurs to move from low to high or vice-versa.

Propagation delay is illustrated in the figure (), the midpoints highlighted in the clock pulse are used to specify the time delays. Effect of setting and resetting operation of SR latch is taken as an example for the demonstration of propagation delay.

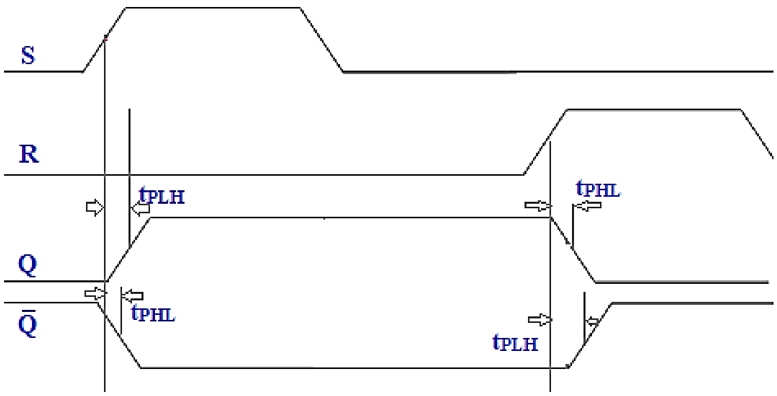


Figure 15: Illustration of Propagation delay

It should be note that, the outputs do not change instantaneously to an input change nor do the output change simultaneously. In the figure (15), propagation delays from low to high is denoted as t_{LPH} and high to low is denoted as t_{PHL}. In general, t_{LPH} and t_{PHL} are different at Q and $\bar{Q}$.

For further analysis, ideal clocks are considered for simplifying the analysis.

Figure (16) shows the timing diagram of SR latch. Timing diagram is a graph that depicts the input and output transitions of a flip-flop or a latch as a function of time.

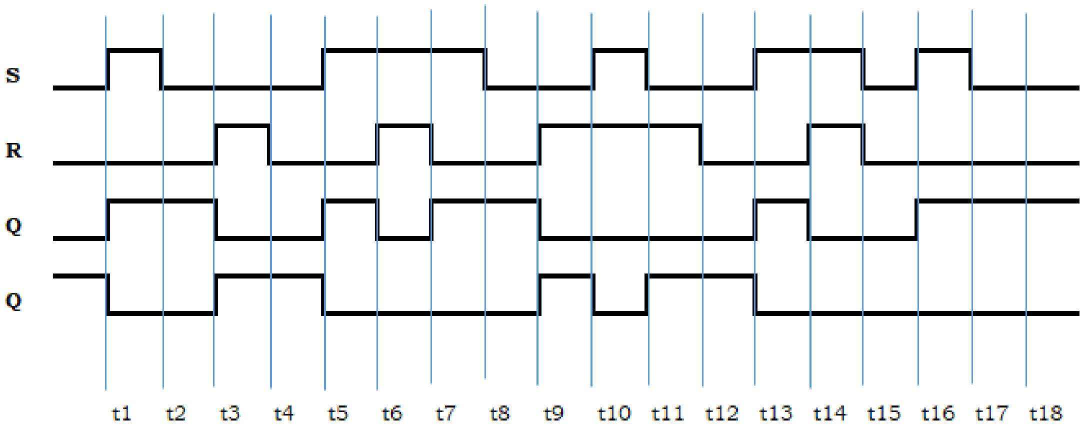


Figure 16: Timing diagram of SR latch.

From the figure (), it should be note that, when S=R=1, both Q and $\bar{Q}$ outputs become 0. In addition, special attention should be given to the response of the latch at time t_{15}. Here it is assumed that the signals on the S and R input terminals are simultaneously changed from 1 to 0. As a consequence, the response of the latch is unpredictable as indicated by the shaded area. The latch may be in its 0-state, or metastable. At time t_{16} , however the application of the 1 on the set input terminal returns the latch to predictable behavior after a short propagation delay time.

II.3. Flip-Flops

Latches have a property called transparency, means the output change occurs immediately when the input change occurs. In certain applications, this is an undesirable property. Hence it is

necessary to synchronize the change in output with control line. The device/element which has a property of synchronizing output change in accordance with the control line is called flip-flop.
Classification of Flip-flops

1. Pulse Triggered Flip-flops

The behavior of flip-flops dependent upon the rising and falling edges of the clock signal as well as the period of time in which the control signal is high.

Examples: MSSR and MSJK

2. Edge Triggered Flip-flops

The behavior of the flip-flop is dependent on either rising edge or positive edge of the control signal, is called edge triggered flip-flops.

Examples: Positive edge and negative edge triggered

In this section pulse triggered master slave SR flip- flops are discussed.

1. Pulse triggered Master slave SR flip-flop.
Master slave SR flip-flop can be constructed using two SR latches with an inverter shown in figure (17). The inputs S and R are used to set and reset the flip-flop. A clock signal 'Clk' is applied to control the input line.

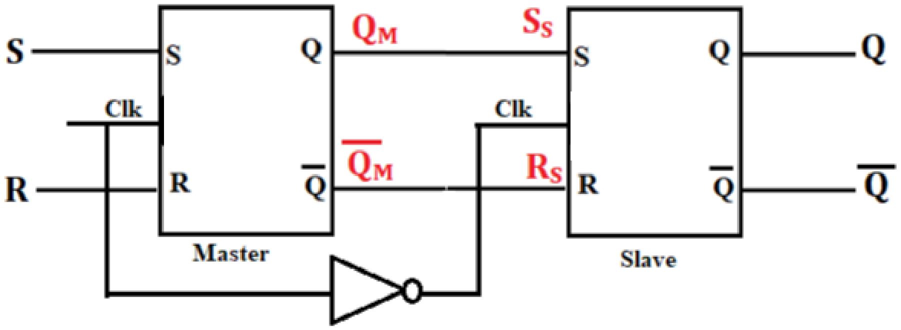


Figure 17: Pusle triggered Master Slave SR flip flop

The operation of the pulse triggered Master-Slave SR flip flop is explained as follows. The first SR latch shown in figure (17) is called Master latch and the second latch is called Slave latch. The slave latch is driven by master latch. Master latch gets enabled at rising edge of the clock pulse and active for entire ON period of the clock pulse. QM and QM' are the outputs of the master latch. The slave latch will enable at the falling edge of the clock pulse because of the NOT gate.
Figure shows the status of master and slave latches with practical clock pulse.

Master latch is active from time t2 to t3, at the same time slave is disabled. Slave latch is active before t1 and after t4, hence, if either master latch or slave latch is active at a time, not both simultaneously.

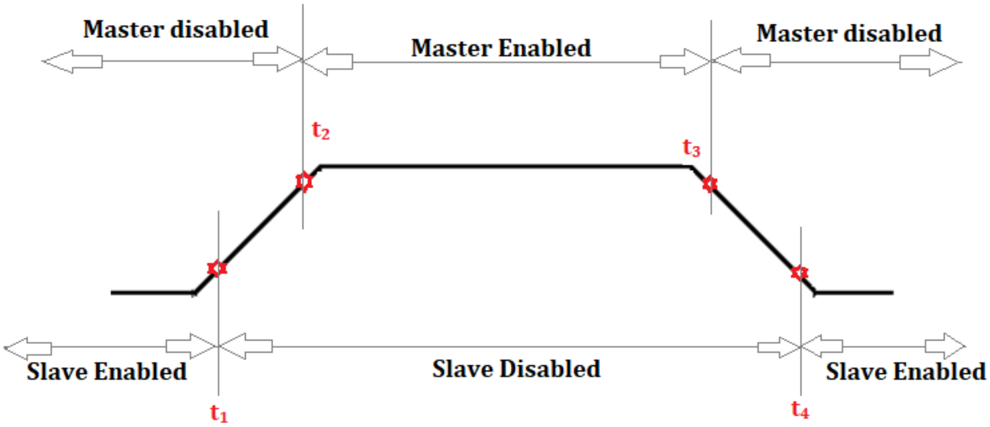


Figure 18: Illustration of enable and disable period of master and slave latches with practical clock.

Case (i): If clock =0, the master latch is disabled and any changes on S and R are neglected. At the same time slave latch is enabled because of NOT gate and slave output is same as that of the master latch, since outputs (QM and QM') of master latch serves as inputs (SS and RS) to the slave latch.
Case (ii) If clock signal rises to high, i.e., at the rising edge of the clock pulse, slave latch is disabled and master latch enabled at time t2. During the ON period (from t2 to t3), the master latch responds to the inputs on S and R lines, meanwhile slave latch is disabled and any change on the master latch are not reflected in slave latch, hence the output is same as the previous state.
Case (iii) If clock signal reduces to zero, i.e., at the falling edge of the clock pulse, the slave latch is enabled and master latch is disabled. The output of the slave latch is the same as the state of master latch as mentioned in case (i).
Table (17), summarized the functionality of master slave SR flip-flop.

Function table/Truth Table

Clock	**S**	**R**	$\boldsymbol{Q_{N+1}}$	$\overline{\boldsymbol{Q_{N+1}}}$
1	0	0	Q_N	$\overline{Q_N}$
1	0	1	0	1
1	1	0	1	0
1	1	1	X	X
0	X	X	Q_N	$\overline{Q_N}$

Table 17: Function Table of Master slave SR flip-flop

Timing Diagram

The analysis of flip-flops is easy with timing diagrams, the timing diagram for the random sequence of S and R inputs is shown in figure ().

Example-1

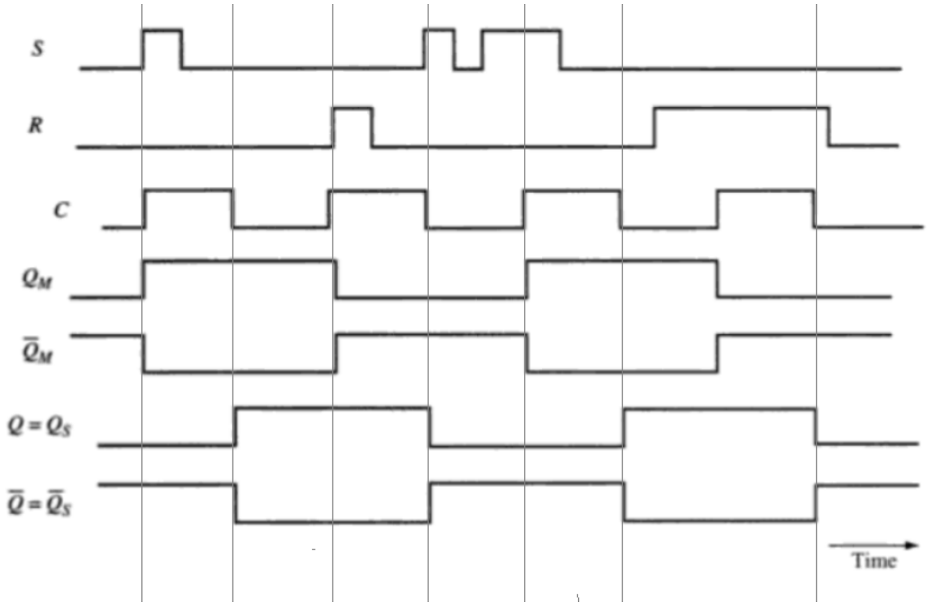


Figure 19: Timing diagram of Master Slave SR Flip-Flop.

- At the first rising edge of the clock pulse, S=1 and R=0, hence QM=1, QM'=0.
- At the falling edge of the clock pulse, slave latch takes QM and QM' as SS and RS inputs respectively and slave outputs Q=1 and Q'=0.
- At the second rising edge of the clock pulse, S input is zero and R input is high, hence master latch enters into reset state, i.e., QM=0 and QM'=1.
- At the second falling edge of the clock pulse, the master latch outputs are reflected at the slave latch.
- It is observed that, during the OFF period of the clock pulse, S input line has changed, but it is not reflected on Master latch, since master is disabled during the OFF period of the clock pulse.
- If S=0 and R=0, the master slave flip flop retains the previous state and if S=1 and R=1, the master slave SR flip-flop will enter invalid state.

Logic Symbol

Figures (20a and b) shows the logic symbols of Master slave SR flip-flop.

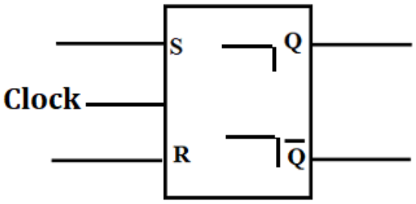


Figure 20: Logic Symbol of Master Slave SR flip-flop

Note: The symbol, ¬ is the output postpone indicator, indicates the output of the response postponed until the negative edge of the clock pulse.

2. Master Slave JK Flip-flop

The invalid state of SR flip-flop can be eliminated by using Master slave JK flip-flop, figure () shows the logic diagram of master slave JK flip-flop, which consisting cascade connection of two SR latches with feedback. First latch is called master latch and second latch is called slave latch.

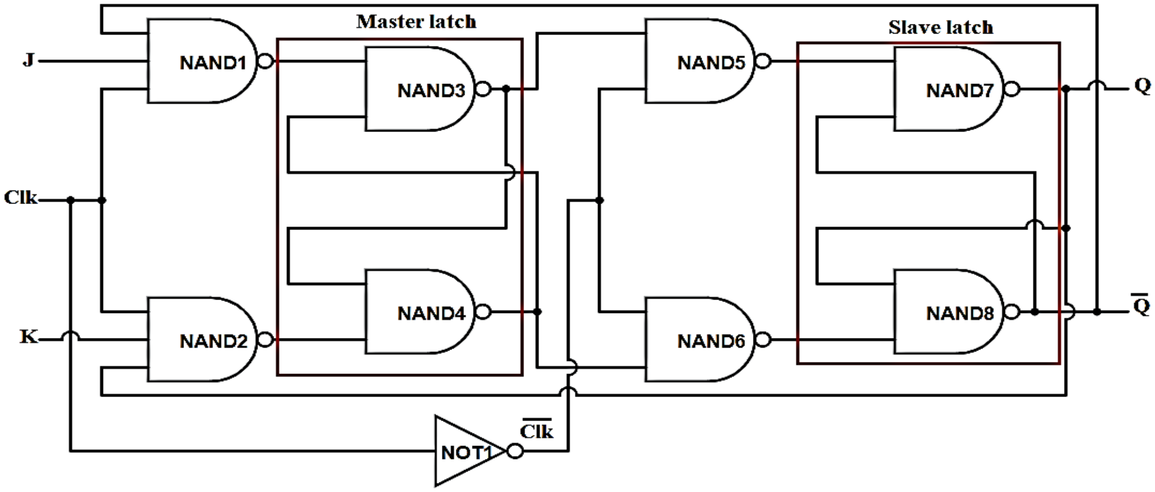


Figure 21: Logic diagram of Master Slave JK Flip-Flop

The master latch will drive the slave latch, at the rising edge of the clock pulse, the master latch gets enabled and responds for any change on input lines S and R. The slave latch starts activating at the negative edge of the clock pulse and responds to the inputs taken by the output of master latch.

The master slave JK flip-flop is converted invalid state of SR flip-flop to toggle condition. This operation is discussed as follows.

- Assume, the initial state of the flip-flop is Q=1 and Q'=0, At the rising edge of the clock pulse, if J and K inputs are 1's, the master latch resets and slave latch also resets at the falling edge, hence, Q=0 and Q'=1.
- Assume, the initial state of the flip flop is Q=0 and Q'=1. At the rising edge of the clock pulse, if J=1 and K=1, the master lath sets and slave latch also sets at the negative edge of the clock pulse. i.e. Q=1 and Q'=0.

  By observing the above cases, the output of the flip-flop is the complement of the previous state called toggling.

Table (18) summarizes the functionality of Master slave JK flip-flop.

Clock	J	K	Q_{N+1}	$\overline{Q_{N+1}}$
0	X	X	Q_N	$\overline{Q_N}$
1	0	0	Q_N	$\overline{Q_N}$
1	0	1	0	1
1	1	0	1	0
1	1	1	Toggle	

Table 18: Function Table of Master slave JK flip-flop

Timing diagram

Figure () shows the timing diagram of master slave JK flip-flop

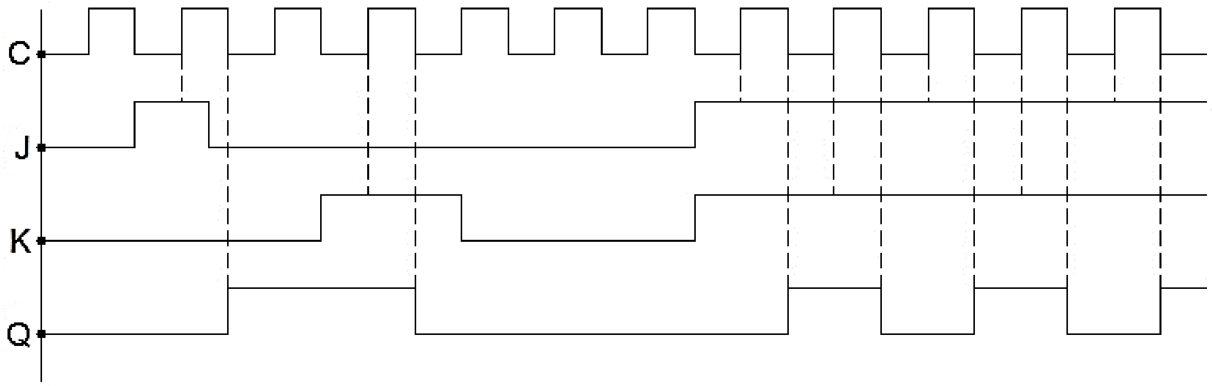


Figure 22: Timing diagram of Master Slave JK Flip-Flop

At the trailing edges of the clock pulse, the master slave responds as per the changes on input lines J and K. Refer function table.

0’s and 1’s catching

If the slave latch is in set state, and logic -1 on the K-input line during the ON period of the clock signal causes, master latch to reset, then, the slave latch resets when the clock signal returns to zero. This behavior is called 0’s catching.

If the slave latch is in reset state, and logic-1 on the J-input line during the ON period of the clock signal causes, master latch to set, then, the lave latch sets when the clock signal returns to zero. This behavior is called 1’s catching.

NOTE: Once the master latch is reset by logic-1 signal on K input line, a subsequent logic -1 signal on the J input line during the same period in which c=1 does not change its state until C returns to zero, due to feedback. The feedback signal from slave latch Q’=0, keeps the output J-input NAND gate at logic-0.

3. D flip-flop using Master Slave SR flip-flop

D flip-flop is the data flip-flop, which is designed using master slave SR flip-flop, by placing an inverter between the S and R inputs shown in figure (23).

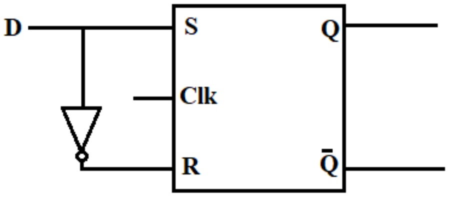


Figure 23: Logic Symbol of D flip-flop using Master Slave SR Flip-Flop

- If Clk=0, the output of the flip-flop is same as that of the previous state, irrespective of the levels of input D.
- If Clk=1, the data at the input line will be transferred to the output, i.e., Q=D.

Function Table

Table (19) summarizes the functionality of master slave SR D-flip flop

Clock	**D**	$\boldsymbol{Q_{N+1}}$	$\boldsymbol{\overline{Q_{N+1}}}$
0	X	Q_N	$\overline{Q_N}$
1	0	0	1
1	1	1	0

Table 19: Function Table of D Flip-flop

4. T Flip-flop using Master Slave JK flip-flop

The toggling state of master slave JK flip-flop is used for the design of counters, hence special type of flip-flop can be designed by joining J and K input lines shown in figure (24). This type of flip-flop is called T flip-flop; T stands for Toggle.

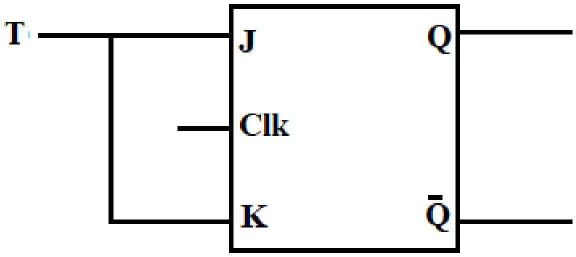


Figure 24: Logic symbol of pulse triggered triggered T flip-flop

Function table:

The functionality of the T flip-flop is summarized in the table (20).

Clock	**T**	$\boldsymbol{Q_{N+1}}$	$\overline{\boldsymbol{Q_{N+1}}}$
0	X	Q_N	$\overline{Q_N}$
1	0	Q_N	$\overline{Q_N}$
1	1	$\overline{Q_N}$	Q_N

Table 20: Function Table of T flip-flop

Edge Triggered Flip-flops.

The edge triggered flip-flops, use just one of the edges of the clock signal to effective reading of the information input line, this is referred t as edge triggering.

Two types of edges in clock signal, positive edge and negative edge, i.e., triggering occurs either on positive edge or negative edge.

The response of the flip-flops on triggering edge immediately occurs on rising edge or falling edge. Once triggering occurs, flip-flop is insensitive to the changes on input line until the next triggering edge.

1. **Positive edge triggered D Flip-flop.**

Positive edge triggered D flip-flop can be designed using three pairs of cross coupled NAND gates, i.e., positive edge triggered D flip-flop consisting of three S'R' latches shown in figure (25).

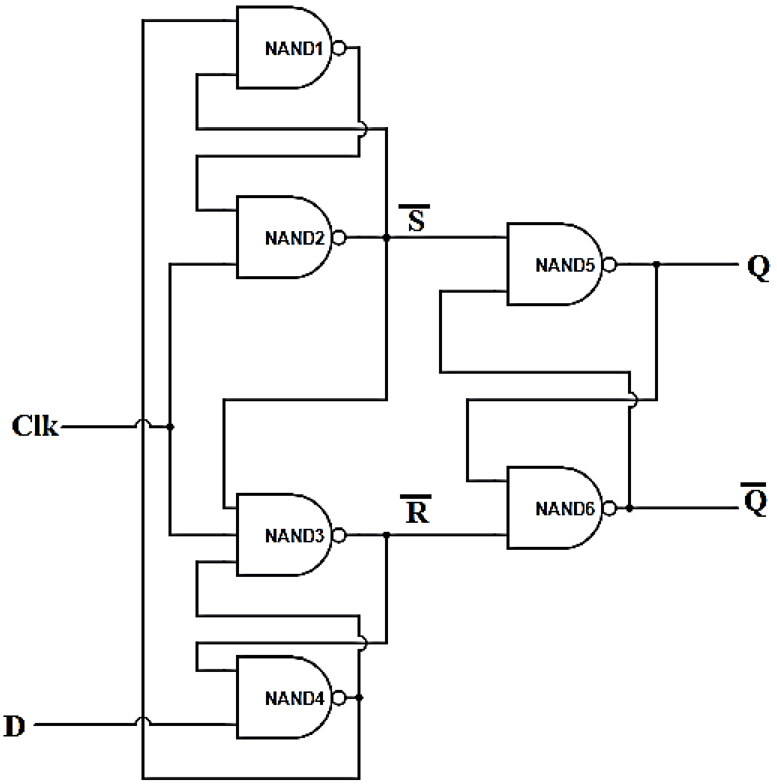


Figure 25: Logic diagram of Positive edge triggered D flip-flop

The working of the flip-flop is described as follows.

Case (i): If clock =0, regardless of the input at D, the outputs of NAND gates 2 and 3 are 1. These signals are the inputs for the NAND gates 5 and 6 (NAND gates 5 and 6 acts as an S'R' latch), if S'=1 and R'=1, S'R' latch retains the previous state.

Case(ii): At the rising edge of the clock pulse and D=0, the output of NAND gates 2 and 3 are 1 and 0 respectively, this causes the S'R' latch to reset. It is observed that, during the ON period of the clock pulse, if D input changes to 1, the flip-flop remains at the reset state. The flip-flop will not recognize any change occurs at input line until the next rising edge of the clock pulse.

Case (iii): At the rising edge of the clock pulse and D=1, the output of NAND gates 2 and 3 are 1 and 0 respectively, this causes the S'R' latch to Set. It is observed that, during the ON period of the clock pulse, if D input changes to 0, the flip-flop remains at the set state. The flip-flop will not recognize any change occurs at input line until the next rising edge of the clock pulse.

Table (21) summarizes the functionality of positive edge triggered D flip-flop.

Function table.

Inputs			**Outputs**		
D	**Clk**		$\mathbf{Q_{n+1}}$	$\overline{\mathbf{Q_{n+1}}}$	
0	↑		0	1	Set
1	↑		1	0	Reset
X	0		Q_n	$\overline{Q_n}$	Previous state
X	1		Q_n	$\overline{Q_n}$	Previous state

Table 21: Function Table of Positive edge triggered D flip-flop

Figure (26) shows the logic symbols of positive edge triggered D flip-flop.

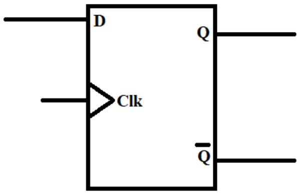


Figure 26: Logic Symbol of positive edge triggered D flip-flop

Timing diagram

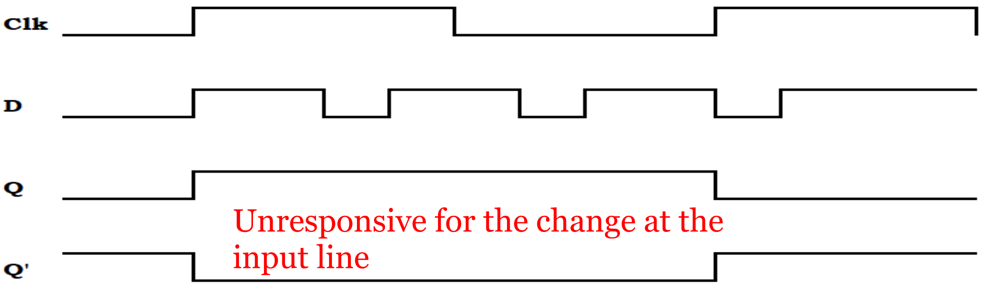


Figure 27: Timing diagram of positive edge triggered D flip-flop

2. Negative edge triggered D flip-flop

Figure (28) shows the logic diagram of negative edge triggered flip-flop, the functionality of this flip-flop is same as that of the positive edge triggered flip-flop except the state transition of the flip-flop takes place at the falling edge of the clock pulse.

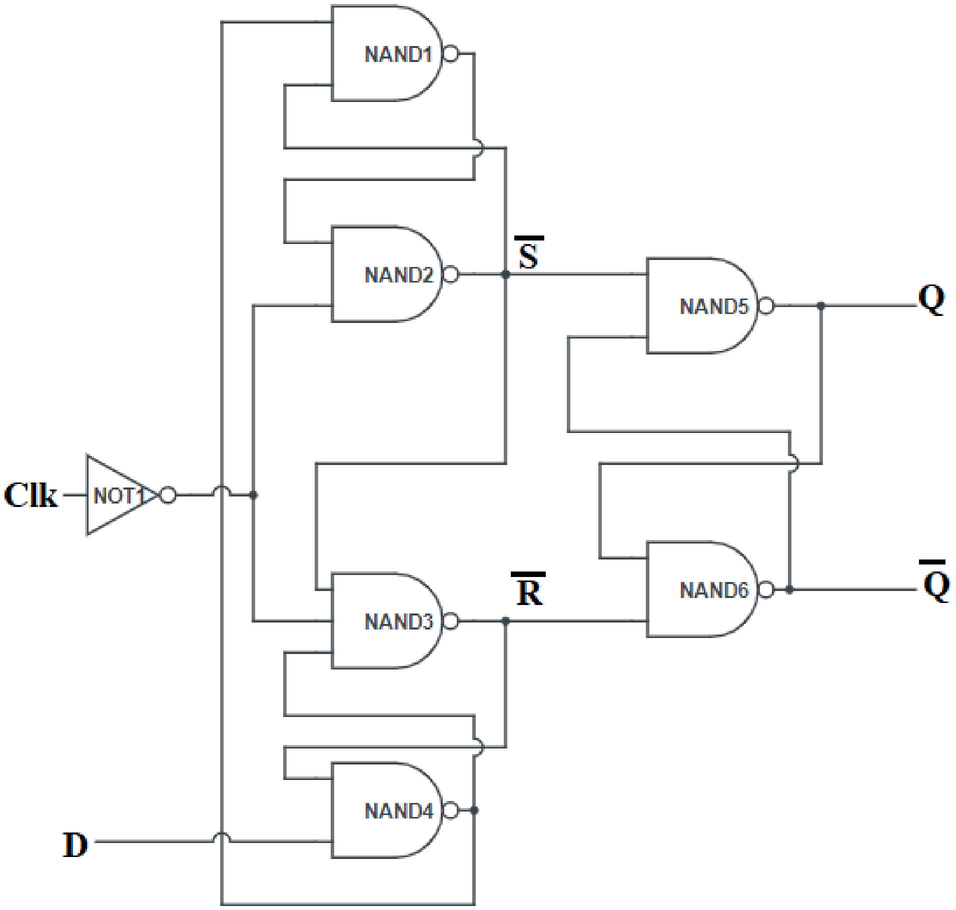


Figure 28: Logic diagram of neeagtive edge triggered D flip-flop

Function table

Inputs			Outputs		
D	Clk		Q_{n+1}	$\overline{Q_{n+1}}$	
0	↓		0	1	Set
1	↓		1	0	Reset
X	0		Q_n	$\overline{Q_n}$	Previous state
X	1		Q_n	$\overline{Q_n}$	Previous state

Table 22: Function Table of Negative edge triggered D flip-flop

Logic symbol

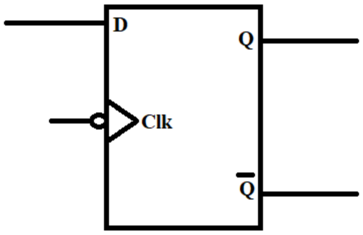


Figure 29: Logic symbol of negative edge triggered D flip-flop

Asynchronous Inputs

All information input lines of flip flops are synchronous inputs, to have more flexibility, two additional inputs have been introduced to set and reset forcibly. These input lines are called asynchronous inputs denoted as PR’ and CLR’, that is these input line do not depend on the control/clock signal.

Figure (30) shows the positive edge triggered D flip-flop with asynchronous inputs.

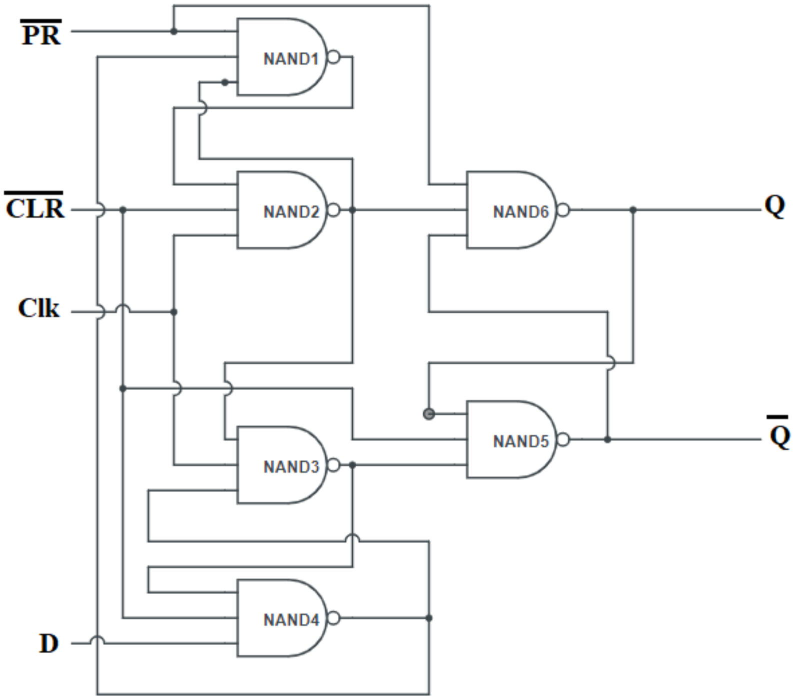


Figure 30: Logic diagram of edge triggered D flip-flop with asynchronous inputs

The operation of asynchronous input on flip-flop is discussed as follows.

Case (i) : If PR’=0 and CLR’=1, regardless of the input clock and D, the outputs of gates 5 and 6 are 1 and 0, hence with asynchronous inputs, the initial state of the device can be set into set state.

Case (ii): If PR’=1 and CLR’=0, regardless of the inputs clock and D, the outputs of gates 5 and 6 are 0 and 1 respectively, hence the asynchronous inputs, force the device to enter into reset state.

Case (iii): If PR'=0 and CLR'=0, regardless of the clock and D inputs, the outputs of gates 5 and 6 are 1, which is invalid state, hence need to avoid applying logic zero through the asynchronous inputs.
Case (iv): if PR'=1and CLR'=1, the flip flop responds as per the input line and clock signal.

Table (23) summarized the functionality.

Inputs					Outputs		
$\overline{\mathbf{PR}}$	$\overline{\mathbf{CLR}}$	**D**	**Clk**		$\mathbf{Q_{n+1}}$	$\overline{\mathbf{Q_{n+1}}}$	
0	1	X	X		1	0	Initial state forced to set.
1	0	X	X		0	1	Initial state forced to reset
0	0	X	X		1	1	Invalid
1	1	0	↑		0	1	Set
1	1	1	↑		1	0	Reset
1	1	X	0		Q_n	$\overline{Q_n}$	Previous state
1	1	X	1		Q_n	$\overline{Q_n}$	Previous state

Table 23: Function Table of Positive edge triggered D flip-flop with asynchronous inputs

Logic symbol

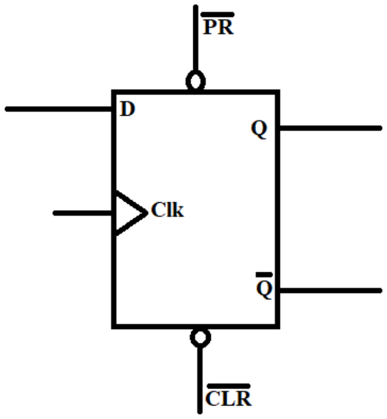


Figure 31: Logic symbol of edge triggered D flip-flop with asynchronous inputs

III. Registers

A single flip-flop is able to store single bit information either 0 or 1, but to store more than one bit information, a group of flip-flops need to be connected. A group of flip-flops is called a register. If a register contains n flip-flops, it is able to stor n bit information.

Registers can be used to generate the specified sequence and can also be used to shift the content of flip-flop position wise, based on this the applications of registers are classified into two categories.

1. Shift registers and
2. Counters

1. Shift registers

A shift register is an entity of flip-flops, which are capable of shifting the state of flip-flop positionwise in one direction or two directions.

Example: To store 4-bit data 1001, four flip-flops are used.

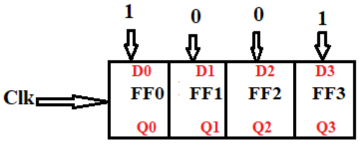


Figure 32: Basic shift register

Based on the direction of shifting the content of flip-flop, shift registers are classified into two types.

a) Unidirectional shift registers

The unidirectional shift registers, shifts the contents of flip-flops in only one direction either right shift or left shift.

b) Bideirectional shift registers

The Bidirectional shift registers are capable of performing right shift as well as left shift through a proper control signal.

c) Universal shift register

The universal shift registers are capable of performing right shift as well as left shift through a proper control signal along with parallel loading and memory.

Shift registers further classified into four types, based on the way the input is loaded and the output is received.

a) Serial input and serial ouptut

The information will be loaded serially through a single line and output will be received serially through a single line is called serial input and serial output (SISO) shift register shown in figure (33).

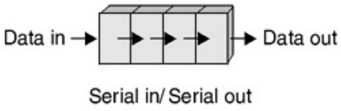


Figure 33: SISO Shift register

b) Serial input and parallel output

the information will be loaded serially through a single line and output will be received in parallel through multiple lines is called serial input parallel output (SIPO) shift register shown in figure (34).

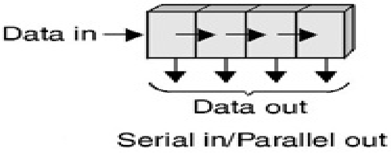


Figure 34: SIPO Shift register

c) Parallel input and serial output

The information bits will be loaded in parallel through multiple lines and output will be received through a single line is called parallel input serial output (PISO) shift register shown in figure (35).

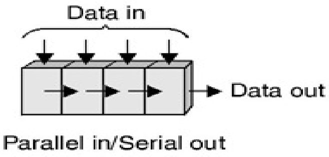


Figure 35: SISO Shift register

d) Parallel input and parallel output

The information bits will be loaded in parallel through multiple lines and output will be taken in parallel with multiple output lines is called parallel input parallel output (PIPO) shift register shown in figure (36).

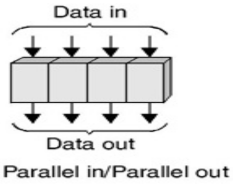


Figure 36: SISO Shift register

1. Serial input serial output (SISO) unidirectional shift register

Figure (37, shows the logic diagram of 4-bit sireial input serial output shift registers, designed using four D flip-flops, all the flip-flops are positive edge triggered flip flops.

Logic Diagram:

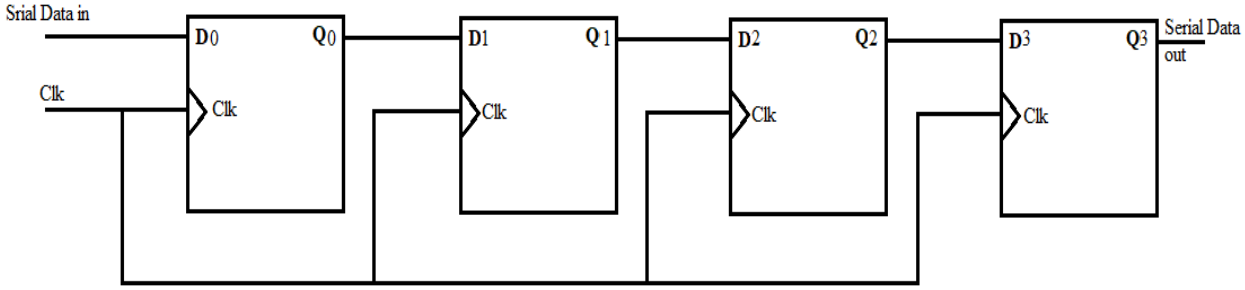
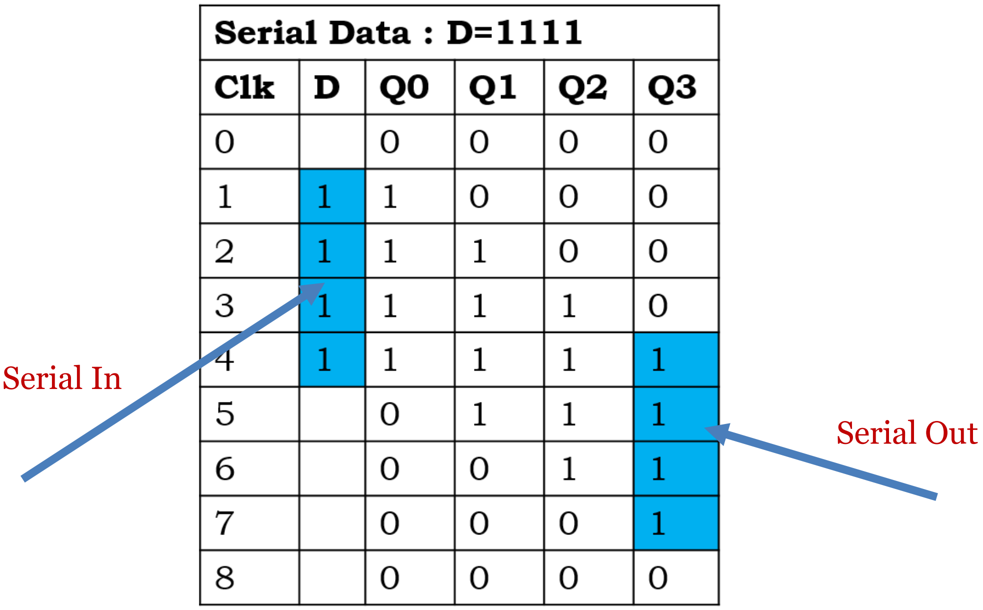


Figure 37: SISO Shift register

At the every rising edge of the clock pulse, the contents of flip-flops will be shifted towards the succeeding stages of the flip-flops position wise.

Truth table



Serial Data : D=1111					
Clk	**D**	**Q0**	**Q1**	**Q2**	**Q3**
0		0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1
5		0	1	1	1
6		0	0	1	1
7		0	0	0	1
8		0	0	0	0

Serial In
Serial Out

Table 24: Truth table of SISO shift register

Timing diagarm

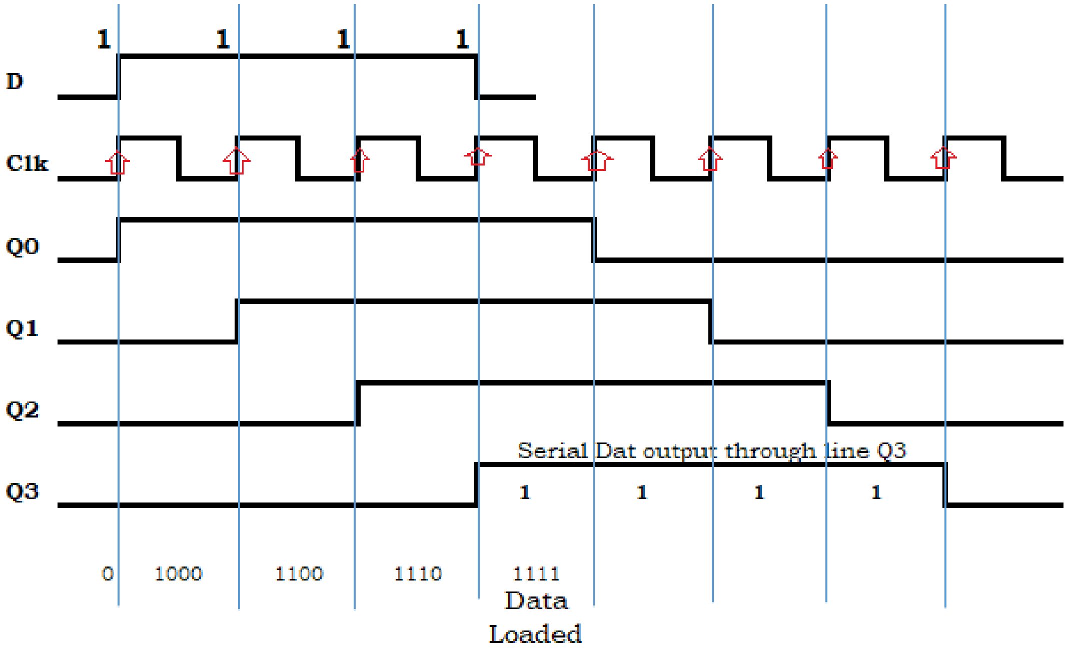


Figure 38: Timing diagram of SISO shift register

2. Serial input parallel output (SIPO) unidirectional shift register

Figure (39), shows the logic diagram of 4-bit sireial input parallel output shift registers, designed using four D flip-flops, all the flip-flops are positive edge triggered flip flops.

Logic Diagram:

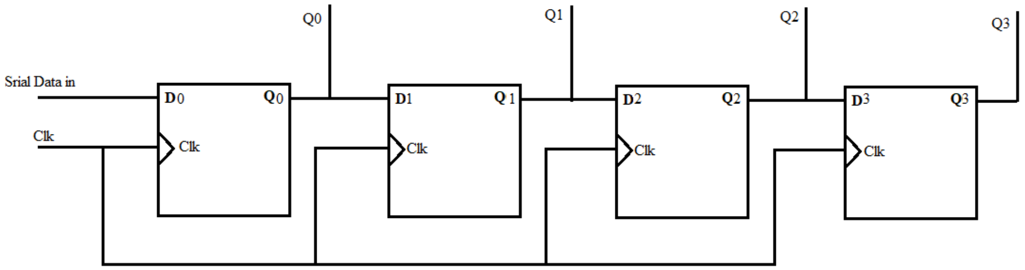


Figure 39: SIPO Shift register

At the every rising edge of the clock pulse, the contents of flip-flops will be shifted towards the succeeding stages of the flip-flops position wise.

Truth table

Serial Data : D=1111					
Clk	**D**	**Q0**	**Q1**	**Q2**	**Q3**
0		0	0	0	0
1	1	1	0	0	0
2	1	1	1	0	0
3	1	1	1	1	0
4	1	1	1	1	1

Serial In

Parallel Out

Table 25: Truth table of SIPO shift register

Timing diagarm

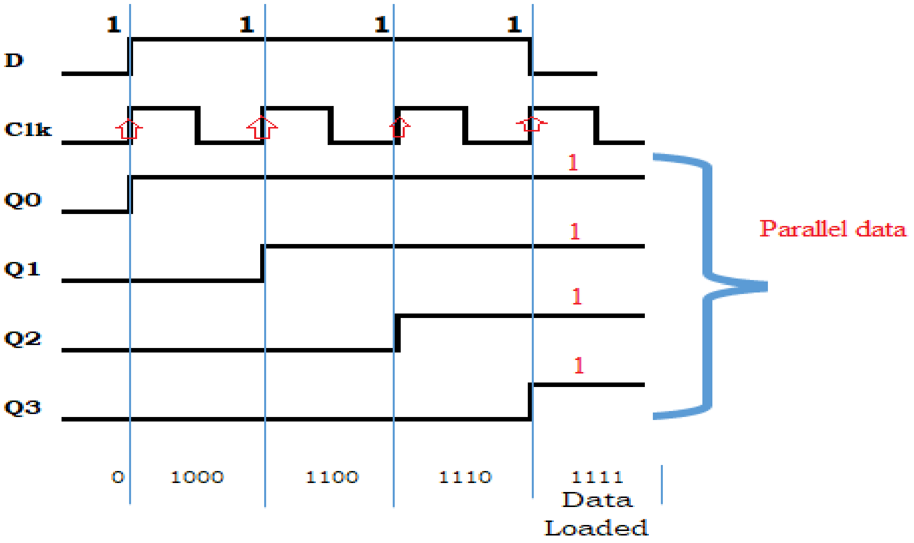


Figure 40: Timing diagram of SIPO shift register

3. Parallel input Parallel output (PIPO) unidirectional shift register

Figure (41), shows the logic diagram of 4-bit Parallel input parallel output shift registers, designed using four D flip-flops, all the flip-flops are positive edge triggered flip flops.

Logic Diagram:

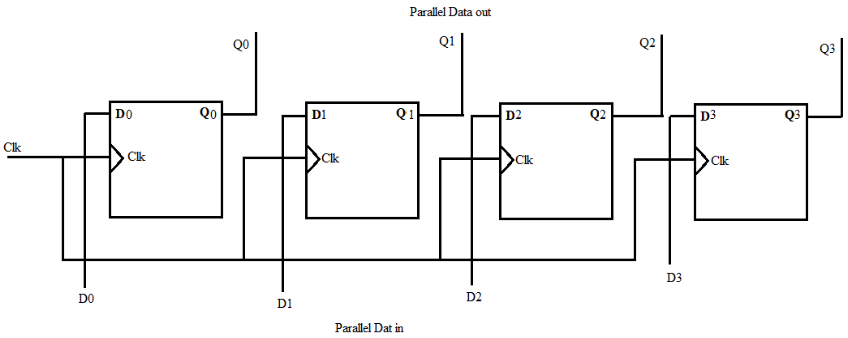


Figure 41: PIPO Shift register

At the every rising edge of the clock pulse, the contents of flip-flops will be shifted towards the succeeding stages of the flip-flops position wise.

Truth table

Parallel Data : D=1111				
Clk	**Q0**	**Q1**	**Q2**	**Q3**
0	0	0	0	0
1	1	1	1	1

Parallel Out

Table 26: Truth table of PIPO shift register

Timing diagram

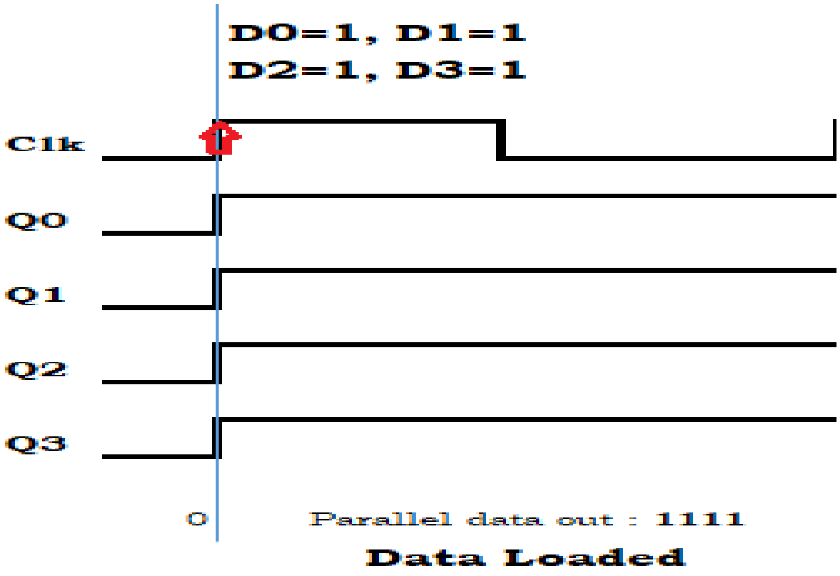


Figure 42: Timing diagram of PIPO shift register

4. Parallel input Serial output (PISO) unidirectional shift register

Figure (43), shows the logic diagram of 4-bit Parallel input serial output shift registers, designed using four D flip-flops, all the flip-flops are positive edge triggered flip flops.

- $Shift/\overline{Load} = 0(Parallel\ Loading)$
- $Shift/\overline{Load} = 1(Shifting)$

Logic Diagram:

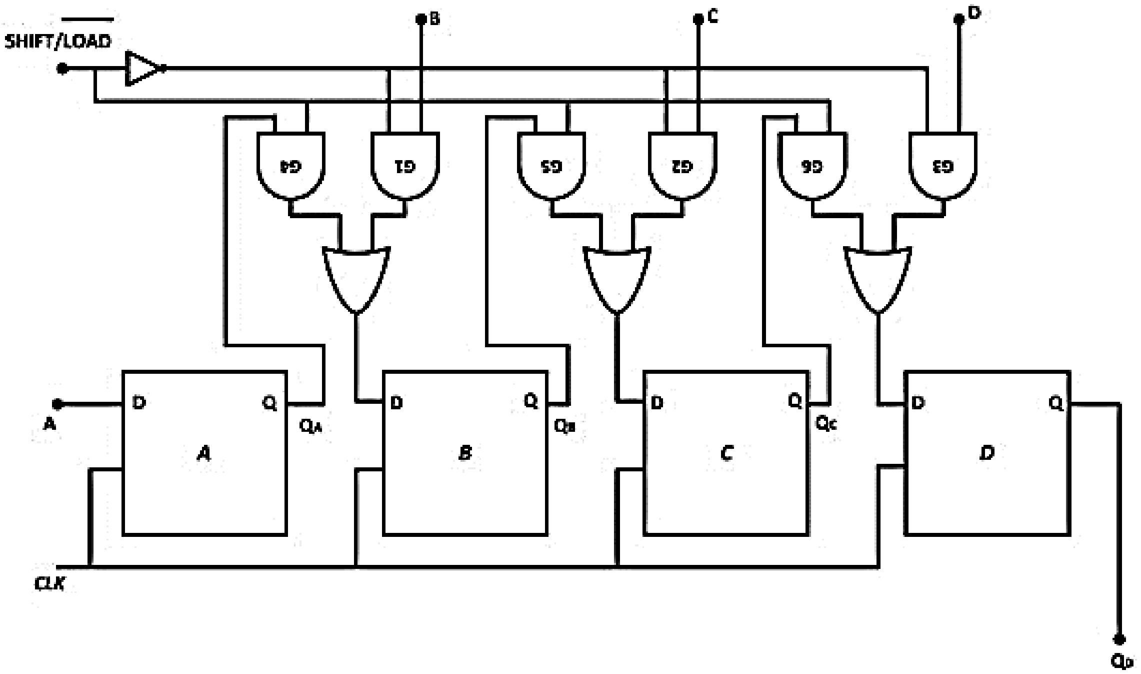


Figure 43: PISO Shift register

At the every rising edge of the clock pulse, the contents of flip-flops will be shifted towards the succeeding stages of the flip-flops position wise.

Truth table

Parallel Data : D=1111				
Clk	**Q0**	**Q1**	**Q2**	**Q3**
0	0	0	0	0
1	1	1 Parallel Load	1	1 Serial Out
2	0	1	1	1
3	0	0	1	1
4	0	0	0	1

Table 27: Truth table of PISO shift register

Timing diagram

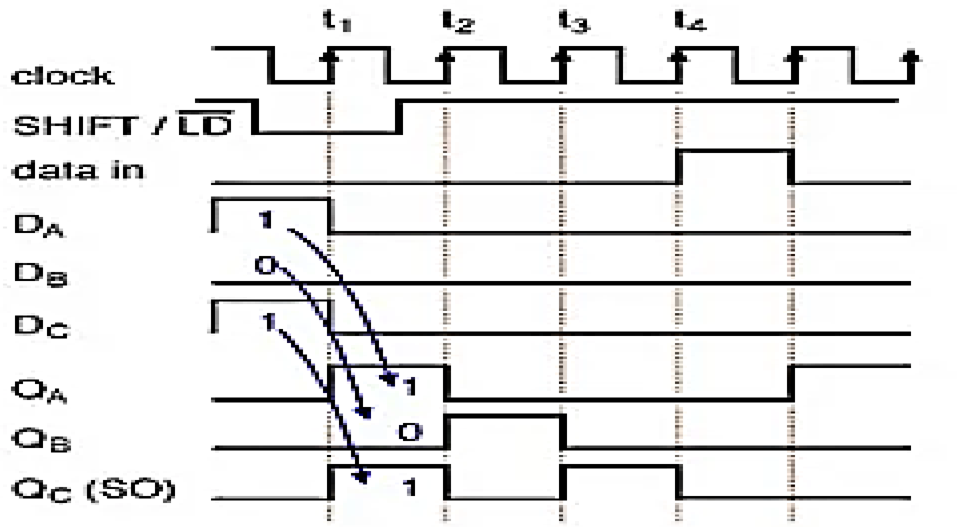


Figure 44: Timing diagram of PISO shift registerS

NOTE:

- A single circuit which performs all the shift register operations in single direction shown in figure (45).
- $Shift/\overline{Load} = 0(Parallel - in)$
- $Shift/\overline{Load} = 1(Serial - in)$

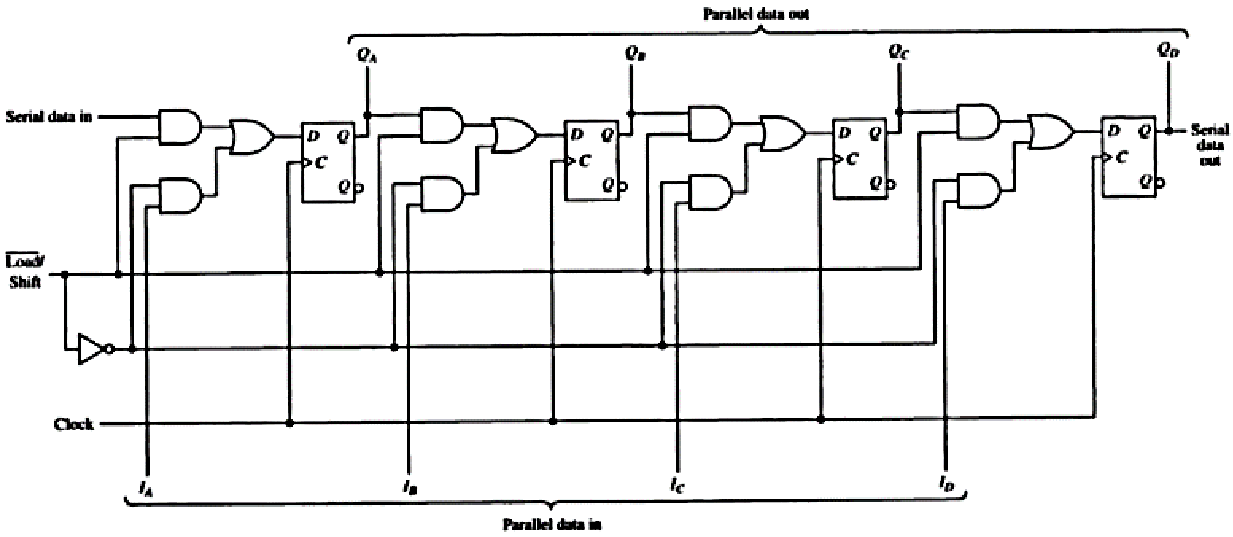


Figure 45: Single circuit which performs SISO, SIPO, PIPO and PISO

Bidirectional Shift register

Performs both left shift and right shift

M=0 (Shift left operation), M=1 (Shift right operation)

Logic diagram

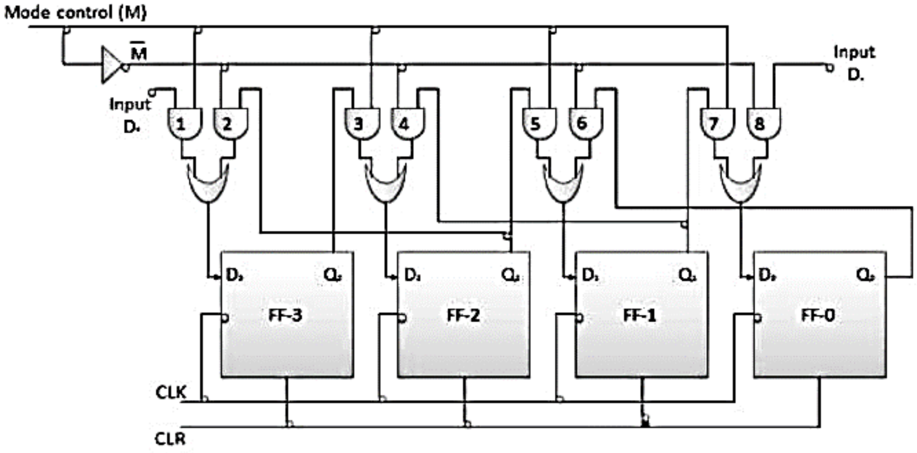


Figure 46: Logic diagram of bidirectional shift registers

Universal Shift Register

Bidirectional Shift Register + (SISO, SIPO, PIPO, PISO)+Memory

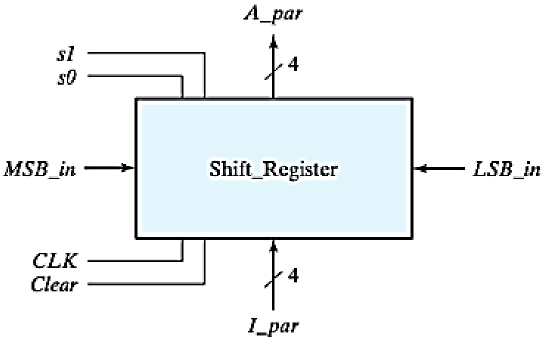


Figure 47: Logic symbol of universal shift register

S1	**S0**	**Register Operation**
0	0	No Change (Memory)
0	1	Shift left
1	0	Shift right
1	1	Parallel loading

Table 28: Universal shift opearions table

Logic Diagram

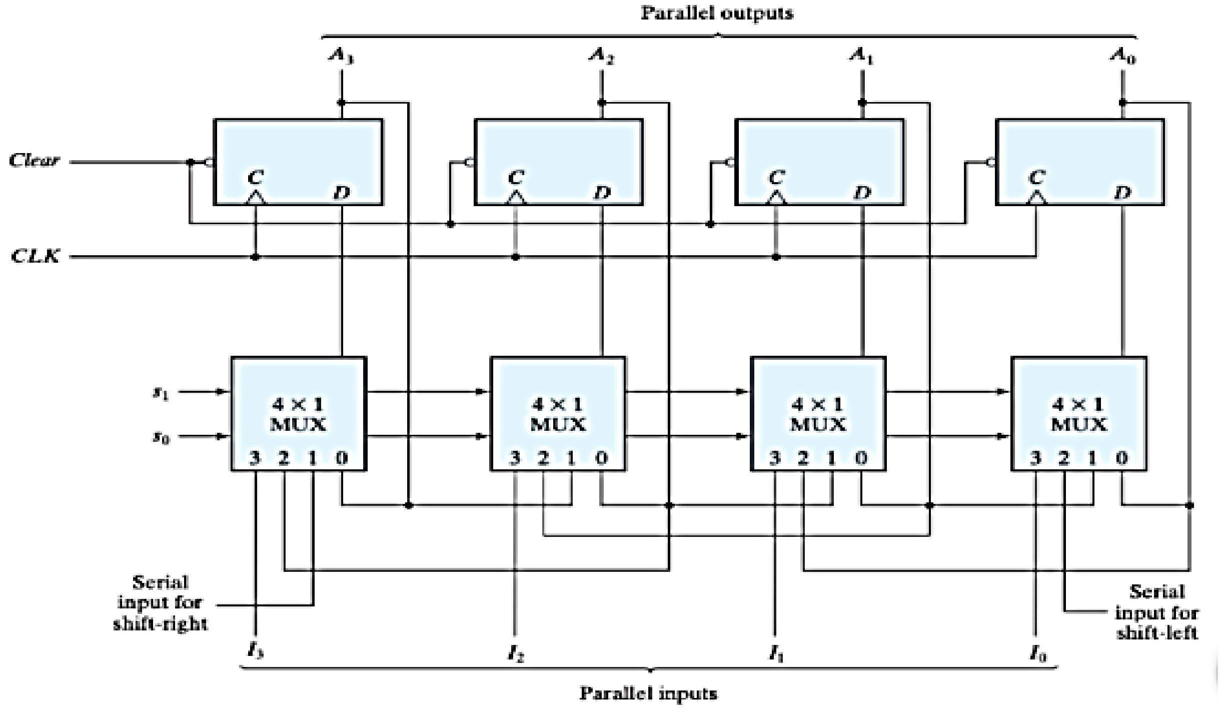


Figure 48: Logic diagram of universal shift register

NOTE: Students are asked to write the detailed working principle of all the shift registers (Explanation for all shift registers are omitted in this notes and asked to explain).

IV. Counters

Registers can also be used to generate the specified sequence, counting the binary values either incrementally or in decrement is an example of sequence. Hence, counter is a sequential circuit, which counts the pulses either in ascending or descending order.
n-bit counter will be designed using n-flip-flops.

Counters are classified into two types based on the application of clock signal.

1. Synchronous counters and
2. Asynchronous counters.
3.

The clock signal is applied to all the flip-flops of a counter circuit simultaneously is called synchronous contours.

Figure (49) shows the two-bit counter, the clock signal for the two flip-flops are supplied simultaneously. $\overline{CLR}$ is an asynchronous input, logic-0 to this pin sets the initial value of all the flip-flop to zero.

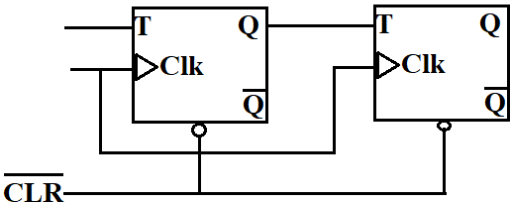


Figure 49: Example for synchronous counter

The output of the previous state flip serves as a clock input to the next state flip-flop is called asynchronous counters.

Figure (50) shows that, the clock input for the second flip-flop will be supplied from the output of first flip-flop.

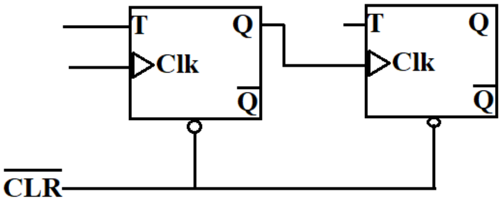


Figure 50: Example for asynchronous counter

Similarly, counters can be designed using any type of flip-flops, depending on the counting sequence, counters are further classified into three types.

1. Up-counter and
   Counts the clock pulse incrementally.
2. Down-counter
   Counts the clock pulse in decrement order.
3. Ring counter
   Outputs the specified sequence in circular/ring format.

Asynchronous UP-Counter (Binary Ripple counter)

Counters whose counting sequence corresponds to that of binary numbers are called binary counters. The modulus of binary counter is 2^n, where n is the number of flip-flops required to design the counter.

Example:

if n=3, the counter performs counting from 000 to 111 (8 combinations(0 to $2^n - 1$)), hence three flip-flops are required.

Similarly

If n=4, the counter performs counting from 0000 to 1111 (16 combinations(0 to $2^n - 1$)), hence four flip-flops are required.

Figure (51) shows the logic diagram of 4-bit up counter using positive edge triggered T flip-flops. Each positive transition of clock makes the flip-flop to toggle.

Logic Diagram

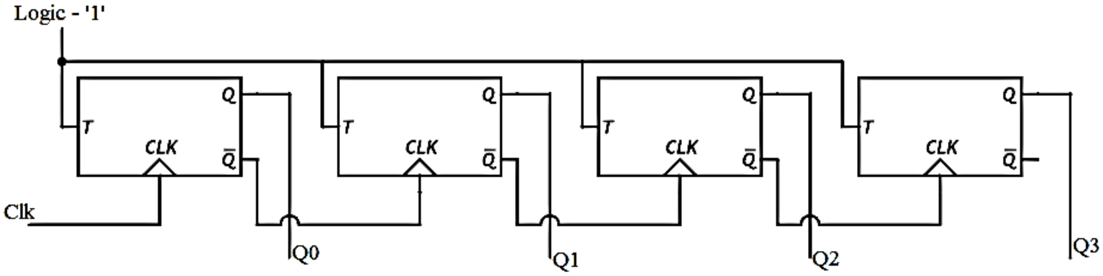


Figure 51: Example for asynchronous counter

Output $\bar{Q}$ of previous order flip-flop serves clock signal for the next order flip-flops, and clock pulse applied to all the flip-flops is not simultaneous, hence called as asynchronous counter. T inputs of all flip-flops are connected to logic-1, which acts as a toggle device. Output of the last stage flip-flop is MSB and the output of first stage flip-flop is LSB.

For every rising edge of the clock pulse the content of first stage(Q0) T-flip-flop will toggles, for every falling edge of the Q0 the content of second stage(Q1) T-flip-flop will toggles, this process continues until the last stage flip-flop.

Assume, initially, the contents of all the four T flip-flops are zero. At the rising edge of the clock pulse Q0 becomes '1' and the remaining flip-flop outputs remains zero. The later stage flip-flops output change occurs at the next falling edge of the previous stage flip-flop outputs, shown in the timing diagram figure (52).

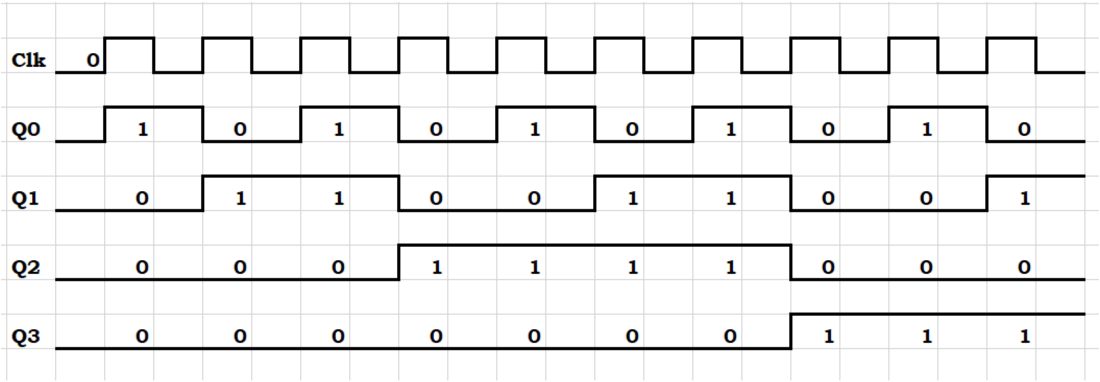


Figure 52: Timing diagram of 4-bit asynchronous binary ripple up counter

Change in the state of flip-flops occurs through the outputs of the previous stage flip-flops, that is the effect of count pulse must ripple through the counter. Hence the name ripple counter.

Asynchronous Down-Counter (Binary Ripple counter)

Figure (53) shows the logic diagram of 4-bit down counter using positive edge triggered T flip-flops. Each positive transition of clock makes the flip-flop to toggle.

Logic Diagram

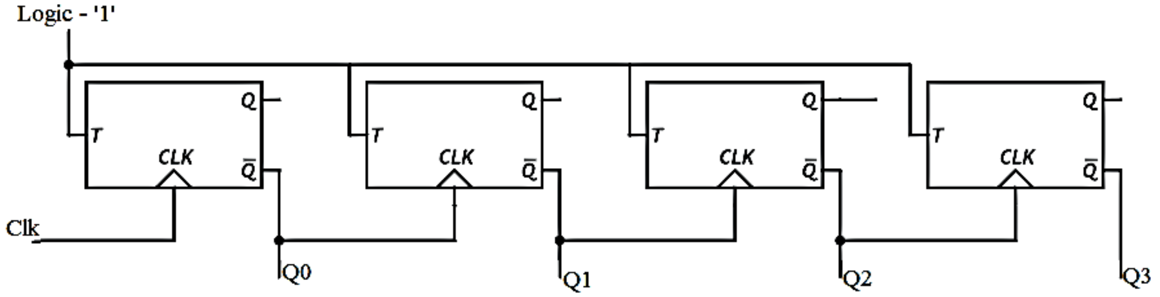


Figure 53: Timing diagram of 4-bit asynchronous binary ripple up counter

Output $\bar{Q}$ of previous order flip-flop serves clock signal for the next order flip-flops, and clock pulse applied to all the flip-flops is not simultaneous, hence called as asynchronous counter. T inputs of all flip-flops are connected to logic-1, which acts as a toggle device. Output $\overline{Q_3}$ of the last stage flip-flop is MSB and the output $\overline{Q_0}$ of first stage flip-flop is LSB.

For every rising edge of the clock pulse the content of first stage($\overline{Q0}$) T-flip-flop will toggles, for every falling edge of the Q0 the content of second stage(Q1) T-flip-flop will toggles, this process continues until the last stage flip-flop.

Assume, initially, the contents of all the four T flip-flops are zero, hence complement of the outputs is 1's. At the rising edge of the clock pulse $\overline{Q0}$ becomes '0' and the remaining flip-flop complement outputs remains 1. The later stage flip-flops output change occurs at the next falling edge of the previous stage flip-flop outputs, shown in the timing diagram figure (54).

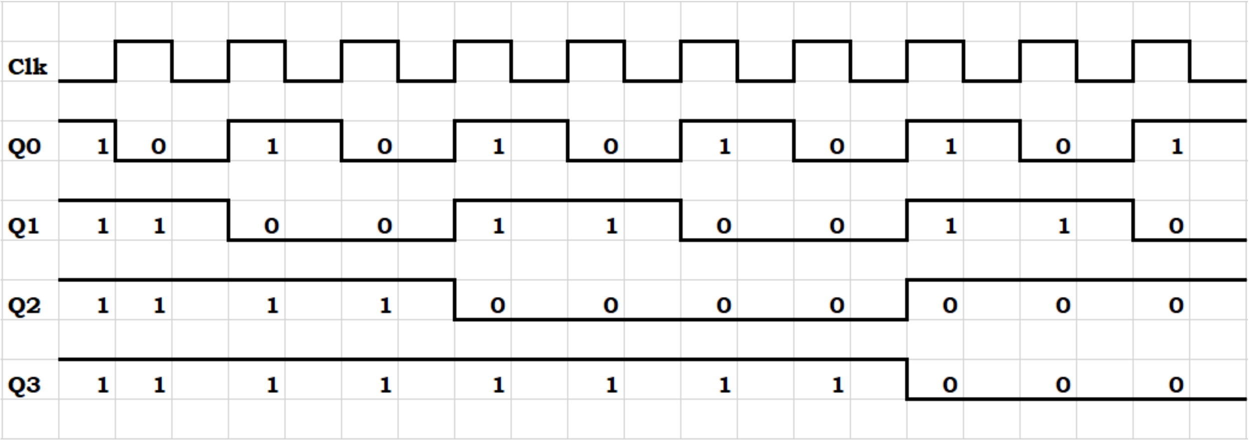


Figure 54: Timing diagram of 4-bit asynchronous binary ripple up counter

Change in the state of flip-flops occurs through the outputs of the previous stage flip-flops, that is the effect of count pulse must ripple through the counter. Hence the name ripple counter

V. Miscelaneous Concepts

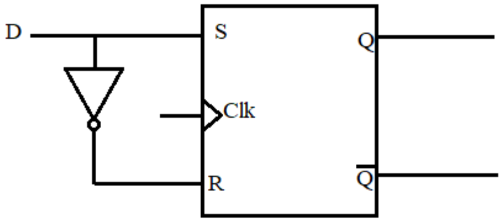
Flip-flops conversion

1. SR to D flip-flop

Excitation table

Q_n	D	Q_{n+1}	S	R
0	0	0	0	X
0	1	1	1	0
1	0	0	0	1
1	1	1	X	0

Logic diagram



Excitation equations

$$S = D \text{ and } R = \overline{D}$$

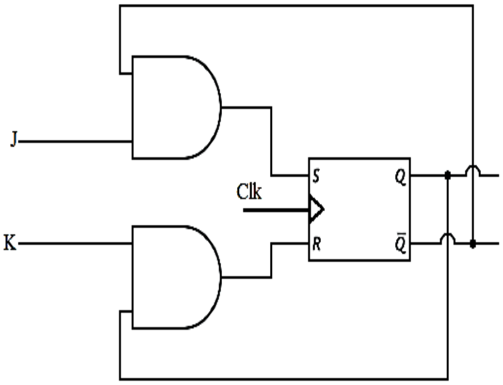
2. SR to JK flip-flop

Excitation equations

$$S = \overline{Q_n}J \text{ and } R = Q_nK$$

Excitation table

Q_n	J	K	Q_{n+1}	S	R
0	0	0	0	0	X
0	0	1	0	0	X
0	1	0	1	1	0
0	1	1	1	1	0
1	0	0	1	X	0
1	0	1	0	0	1
1	1	0	1	X	0
1	1	1	0	0	1

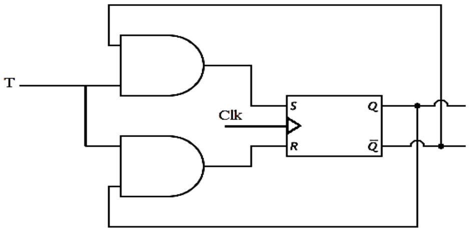


3. SR to T flip-flop

Excitation table

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

Logic Diagram



Excitation equations

$$S = T\overline{Q_n} \text{ and } R = TQ_n$$

4. JK to D flip-flop

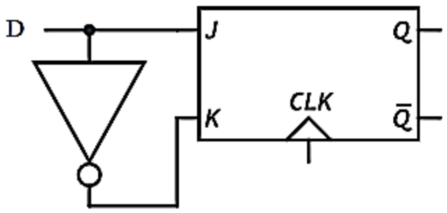
Excitation table

Q_n	D	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	1	X
1	0	0	X	1
1	1	1	X	0

Excitation equations

$$J = D \text{ and } K = \overline{D}$$

Logic diagram



5. JK to T flip-flop

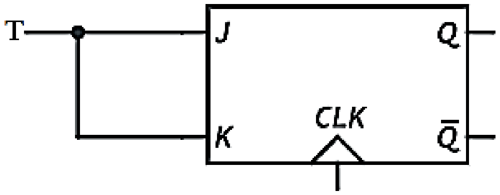
Excitation table

Q_n	T	Q_{n+1}	J	K
0	0	0	0	X
0	1	1	1	X
1	0	1	X	0
1	1	0	X	1

Excitation equations

$$J = T \text{ and } K = T$$

Logic diagram



6. D to T flip-flop

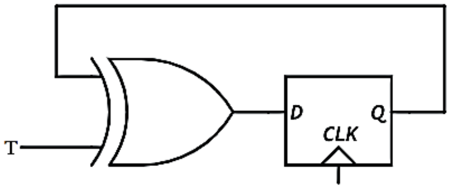
Excitation table

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

Excitation equation

$$D = T \oplus Q_n$$

Logic diagram



7. D to JK flip-flop

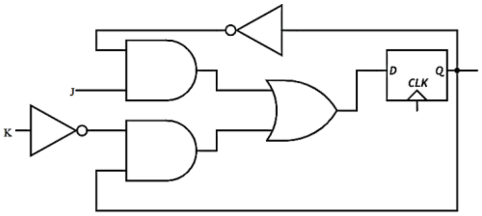
Excitation table

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

Excitation equation

$$D = Q_n\overline{K} + \overline{Q_n}J$$

Logic diagram

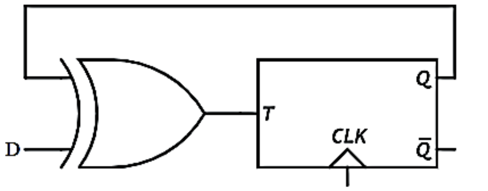


8. T to D flip-flop

Excitation table

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

Logic diagram



Excitation equation

$$T = Q_n \oplus D$$

9. T-JK Flip-flop

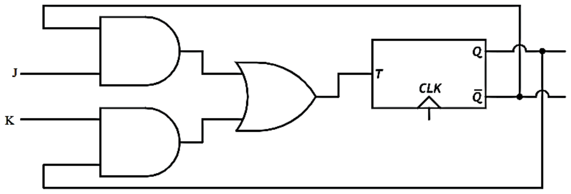
Excitation table

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

Excitation equation

$$T = Q_n K + \overline{Q_n} J$$

Logic diagram



NOTE: JK to SR, T to SR and D to SR conversion is not possible.

******

UNIT-IV : Counters design and state machines

Counters

Registers can also be used to generate the specified sequence, counting the binary values either incrementally or in decrement is an example of sequence. Hence, counter is a sequential circuit, which counts the pulses either in ascending or descending order.
n-bit counter will be designed using n-flip-flops.

Counters are classified into two types based on the application of clock signal.

4. Synchronous counters and
5. Asynchronous counters.
6.

The clock signal is applied to all the flip-flops of a counter circuit simultaneously is called synchronous contours.

Figure (49) shows the two-bit counter, the clock signal for the two flip-flops are supplied simultaneously. $\overline{CLR}$ is an asynchronous input, logic-0 to this pin sets the initial value of all the flip-flop to zero.

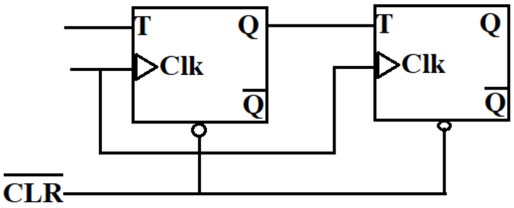


Figure 49: Example for synchronous counter

The output of the previous state flip serves as a clock input to the next state flip-flop is called asynchronous counters.

Figure (50) shows that, the clock input for the second flip-flop will be supplied from the output of first flip-flop.

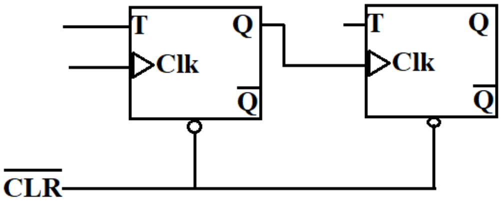


Figure 50: Example for asynchronous counter

Similarly, counters can be designed using any type of flip-flops, depending on the counting sequence, counters are further classified into three types.

4. Up-counter and
   Counts the clock pulse incrementally.
5. Down-counter
   Counts the clock pulse in decrement order.
6. Ring counter
   Outputs the specified sequence in circular/ring format.

Asynchronous UP-Counter (Binary Ripple counter)

Counters whose counting sequence corresponds to that of binary numbers are called binary counters. The modulus of binary counter is 2^n, where n is the number of flip-flops required to design the counter.

Example:

if n=3, the counter performs counting from 000 to 111 (8 combinations(0 to $2^n - 1$)), hence three flip-flops are required.

Similarly

If n=4, the counter performs counting from 0000 to 1111 (16 combinations(0 to $2^n - 1$)), hence four flip-flops are required.

Figure (51) shows the logic diagram of 4-bit up counter using positive edge triggered T flip-flops. Each positive transition of clock makes the flip-flop to toggle.

Logic Diagram

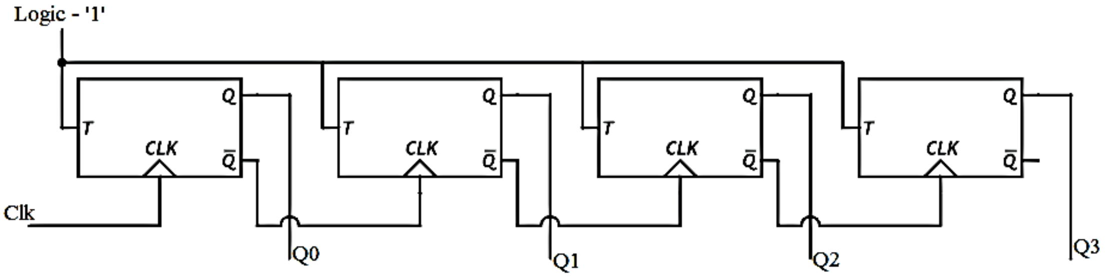


Figure 51: Example for asynchronous counter

Output $\bar{Q}$ of previous order flip-flop serves clock signal for the next order flip-flops, and clock pulse applied to all the flip-flops is not simultaneous, hence called as asynchronous counter. T inputs of all flip-flops are connected to logic-1, which acts as a toggle device. Output of the last stage flip-flop is MSB and the output of first stage flip-flop is LSB.

For every rising edge of the clock pulse the content of first stage(Q0) T-flip-flop will toggles, for every falling edge of the Q0 the content of second stage(Q1) T-flip-flop will toggles, this process continues until the last stage flip-flop.

Assume, initially, the contents of all the four T flip-flops are zero. At the rising edge of the clock pulse Q0 becomes ‘1’ and the remaining flip-flop outputs remains zero. The later stage flip-flops output change occurs at the next falling edge of the previous stage flip-flop outputs, shown in the timing diagram figure (52).

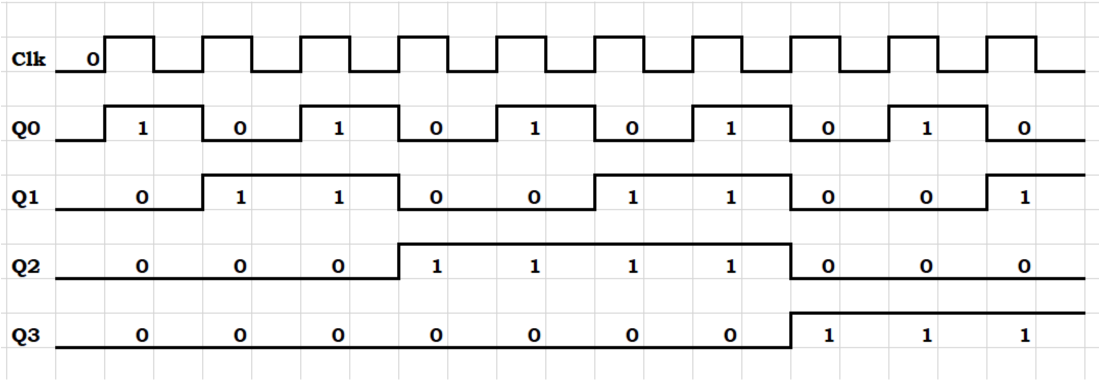


Figure 52: Timing diagram of 4-bit asynchronous binary ripple up counter

Change in the state of flip-flops occurs through the outputs of the previous stage flip-flops, that is the effect of count pulse must ripple through the counter. Hence the name ripple counter.
Asynchronous Down-Counter (Binary Ripple counter)
Figure (53) shows the logic diagram of 4-bit down counter using positive edge triggered T flip-flops. Each positive transition of clock makes the flip-flop to toggle.
Logic Diagram

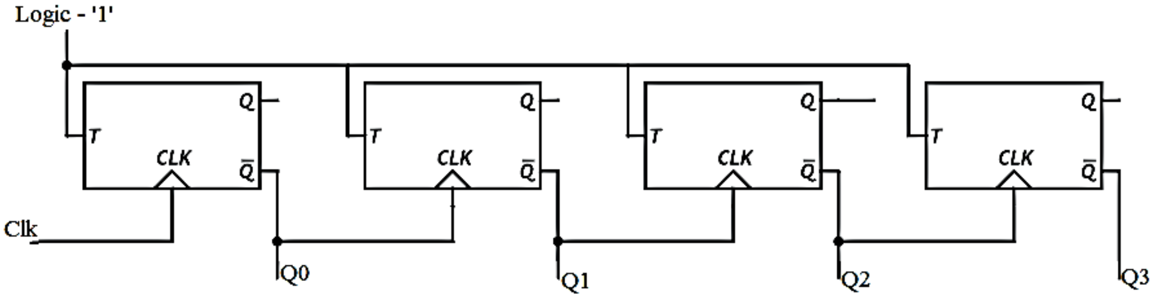


Figure 53: Timing diagram of 4-bit asynchronous binary ripple up counter

Output $\bar{Q}$ of previous order flip-flop serves clock signal for the next order flip-flops, and clock pulse applied to all the flip-flops is not simultaneous, hence called as asynchronous counter. T inputs of all flip-flops are connected to logic-1, which acts as a toggle device. Output $\overline{Q_3}$ of the last stage flip-flop is MSB and the output $\overline{Q_0}$ of first stage flip-flop is LSB.
For every rising edge of the clock pulse the content of first stage($\overline{Q0}$) T-flip-flop will toggles, for every falling edge of the Q0 the content of second stage(Q1) T-flip-flop will toggles, this process continues until the last stage flip-flop.
Assume, initially, the contents of all the four T flip-flops are zero, hence complement of the outputs is 1's. At the rising edge of the clock pulse $\overline{Q0}$ becomes '0' and the remaining flip-flop complement

outputs remains 1. The later stage flip-flops output change occurs at the next falling edge of the previous stage flip-flop outputs, shown in the timing diagram figure (54).

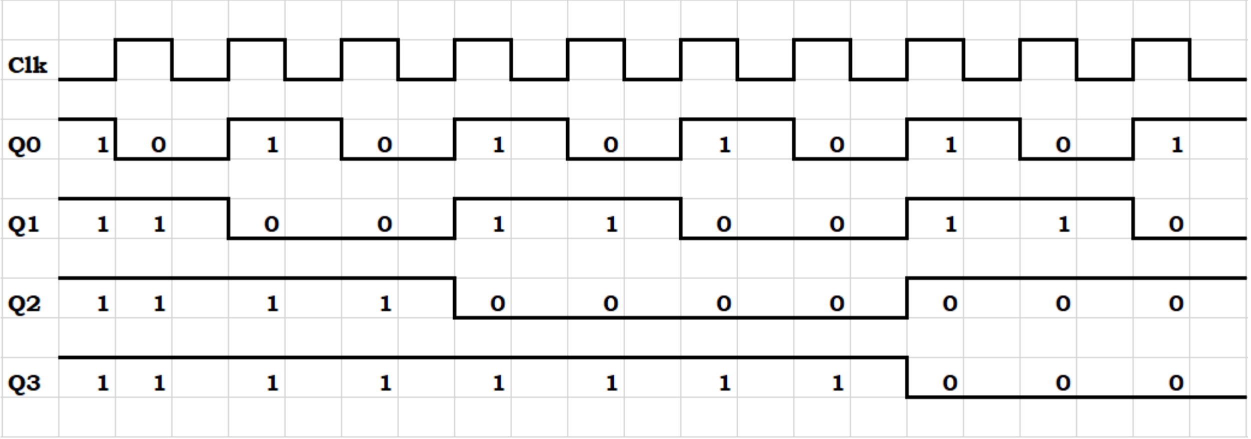


Figure 54: Timing diagram of 4-bit asynchronous binary ripple up counter

Change in the state of flip-flops occurs through the outputs of the previous stage flip-flops, that is the effect of count pulse must ripple through the counter. Hence the name ripple counter

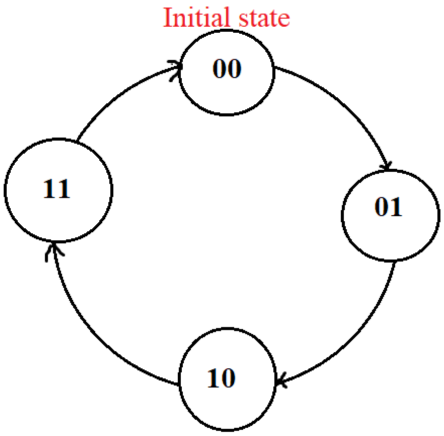
State diagram.

A graphical representation of the behavior of counters or any sequential circuits is called state diagram. State diagrams helps for the design of counters easily, which shows the transition from present states to the next states.

Procedure to draw the state diagram.

- Each state is represented by a circle, and the present state should be mentioned inside the circle.
- Use arrow associated lines to show the transition from present state to the next state. NOTE: if the present state and next state is same, then connect the arrow associated line to the same state.

Example: Two bit up-counter, four states counting from 00 to 11.



State table

The information contained in the state diagram is tabulated in a tabular form is called state table also called state synthesis table.

Example: Two-bit counter

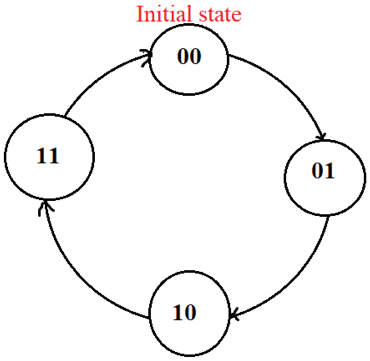
Present states		**Next states**	
Q_1	**Q_0**	**Q_1^+**	**Q_0^+**
0	0	0	1
0	1	1	0
1	0	1	1
1	1	0	0

Procedure to design synchronous counters.

1. Identify the number of flip-flops required based on the number of bits of sequence.
2. Draw the state diagram
3. Write the excitation table of the chosen flip-flop.
4. Obtain the excitation table for the complete circuit.
5. simplify the excitation equations using K-map.
6. Draw the logic diagram.
7. Verify the logic diagram by function table and timing diagram.

In the following section, the design of two-bit synchronous counter is discussed using T flip-flop.

1. No. of flip-flops is required =2, two-bit counter – sequence is 00, 01, 10 and 11.
2. State diagram



3. Excitation table of T-flip-flop

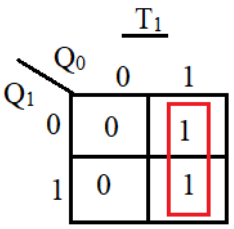
States		**Excitations**
Q	**Q^+**	**T**
0	0	0
0	1	1

1	0	1
1	1	0

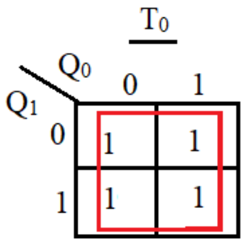
4. Excitation table for complete circuit

Present states		Next states		Excitations	
Q_1	Q_0	Q_1^+	Q_0^+	T_1	T_0
0	0	0	1	0	1
0	1	1	0	1	1
1	0	1	1	0	1
1	1	0	0	1	1

5. K-map simplification

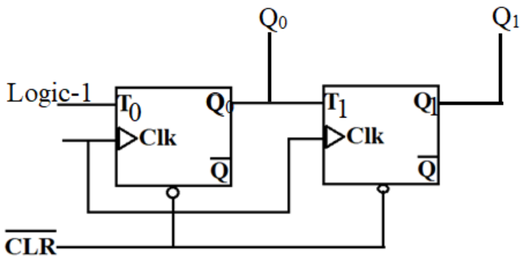


$T_1 = Q_0$



$T_0 = 1$

6. Logic diagram



7. Verification

Function table

Clock Pulse	Outputs	
	Q_1	Q_0
0	0	0
1	0	1
2	1	0
3	1	1
4	0	0

Timing diagram

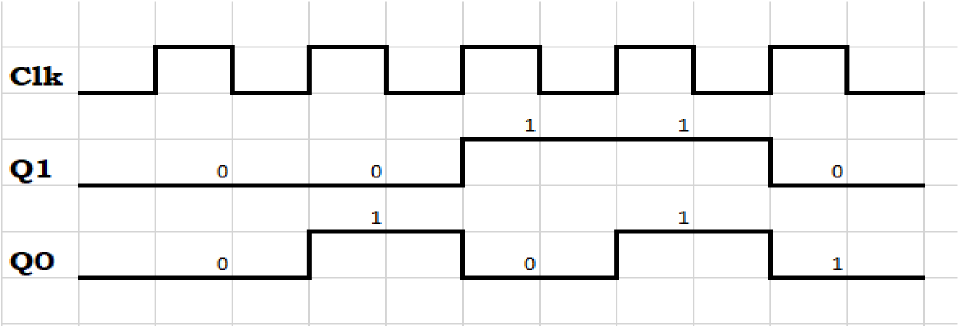


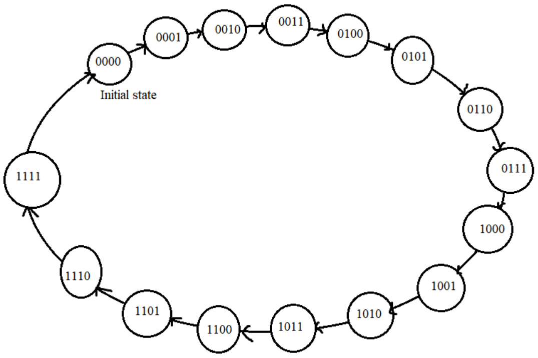
Figure: Timing diagram of two-bit counter

Similarly, counters can be designed using any type of flip-flops, depending on the counting sequence, counters are further classified into three types.

7. Up-counter and
   Counts the clock pulse incrementally.
8. Down-counter
   Counts the clock pulse in decrement order.
9. Ring counter
   Outputs the specified sequence in circular/ring format.

1. 4-bit synchronous up-counter using T flip-flops.

1. No. of Flip-flops required=04.
2. State diagram



3. Excitation table of T-flip-flop

States		Excitations
Q	Q^+	T
0	0	0
0	1	1
1	0	1
1	1	0

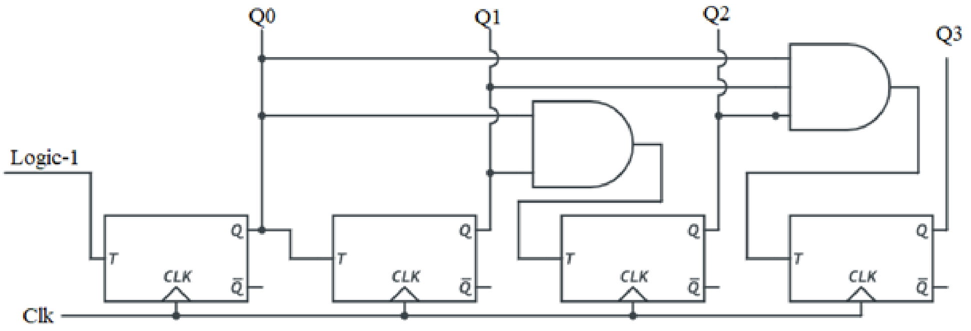
4. Excitation table of complete circuit

Present states				Next states				Excitations			
Q3	**Q2**	**Q1**	**Q0**	**Q3+**	**Q2+**	**Q1+**	**Q0+**	**T3**	**T2**	**T1**	**T0**
0	0	0	0	0	0	0	1	0	0	0	1
0	0	0	1	0	0	1	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	1
0	0	1	1	0	1	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	1
0	1	0	1	0	1	1	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	1
0	1	1	1	1	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	1
1	0	0	1	1	0	1	0	0	0	1	1
1	0	1	0	1	0	1	1	0	0	0	1
1	0	1	1	1	1	0	0	0	1	1	1
1	1	0	0	1	1	0	1	0	0	0	1
1	1	0	1	1	1	1	0	0	0	1	1
1	1	1	0	1	1	1	1	0	0	0	1
1	1	1	1	0	0	0	0	1	1	1	1

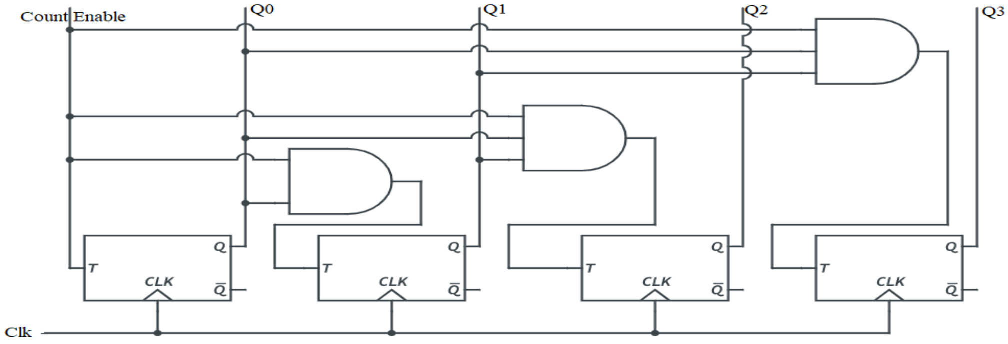
5. K-map Simplification

$$T_3 = Q_2Q_1Q_0, T_2 = Q_1Q_0, T_1 = Q_0, T_0 = 1$$

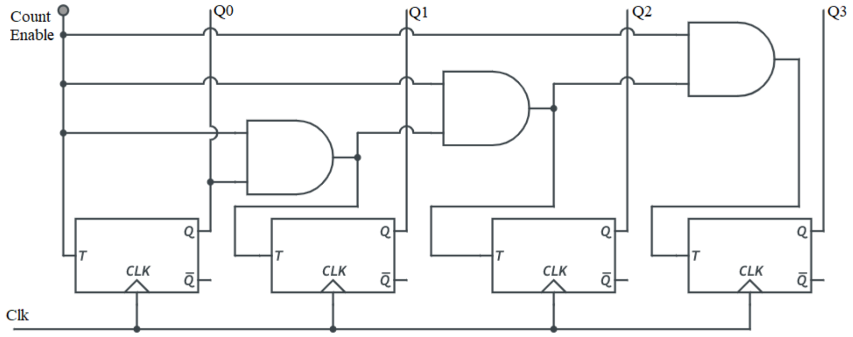
6. Logic Diagram



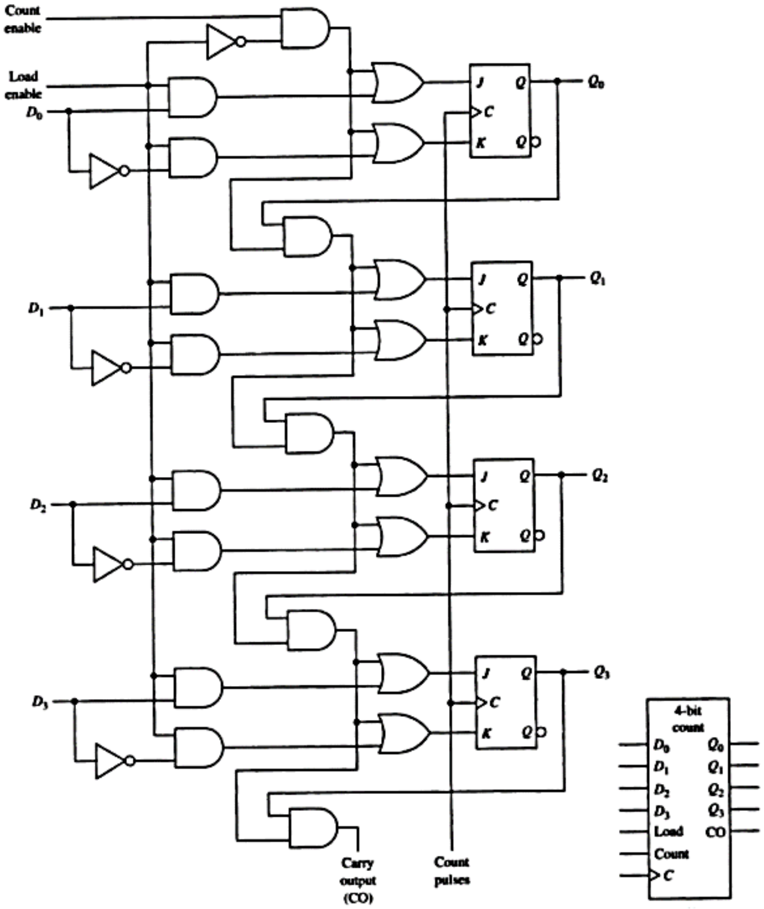
a. NOTE: 4-bit synchronous counter with count enable input



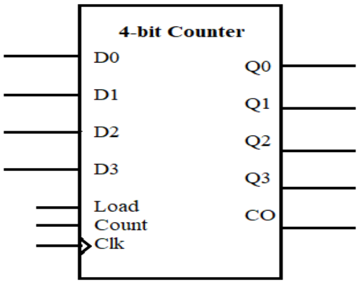
b. NOTE: 4-bit synchronous counter variation (design using only two input AND gates)



c. 4-bit synchronous counter with parallel loading using JK flip-flops

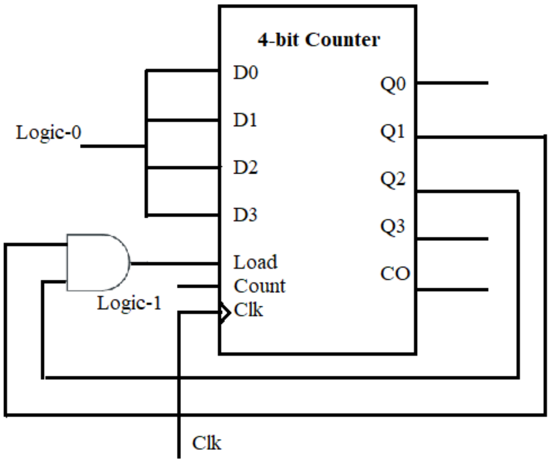


Logic Symbol

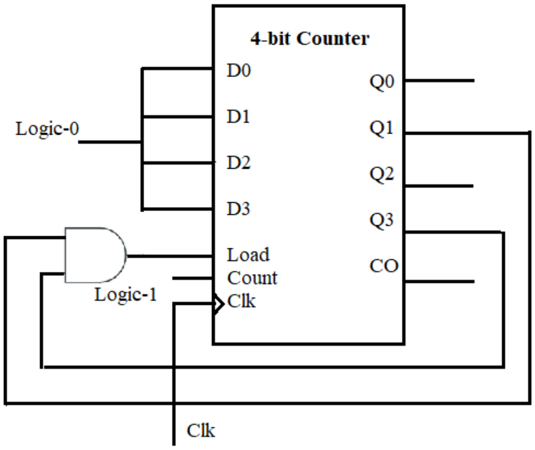


2. MOD counters with parallel loading

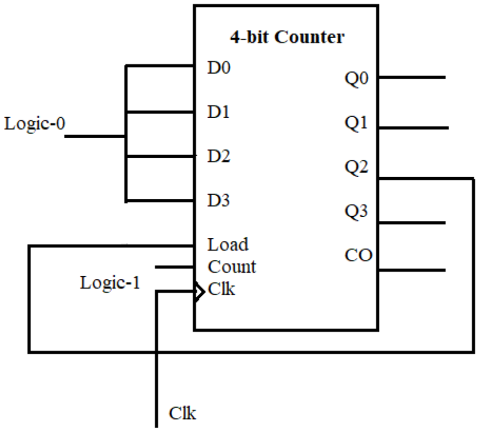
Example-1: Synchronous Mod-6 counter with parallel loading



Example-2: MOD-10 Counter



Example-3: MOD-4 Counter



3. Realization of 8-bit synchronous counter using two 4-bit counters

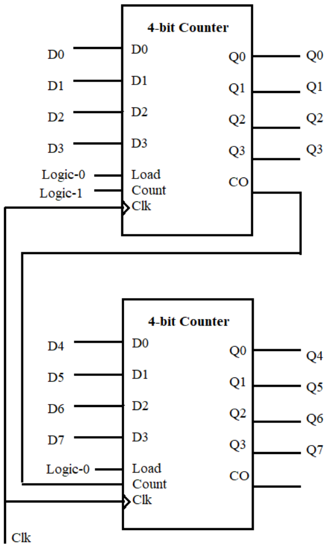


Figure : 8-bit synchronous counter

Counters based on shift registers

registers can also be used to generate the sequences repeatedly and acts as the counters, there are two types of counters Shift based on shift registers.

1. Ring Counter

The mod of ring counter is n, where n is the number of flip-flops are used in the design.

In the following section discusses the 4-bit ring counter using four D flip-flops shown in figure (). The previous stage outputs of flip-flops serve as the input to next stage flip-flops and last stage flip-flop output serves as the input of first stage flip-flop, which shifts the contents of flip-flops position wise and also in circular.

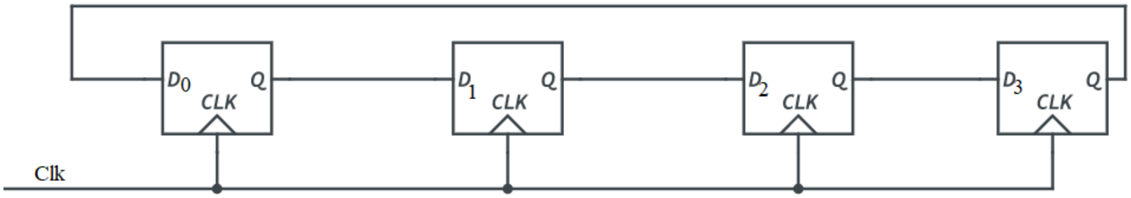


Figure: 4-bit ring counter

Truth table

Clk	Q0	Q1	Q2	Q3
0	1	0	0	0
1	0	1	0	0
2	0	0	1	0
3	0	0	0	1
4	1	0	0	0

Initially, the contents of flip-flops (Q0Q1Q2Q3) are assumed as (1000), at every rising edge of the clock pulse, the content of previous stage flip-flop will be shifted to the next stage flip-flop. The ring counter generates the four sequence and the sequence repeats after every four clock pulses, hence the mod of this counter is four.

Timing diagram

Figure shows the timing diagram of 4-bit ring counter.

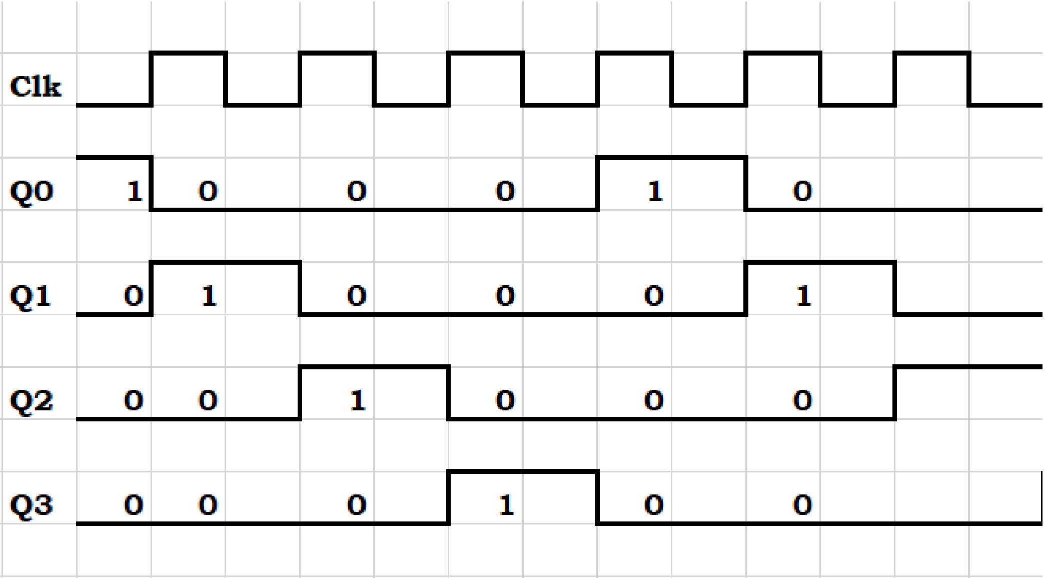


Figure: Timing diagram of 4-bit ring counter.

2. Twisted ring counter

The twisted ring counter is the modification of ring counter; complemented output of last stage serves as the input to the first stage flip-flop. The modulus of the twisted ring counter is 2n, where n is the number of flip-flops are used in the design. Figure () shows the logic diagram of 4-bit twisted ring counter using D flip-flops.

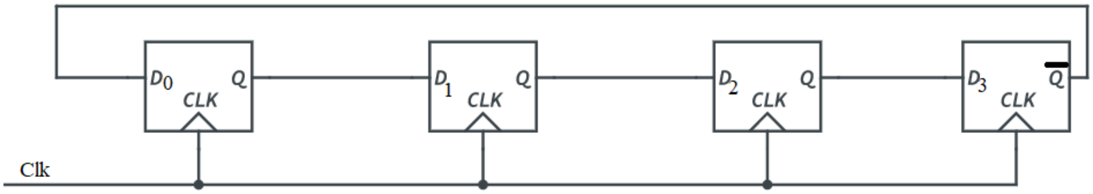


Figure : 4-bit twisted ring counter

Initially, the contents of flip-flops (Q0Q1Q2Q3) are assumed as (0000), at the first rising edge of the clock pulse $D_0 = \overline{Q_3}$ and hence, $Q_0 = D_0, Q_1 = D_1, Q_2 = D_2, Q_3 = D_3$ this data shift position wise at the flip-flops at every rising edge of the clock pulse, The twisted ring counter generates the eight sequence and the sequence repeats after every eight clock pulses, hence the mod of this counter is eight. The twisted ring counter is also called Johnson counter and switch tail counter.

Truth Table:

Clk	Q0	Q1	Q2	Q3
0	0	0	0	0
1	1	0	0	0
2	1	1	0	0
3	1	1	1	0
4	1	1	1	1
5	0	1	1	1
6	0	0	1	1
7	0	0	0	1
8	0	0	0	0

Timing diagram

Figure shows the timing diagram of Johnson counter.

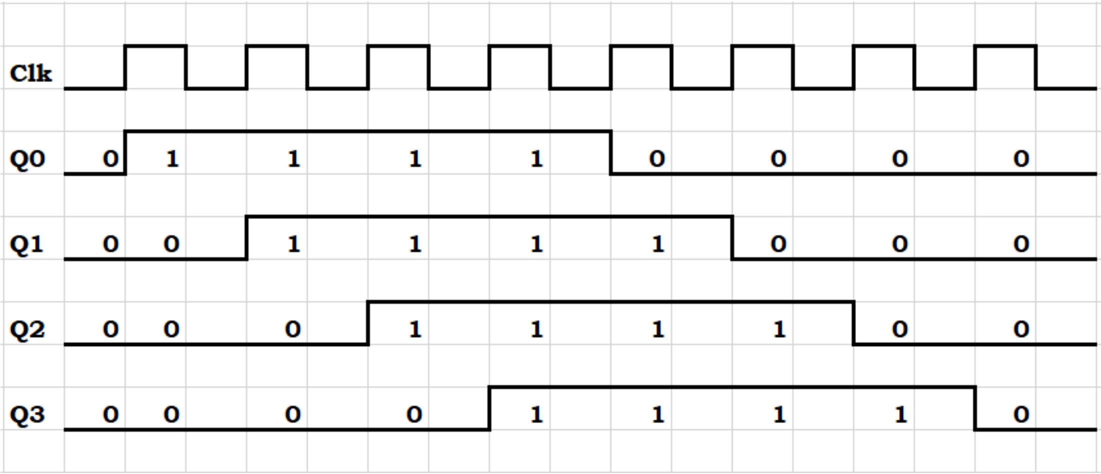


Figure : Timing diagram of twisted ring counter

.

******

NOTE: State machines, state table and state diagram concepts need to be added

UNIT-V : VHDL

I. Introduction to VHDL

VHDL, which stands for VHSIC Hardware Description Language.

VHSIC stands for Very High Speed Integrated Circuit.

VHDL is a Computer Aided Design (CAD) tool for the modern design and synthesis of digital systems.

Featurs of VHDL

- HDL is a very efficient tool for implementing and synthesizing designs on chips.
- It is a high level programming language similar to C.
- Debugging is easy, since HDL packages implement simulators and test-benches.
- HDL modules follow the general structure of software languages such as C.
- VHDL is a case insensitive language
- VHDL is a platform independent language.

II. History of VHDL.

VHDL was developed in the early 1980s under the VHSIC program at U.S. Department of Defence (DoD). Until the development of VHDL, each company used its own primitive hardware description languages, which are only gate level design tools and they did not support very large scale design.

To meet the need of designing large scale systems, a research team from three companies – IBM, Texas Instruments, and Intermetrics was jointly contributed at DoD for the development of powerful hardware description language based tools. The team produced the first publicly available standard, VHDL version 7.2, in 1985.

In 1986, the institute of electrical and electronics engineers (IEEE) was tasked with globally standardizing the language in 1987, the IEEE completed their mission and added several enhancements to the language, the result was the IEEE standard 1076-1987 version of VHDL, which was also recognized by the American National Standards Institute (ANSI).

In 1993, further updations were done with added features, the updated version is a 1076-1993 version of VHDL. One of the major enhancement is the addition of package std_logic_1164, which supports addition seven logic levels in addition to the existing two levels.

Nowadays VHDL is a very popular design tool among industry and academia. Additional tools have been added to the language, such as graphics based simulators that allow the user to interact graphically on the screen with simulator to compile, simulate, test and verify the design. Also an analog extension to the language is underway.

III. Structure of VHDL module.

VHDL module has two major constructs: entity and architecture. Entity describes the input and output signals of the system to be described. And is given a name or identifier by the user.

Architecture describes the functionality of the system to be designed. That is the relation between input and outputs of the system, and must be bound to an entity. The structure of any VHDL module/program as follows.

```
entity entity_name is
port(variable_1:mode data_type;
variable_2:mode data_type;
.
.
variable_n:mode data_type);

architecture architecture_name of entity_name is
signal declaration;
constant declaration;
component declaration;
configurations;
packages;
begin
hardware_description_Statement_1;
hardware_description_Statement_2;
.
.
hardware_description_Statement_n;
end architecture_name;
```

entity_name and architecture_name are identitifiers, these are used to name the module. Port declaration in entity is the listing of input and output variables of the system to be designed. Mode indicates the direction of input and output variables of the system. VHDL supports huge set of data types, data type of any variable indicates the type of value and range of values allowed by the corresponding variable. Architecture, consists of set of hardware description statements for the system to be designed. Every hardware description statements must be written after the begin statement. Signal declaration part is the list of intermediate signals of the system to be designed, component declaration section, lists the components to be used in the design. End statements must be written for completeness of the module or program.

IV. Port Modes

VHDL supports four mode types, which indicates the direction of the input and output variables, they are.

1. in : in is the port read mode. This mode is used for input signals, and these variables always appear on right hand side at the assignment statements.
2. out : out is the port write mode. This mode is used for output signals, and these variables always appear on left hand side at the assignment statements.
3. buffer : buffer is the port read as well as write mode with limited fan-out. These variables appear either on right hand side or left hand side at the assignment statements.
4. inout : inout is the port read as well as write mode without any constraints. These variables appear either on right hand side or left hand side at the assignment statements.

V. Data types

Data type of any element determines the variables allowed value with range. VHDL has a rich set of data types, some of them are discussed as follows.

1. Scalar data type:
   Scalar data type variables allows only numerical values. Scalar data type are further categorized into many types, they are discussed as follows.
   i) *bit* type : The only values allowed are either ‘0’ and ‘1’.
   ii) *boolean* type : The values allowed in this type is boolen type, true or false. Generally these boolean data type variables are used in the conditional constrcuts.
   Example:

```
a : in boolean;
b : in boolean;
big : out bit;
if (a>b)
big<=a;
else
big<=b;
```

iii) *integer* type : Allows all positive and negative values including zero. The range is from -2,147,483, 647 to +2,147,483,647. Integer data types can be used as *natural*, *positive* etc. natural data type allows all positive values including zero, positive data type only positive numbers.

Example:

```
a : in integer;
```

iv) *real* type : Allows fractional values, with range is -1.0E38 to 1.0E38.

v) *character* type : These variables under character type allows a single character or group of character (string).

vi) *physical* type : Allows the values which are measured in units. Like time in seconds, weight in grams, etc..

2. Composite type

Composite data type are used to declare a bus type variables. There are three types of composite data types, they are

i) *bit vector* : `port (a:in bit_vector (3 downto 0));`

ii) *array* type : collection of homogeneous elements.

```
port (a(7 downto 0):in integer);
```

iii) *record* type : group elements of different data types.

Example:

```
record
student_name: in character;
sl_no:in integer;
percentage:in real;
end record;
```

3. Other data-types

i) std_logic type: std_logic data-type variables take nine values, they are listed in the following table.

```
port (a: in std_logic);
```

std_logic data-type variables take nine values, they are listed in the following table.

U	Uninitiated.
X	Unknown
0	Low
1	High
Z	High impedance
W	Weak unknown
L	Weak low
H	Weak high
-	Don’t care

ii) std_logic_vector type: bus type std_logic data-type.

```
port(a:in std_logic_vector(7 downto 0);
```

iii) *signed* type : Numeric data type with MSB is sign bit.
iv) *unsigned* type : Numeric data type of positive value.

VI. Operators

The operators are used to perform operations on operands, general operations are logical, arithmetic, relational and shift operations, these are discussed as follows.

1. **Logic operators:** AND, OR, NOT, NAND, NOR, XOR and XNOR are logical operators which performs on two operands. In VHDL all these operators performs bitwise logical operations.
   Example:
   A=1010, B=1011, A and B=1010.

2. **Relational operators**: Table below summarized the types of relational operators.

Operator	Description	Operand type	Result type
=	Equality	Any type	Boolean
/=	Inequality	Any type	Boolean
<	Less than	Scalar	Boolean
<=	Less than or equal	Scalar	Boolean
>	Greater than	Scalar	Boolean
>=	Greater than or equal	Scalar	Boolean

3. **Arithmetic operators:** Table below summarized the different types of arithmetic operators.

Operator	Description	Operand type	Result type	Example
+	Addition	A-Numeric B-numeric	Numeric	A=5, B=3.5 A+B=8.5

-	Subtraction	A-Numeric
		B-numeric
			Numeric	A=5, B=3.5
				A-B=1.5

*	Multiplication	A-Integer or real B- Integer or real	Same as A	A=5, B=3 A*B=15
*	Multiplication	A-Physical B- Integer or real	Same as A	A=5 sec, B=3.5 A*B=17.5sec
*	Multiplication	A-Integer or real B- Physical	Same as B	A=5, B=3.5sec A+B=17.5sec

/	Division	A-Integer or real B- Integer or real	Same as A	A=5, B=2 A/B=2
/	Division	A-Physical B- Integer or real	Same as A	A=5sec, B=2.5 A/B=2sec
/	Division	A-Integer or real B- Physical	Same as B	A=5, B=2.5 sec A/B=2Hz.
mod	Modulus (Quotient)	A- only Integer B- only Integer	Integer	A=5, B=2 A mod B=1
rem	Remainder	A- only Integer B- only Integer	integer	A=5, B=2 A mod B=2
abs	Absoulute	Numerical	Positive numeric	A=-5 abs(A)=5

&	Concatenation	A- numerical or array
		B- numerical or array
			Same as A	A=”DN”
				B=’A’
				A & B=DNA

4. **Shift and rotate operators**

VHDL shift operators are unary operators and the operand must be a bit_vector type. These operators are summarized in the following table with examples.

Operator	**Operation example**	**Description**	**Operand value Before shifting**	**Operand value after shifting**
sll Shift left logically	A sll 1	Shift A one position left	A=1110	A=1100
srl Shift right logically	A srl 1	Shift A one position right	A=1110	A=0111
sla Shift left arithmetically NOTE: for signed numbers	A s1a 1	Shift A one position left arithmetically	A=1111	A=1110
sra Shift left arithmetically	A sra 1	Shift A one position right arithmetically	A=1111	A=0111

NOTE: for signed numbers				
rol rotate left	A rol 1	rotate A one position left arithmetically	A=1001	A=0011
ror rotate right	A ror 1	rotate A one position right arithmetically	A=1001	A=1100

VII. Styles of VHDL module descriptions

1. **Behavioral modeling/modeling**
   Describes how the output behaves with the inputs. In behavioral modeling the statements are executed sequentially, i.e., all statements are to be written inside the process block. Process is the keyword in VHDL, the statements under this block are executed sequentally.

   Example:

```
Entity of half_adder is
Port (A, B: in std_logic; S, C:out std_logic);
End half_adder;
Architecture behavioural of half_adder is
Begin
Process(A, B)
S<=A xor B;
C<=A and B;
End process;
End behavioural;
```

2. **Structural description/modeling**
   The hardware will be described using components or gates through the keyword “component”

   Example:

```
Entity of half_adder is
Port (A, B: in std_logic; S, C:out std_logic);
End half_adder;
Architecture structural of half_adder is
Component xor2
Pot(I1, I2: in bit; O1: out bit);
End xor2;
```

```
Component and2
Pot(I1, I2: in bit; O1: out bit);
End and2;
Begin
Xor2 portmap(A, B, S);
And2 portmap(A, B, C);
End structural;
```

3. **Data-flow description/modeling**

   The statements of the data-flow description are executed concurrently.

   Example:

```
Entity of half_adder is
Port (A, B: in std_logic; S, C:out std_logic);
End half_adder;
Architecture dataflow of half_adder is
Begin
S<=A xor B;
C<=A and B;
End behavioural;
```

4. **Switch level description/modeling**

   The system is described using switches or transistors. VHDL does not have built in switch level primitives, but can be constructed using packages.

   Example:

```
Entity of inverter is
Port (x: in std_logic; y:out std_logic);
End inverter;
Architecture switch_level of inverter is
Component nmos
Pot(I1, I2: in bit; O1: out bit);
End nmos;
Component pmos
Pot(I1, I2: in bit; O1: out bit);
End pmos;
For all pmos use entity work.mos(pmos_behavioural);
For all nmos use entity work.mos(pmos_behavioural);
Constant vdd: std_logic:='1';
Constant gnd: std_logic:='0';
Begin
P1=pmos portmap(y, vdd, x);
N1=(nmos portmap(y, gnd, x);
End switch_level;
```

5. **Mixed level description/modeling**

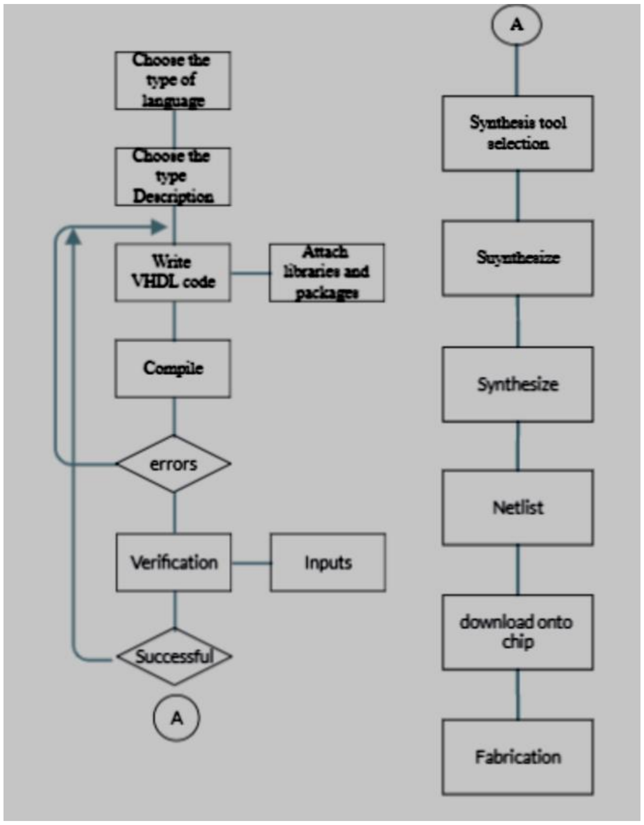
   Combination of two or more type of descriptions. i.e., description can be written using more than one type of modeling.

   Example: Combination of behavioral and data flow modeling.

```
Entity of example is
Port (A, B: in integer; x, y, z:out integer);
End example;
Architecture mixed_model of example is
X<=B+A
Begin
Process(A, B)
Y<=A - B;
z<=A * B;
End process;
End mixed_model;
```

VIII. Simulation and synthesis

Simulation is the process of verifying the functionality of the design. Synthesis is the process of compilation or generating netlist and map into implementation technology. The complete synthesis and simulation process are shown in the following flowchart.



IX. Programming Examples

VHDL Code for a Half-Adder

```
VHDL Code:

Library ieee;
use ieee.std_logic_1164.all;

entity half_adder is
   port(a,b:in bit; sum,carry:out bit);
end half_adder;

architecture data of half_adder is
begin
   sum<= a xor b;
   carry <= a and b;
end data;
```

VHDL Code for a Full Adder

```
Library ieee;
use ieee.std_logic_1164.all;

entity full_adder is port(a,b,c:in bit; sum,carry:out bit);
end full_adder;

architecture data of full_adder is
begin
   sum<= a xor b xor c;
   carry <= ((a and b) or (b and c) or (a and c));
end data;
```

VHDL Code for a Half-Subtractor

```
Library ieee;
use ieee.std_logic_1164.all;

entity half_sub is
   port(a,c:in bit; d,b:out bit);
end half_sub;

architecture data of half_sub is
begin
   d<= a xor c;
   b<= (a and (not c));
end data;
```

VHDL Code for a Full Subtractor

```
Library ieee;
use ieee.std_logic_1164.all;

entity full_sub is
  port(a,b,c:in bit; sub,borrow:out bit);
end full_sub;

architecture data of full_sub is
begin
  sub<= a xor b xor c;
  borrow <= ((b xor c) and (not a)) or (b and c);
end data;
```

VHDL Code for a Multiplexer

```
Library ieee;
use ieee.std_logic_1164.all;

entity mux is
  port(S1,S0,D0,D1,D2,D3:in bit; Y:out bit);
end mux;

architecture data of mux is
begin
  Y<= (not S0 and not S1 and D0) or
     (S0 and not S1 and D1) or
     (not S0 and S1 and D2) or
     (S0 and S1 and D3);
end data;
```

VHDL Code for a Demultiplexer

```
Library ieee;
use ieee.std_logic_1164.all;

entity demux is
  port(S1,S0,D:in bit; Y0,Y1,Y2,Y3:out bit);
end demux;

architecture data of demux is
begin
  Y0<=  ((Not S0) and (Not S1) and D);
  Y1<=  ((Not S0) and S1 and D);
  Y2<=  (S0 and (Not S1) and D);
  Y3<=  (S0 and S1 and D);
end data;
```

VHDL Code for a 8 x 3 Encoder

```
library ieee;
use ieee.std_logic_1164.all;

entity enc is
   port(i0,i1,i2,i3,i4,i5,i6,i7:in bit; o0,o1,o2: out bit);
end enc;
 architecture vcgandhi of enc is
begin
   o0<=i4 or i5 or i6 or i7;
   o1<=i2 or i3 or i6 or i7;
   o2<=i1 or i3 or i5 or i7;
end vcgandhi;
```

VHDL Code for a 3 x 8 Decoder

```
library ieee;
use ieee.std_logic_1164.all;

entity dec is
   port(i0,i1,i2:in bit; o0,o1,o2,o3,o4,o5,o6,o7: out bit);
end dec;

architecture vcgandhi of dec is
begin
   o0<=(not i0) and (not i1) and (not i2);
   o1<=(not i0) and (not i1) and i2;
   o2<=(not i0) and i1 and (not i2);
   o3<=(not i0) and i1 and i2;
   o4<=i0 and (not i1) and (not i2);
   o5<=i0 and (not i1) and i2;
   o6<=i0 and i1 and (not i2);
   o7<=i0 and i1 and i2;
end vcgandhi;
```

VHDL Code for an SR Latch

```
library ieee;
use ieee.std_logic_1164.all;

entity srl is
   port(r,s:in bit; q,qbar:buffer bit);
end srl;

architecture virat of srl is
   signal s1,r1:bit;
begin
   q<= s nand qbar;
   qbar<= r nand q;
end virat;
```

VHDL Code for an SR Flip Flop

```
library ieee;
use ieee.std_logic_1164.all;

entity srflip is
  port(r,s,clk:in bit; q,qbar:buffer bit);
end srflip;

architecture virat of srflip is
  signal s1,r1:bit;
begin
  s1<=s nand clk;
  r1<=r nand clk;
  q<= s1 nand qbar;
  qbar<= r1 nand q;
end virat;
```

VHDL code for the problem statement:
A system accepts three inputs and generates output as logic ‘1’ if the decimal equivalent of input combination is odd number else output is logic ‘0’.

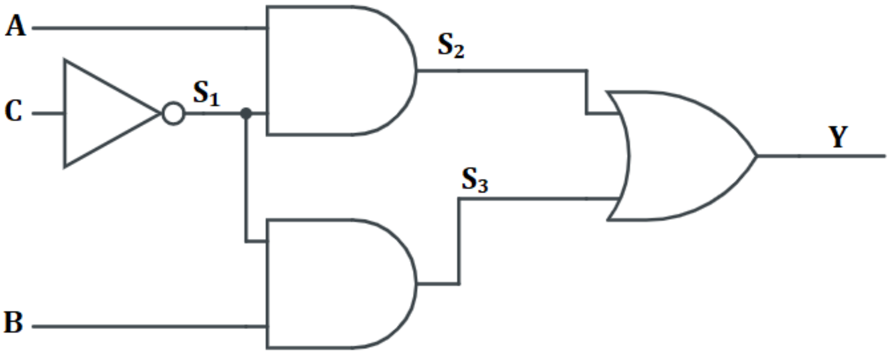
Step-1: Obtain the truth table

Inputs			Output
A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	0

Step-2: Boolean expression (use K-map simplification)

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

Step-3: Logic diagram



Step-4: VHDL Code

```
Entity combinational_circuit is
Port(A, B, C: in bit; Y: out bit);
End combinational_circuit;

Architecture data_flow of combinational_circuit is
Signal s1,s2,s3: inout bit;
Begin
S1<=not C;
S2<=A and S1;
S3<=B and S1;
Y<=S2 and S3;
End data_flow;
```

******

References

1. John M Yarbrough - Digital Logic Applications and Design, Thomson Learning, 2001. ISBN 981-240-062-1.
2. Donald D. Givone, —Digital Principles and Design||, McGraw Hill, 2002. ISBN 978-0-07-052906-9.
3. Nazeih M.Botros-John Weily India Pvt. Ltd. 2008. HDL Programming (VHDL and Verilog)
4. D. P. Kothari and J. S Dhillon, —Digital Circuits and Design‖, Pearson, 2016, ISBN:9789332543539.
5. Charles H Roth, Jr., —Fundamentals of Logic Design, Cengage Learning.
6. K. A. Navas, —Electronics Lab Manual‖, Volume I, PHI, 5th Edition, 2015

   ISBN:9788120351424.
7. https://www.tutorialspoint.com/vlsi_design/vhdl_programming_for_combinational_circuits.htm