

Module 4

Digital Electronics

Syllabus

Number Systems: Introduction, Number systems, Data representation, Binary arithmetic (addition, subtraction using complement form).

Logic gates & Boolean simplification: Boolean algebra, De-Morgan's theorems, Logic Gates, Universal gates.

Combinational Circuits: Half Adder, Full Adder, Multiplexer, Encoders, Decoders.

Sequential Circuits: Introduction to latches and flip flops.

4. Digital systems

Digital systems such as computers, smartphones, calculator process information in digital form or binary form. Digital systems have only two discrete values, 0s and 1s, unlike analog systems which can take a continuous range of values.

Advantages of digital systems:

- 1) Digital signals can convey information with less noise, distortion, and interference.
- 2) Digital signal processing is more secure because digital information can be easily encrypted and compressed.
- 3) Digital systems are more accurate, and the probability of error occurrence can be reduced by employing error detection and correction codes, hence reliable.
- 4) Digital signals can be easily stored on any magnetic media or optical media using semiconductor chips and can be implemented in the form of ICs.

4.1.1 Binary numbers

A decimal number is of base 10.

Binary numbers are numbers expressed in a base-2 number system and use only 2 symbols: “0” and “1”. Each digit is referred to as a bit.

Eg.(1): Decimal equivalent of 11010 =

$$(1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) = 16 + 8 + 2 = 26$$

Eg.(2): Decimal equivalent of 11010.11 =

$$(1 \times 2^4) + (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (1 \times 2^{-1}) + (1 \times 2^{-2}) = 26.75$$

Decimal numbers 1 to 15 can be represented in 4-bit binary as follows:

Decimal	Binary	Decimal	Binary
0	0000	8	1000
1	0001	9	1001
2	0010	10	1010
3	0011	11	1011
4	0100	12	1100
5	0101	13	1101
6	0110	14	1110
7	0111	15	1111

Hexadecimal systems

A number system with base (or radix) sixteen is called a hexadecimal number system. In this system, 16 symbols are used to represent numbers mainly 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E, and F.

4.1.2 Number Base Conversions

Numbers can be converted from one system to the other in the following ways:

1) Any number system to decimal

A number expressed in base r can be converted to decimal equivalent by multiplying each coefficient with the corresponding power of r and adding. Number is separated into integer part and fractional part.

Example: a) Binary to decimal

Convert $(1010.011)_2$ to decimal.

$$\begin{aligned}(1010.011)_2 &= (1 \times 2^3) + (0 \times 2^2) + (1 \times 2^1) + (0 \times 2^0) + (0 \times 2^{-1}) + (1 \times 2^{-2}) + (1 \times 2^{-3}) \\ &= 8 + 2 + 0.25 + 0.125 \\ &= (10.375)_{10}\end{aligned}$$

2) Hexadecimal to binary or vice versa

As $2^4 = 16$, each hexadecimal digit corresponds to four binary digits. Conversion from binary to hexadecimal is done by partitioning binary numbers into groups of four bits starting from the binary point and proceeding to left or right.

Example:

1) Convert $(10110001101011.11110010)_2$ to hexadecimal.

$$(10\ 1100\ 0110\ 1011.1111\ 0010)_2 = (2C6B.F2)_{16}$$

2) $(306.D)_{16} = (0011\ 0000\ 0110.1101)_2$

3) Decimal to any other system

i) Decimal to binary

Convert $(13)_{10}$ to binary.

To convert to binary, divide by 2.

$$13 / 2, \quad Q = 6, \quad R = 1$$

$$6 / 2, \quad Q = 3, \quad R = 0$$

$$3 / 2, \quad Q = 1, \quad R = 1$$

$$1 / 2, \quad Q = 0, \quad R = 1$$

Stop when $Q = 0$.

So binary representation is 1101.

To convert the fractional part of decimal to binary, multiply by 2 and take the integer part alone.

Example: Convert $(0.65625)_{10}$ to binary.

$$0.65625 \times 2 = 1.31250$$

$$0.3125 \times 2 = 0.6250$$

$$0.625 \times 2 = 1.2500$$

$$0.25 \times 2 = 0.5000$$

$$0.5 \times 2 = 1.0000$$

Now it can be stopped. The binary representation is $(0.10101)_2$

Complements

Complements are used in digital computers to simplify subtraction operations and for logical manipulation. There are two types of complements for each base-r system:

- 1) radix complement or r 's complement
- 2) diminished radix complements or $(r-1)$'s complement

For binary numbers, it is 2's complement and 1's complement. For decimal numbers, it is 10's complement and 9's complement.

**Given a number N in base r having n digits,
 r 's complement of N is defined as $r^n - N$.**

$(r-1)$'s complement of N is defined as $(r^n - 1) - N$.

For example, consider a decimal system with base 10.

To find 10's complement of 7

$N = 7$, no. of digits $n = 1$ (7 is single digit no), r is base = 10

$$10\text{'s complement} = r^n - N = 10^1 - 7 = 10 - 7 = 3$$

$$10\text{'s complement of } 9 = 10^1 - 9 = 1$$

$$10\text{'s complement of } 5690 = 10^4 - 5690 = 4310$$

To find 9's complement of 546700,

$$(r^n - 1) - N = 10^6 - 1 - 546700 = 453299$$

For binary numbers, $r = 2$ and $r-1 = 1$

It can be seen that 1's complement of binary number is formed by changing 1's to 0's and 0's to 1's.

$$2\text{'s complement of } 1101 = 2^4 - 1101 = (16)_{10} - 1101 = 10000 - 1101 = 0011$$

It can be seen that 2's complement of binary number is formed by adding 1 to 1's complement.

In binary subtraction, subtracting B from A is equivalent to adding A to 2's complement of B. If the final carry is generated, then the result is positive, final carry is neglected. If the final carry is not produced, the result is negative. In that case, take its 2's complement to get the final result.

4.2 Boolean Algebra

Boolean algebra is a branch of mathematics that deals with operations on logical values with binary variables. The Boolean variables are represented as binary numbers to represent truths: 1 = true and 0 = false.

Basic definitions:

- Boolean algebra

It is a set of elements, a set of operators, and many unproved axioms or postulates.

- Set of elements is any collection of objects, usually having a common property.

Example: $A = \{1, 2, 3, 4\}$ indicates that set A has the elements of 1, 2, 3, and 4.

- A binary operator defined on a set S of elements is a rule that assigns, to each pair of elements from S, a unique element from S.

The most common postulates used to formulate various algebraic structures are:

1. Closure.

A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S.

2. Associative law.

A binary operator $*$ on a set S is said to be associative whenever $(x * y) * z = x * (y * z)$ for all $x, y, z \in S$.

3. Commutative law. A binary operator $*$ on a set S is said to be commutative whenever $x * y = y * x$ for all $x, y \in S$

4. Identity element. A set S is said to have an identity element with respect to a binary operation $*$ on S if there exists an element $e \in S$ with the property that $e * x = x * e = x$ for every $x \in S$

Example: The element 0 is an identity element with respect to the binary operator $+$ on the set of integers $I = \{c, -3, -2, -1, 0, 1, 2, 3, c\}$, since $x + 0 = 0 + x = x$ for any $x \in I$

The set of natural numbers, N , has no identity element, since 0 is excluded from the set.

5. Inverse. A set S having the identity element e with respect to a binary operator $*$ is said to have an inverse whenever, for every $x \in S$, there exists an element $y \in S$ such that $x * y = e$

Example: In the set of integers, I , and the operator $+$, with $e = 0$, the inverse of an element a is $(-a)$, since $a + (-a) = 0$.

6. Distributive law. If $*$ and \bullet are two binary operators on a set S , $*$ is said to be distributive over \bullet whenever $x * (y \bullet z) = (x * y) \bullet (x * z)$.

Field:

A field is an example of an algebraic structure.

- The field of real numbers is the basis for arithmetic and ordinary algebra.
- The binary operator $+$ defines addition.
- The additive identity is 0.
- The additive inverse defines subtraction.
- The binary operator \bullet defines multiplication.
- The multiplicative identity is 1.
- For $a \neq 0$, the multiplicative inverse of $a = 1/a$ defines division (i.e., $a \bullet 1/a = 1$).
- The only distributive law applicable is that of \bullet over $+$:

$$a \bullet (b + c) = (a \bullet b) + (a \bullet c)$$

Axiomatic Definition of Boolean Algebra:

1854: George Boole developed an algebraic system called Boolean algebra.

1904: E. V. Huntington formulated a set of postulates that formally define the Boolean algebra.

1938: C. E. Shannon introduced a two-valued Boolean algebra called switching algebra that represented the properties of bistable electrical switching circuits.

Duality property

All binary operations remain valid when following two steps are performed:

- 1) Interchange OR and AND operators.
- 2) Replace all 1s by 0s and 0s by 1s.

Huntington postulates:

1. (a) The structure is closed with respect to the operator +.
 - (b) The structure is closed with respect to the operator •.
 2. (a) The element 0 is an identity element with respect to +; that is, $x + 0 = 0 + x = x$.
 - (b) The element 1 is an identity element with respect to •; that is, $x \cdot 1 = 1 \cdot x = x$.
 3. (a) The structure is commutative with respect to +; that is, $x + y = y + x$.
 - (b) The structure is commutative with respect to •; that is, $x \cdot y = y \cdot x$.
 4. (a) The operator • is distributive over +; that is, $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$.
 - (b) The operator + is distributive over •; that is, $x + (y \cdot z) = (x + y) \cdot (x + z)$.
 5. For every element $x \in B$, there exists an element $x' \in B$ (called the complement of x) such that (a) $x + x' = 1$ and (b) $x \cdot x' = 0$.
 6. There exist at least two elements $x, y \in B$ such that $x \neq y$.
- Comparing Boolean algebra with arithmetic and ordinary algebra
1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived (for both operators) from the other postulates.
 2. The distributive law of + over • (i.e., $x + (y \cdot z) = (x + y) \cdot (x + z)$) is valid for Boolean algebra, but not for ordinary algebra.
 3. Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
 4. Operator called complement is not available in ordinary algebra.

5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra is defined as a set with only two elements 0 and 1.

Two-valued Boolean Algebra

Two valued Boolean algebra is a set of two elements with operations:

$+$: OR operation; \cdot : AND operation, complement operator: NOT operation

Example: Binary logic is a two-valued Boolean algebra also called “switching algebra” by engineers

• $B = \{0,1\}$

• Operations

AND

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

OR

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

NOT

A	Y
0	1
1	0

- Closure: the result of each operation is either 1 or 0 and $1, 0 \in B$.
- Identity elements: 0 for $+$ and 1 for \cdot
- Commutative laws are obvious from truth tables.
- Distributive laws:
 $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$
 $x + (y \cdot z) = (x + y) \cdot (x + z)$

Complement

$$x + x' = 1: 0 + 0' = 0 + 1 = 1; 1 + 1' = 1 + 0 = 1$$

$$x \cdot x' = 0: 0 \cdot 0' = 0 \cdot 1 = 0; 1 \cdot 1' = 1 \cdot 0 = 0$$

4.2.1 Basic theorems and properties of Boolean Algebra

In Duality, binary operators are interchanged; AND is replaced by OR. Identity elements are interchanged; 1 is replaced by 0.

Postulates of Boolean Algebra:

$$1a) x + 0 = x$$

$$1b) x \cdot 1 = x$$

$$2a) x + x' = 1$$

$$2b) x \cdot x' = 0$$

$$2c) x + 1 = 1$$

$$2d) x \cdot 0 = 0$$

Commutative:

$$3a) x + y = y + x$$

$$3b) xy = yx$$

Distributive:

$$4a) x (y + z) = xy + xz$$

$$4b) x + yz = (x + y) \cdot (x + z)$$

Associative:

$$5a) x + (y + z) = (x + y) + z$$

$$5b) x (yz) = (xy) z$$

Theorems:

Theorem 1(a): $x + x = x$

Proof:

$$\begin{aligned} x+x &= (x+x) \cdot 1 && \text{by postulate 1b} \\ &= (x+x) (x+x') && 2a \\ &= x+xx' && 4b \\ &= x+0 && 2b \\ &= x && 1a \end{aligned}$$

Theorem 1(b): $x \cdot x = x$

Proof:

$$\begin{aligned} x \cdot x &= x x + 0 && \text{by postulate: 1(a)} \\ &= xx + xx' && 2(b) \\ &= x (x + x') && 4(a) \\ &= x \cdot 1 && 2(a) \\ &= x && 1(b) \end{aligned}$$

Theorem 1b) is the dual of theorem 1a)

Theorem 2(a): $x + 1 = 1$

$$\begin{aligned}
 x + 1 &= 1 \cdot (x + 1) && \text{by postulate: 1(b)} \\
 &= (x + x')(x + 1) && 2(a) \\
 &= x + x' \cdot 1 && 4(b) \\
 &= x + x' && 1(b) \\
 &= 1 && 2(a)
 \end{aligned}$$

Theorem 2(b): $x \cdot 0 = 0$ by duality

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

Complement of x' is x , so $(x')' = x$

Theorem 4a: $x + xy = x$

$$\begin{aligned}
 x + xy &= x \cdot 1 + xy && \text{by postulate: 1(b)} \\
 &= x (1 + y) && 4(a) \\
 &= x \cdot 1 && 2(c) \\
 &= x && 1(b)
 \end{aligned}$$

Theorem 4(b): $x (x + y) = x$ by duality

From truth table,

x	y	xy	x+xy
0	0	0	0
0	1	0	0
1	0	0	1
1	1	1	1

Theorem 5: DeMorgan's theorems

5a: $(x + y)' = x'y'$

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

Column 4 = Column 7

5b: $(x y)' = x' + y'$

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

Column 4 = Column 7

Operator precedence for evaluating Boolean expressions is

- a) Parenthesis b) not c) and d) or

4.2.2 Boolean functions:

Boolean function is an algebraic expression consisting of binary variables, binary operators OR and AND, unary operator NOT and parentheses

- It expresses the logical relationship between binary variables and is evaluated by determining the binary value of the expression for all possible values of the variables.

- Examples

1) $F1 = x + y' z$

It means $F1 = 1$ if $x = 1$ or if $y = 0$ and $z = 1$, otherwise $F1 = 0$.

2) $F2 = x' y' z + x' y z + x y'$

It means $F2 = 1$ if $(x = 0, y = 0, z = 1)$ or $(x = 0, y = 1, z = 1)$ or $(x = 1, y = 0)$, otherwise $F2 = 0$.

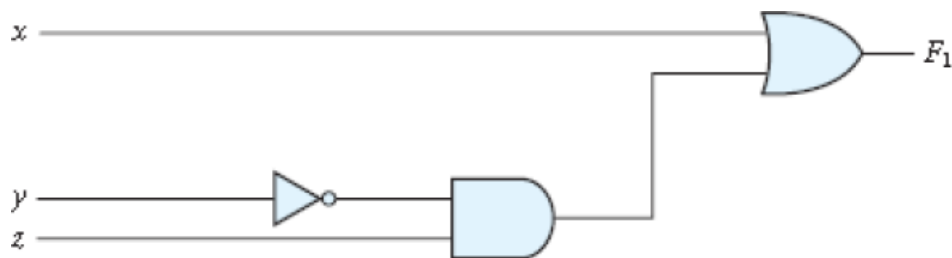
Truth Table

- Boolean function can be represented in a truth table.
- Truth table has 2^n rows where n is the number of variables in the function.
- The binary combinations for the truth table are obtained from the binary numbers by counting from 0 through $2^n - 1$.

Example: Truth table for F1 and F2 can be written as

x	y	z	F1	F2
0	0	0	0	0
0	0	1	1	1
0	1	0	0	0
0	1	1	0	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	0
1	1	1	1	0

Implementation of F1 with logic gates

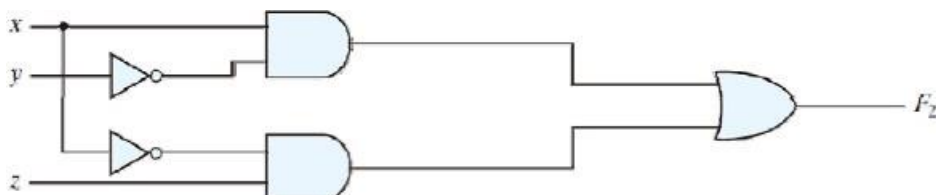


$$F1 = x + y'z$$

F2 can be simplified using the rules of Boolean Algebra.

$$\begin{aligned}
 F2 &= x'y'z + x'yz + xy' \\
 &= x'z(y' + y) + xy' \\
 &= x'z + xy'
 \end{aligned}$$

Implementation of F2 with logic gates



So Boolean expressions can be simplified using the rules of Boolean Algebra.

Literal: a complemented or un-complemented variable (an input to a gate).

Minimization of the number of literals results in a simple circuit with less number of gtes.

$F2 = x'y'z + x'yz + xy'$ has 3 terms with 8 literals. It can be simplified as:

$$F2 = x'z(y' + y) + xy'$$

$$= x'z + xy'$$

Now the simplified function has 2 terms and 4 literals only.

Minterm and maxterm:

Minterm:

Minterm (standard product): an AND term consisting of all literals in their normal form or in their complement form

- For example, two binary variables x and y, has 4 minterms: xy, xy', x'y, x'y'
- n variables can be combined to form 2^n minterms.

Maxterm (standard sum): an OR term consisting of all literals in their normal form or in their complement form.

- Each maxterm is the complement of its corresponding minterm, and vice versa.

x	y	z	Minterms		Maxterms	
			Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Any Boolean function can be expressed as sum of products or minterms (SOP form) or product of sums or maxterms (POS form).

SOP is expressed in the form of $\sum m$.

POS is expressed in the form of πM .

Example: $Y = A'B'C + ABC' + ABC$ is an SOP expression.

$Y = (A + B + C)(A' + B' + C')$ is a POS expression.

Conversion between SOP and POS

Let SOP form be $F(A,B,C) = \sum (1,4,5,6,7) = m_1 + m_4 + m_5 + m_6 + m_7$

POS form is complement of SOP form.

$$F'(A,B,C) = \pi M(0,2,3) = M0. M2. M3$$

Canonical and Standard forms

Canonical form is a method of representing Boolean outputs of digital circuits using Boolean Algebra in such a way that each term contains all the literals.

Standard form is a method of representing Boolean outputs of digital circuits using Boolean Algebra in such a way that there exists atleast one term that does not contain all the literals.

Example:

$Y = A'B'C + ABC' + AB'C$ is in canonical form.

$Y = A + BC$ is in standard form.

Other Logic Operations

For n binary variables, there can be 2^n functions.

Example for four variables, 16 functions are possible.

16 functions can be subdivided into various catagories:

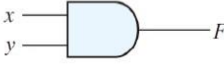
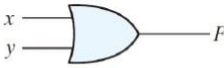
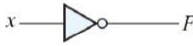
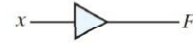
- a) Two functions that produce a constant 0 or 1.
- b) Four functions with unary operations: complement and transfer
- c) Ten functions with binary operators that define eight different operations: AND, OR, NAND, NOR, XNOR, equivalence, inhibition and implication.

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x , but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y , but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y , but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y , then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x , then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

- Complement function produces the complement of each binary variable.
- A function that is equal to the input variable has been given the name transfer.
- Equivalence is a function that is 1 when both binary variables are equal (XNOR).

4.2.3 Digital Logic Gates

Single/ dual input logic gates

Name	Graphic symbol	Algebraic function	Truth table															
AND		$F = x \cdot y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	0																
0	1	0																
1	0	0																
1	1	1																
OR		$F = x + y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	1
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	1																
Inverter		$F = x'$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>1</td></tr><tr><td>1</td><td>0</td></tr></table>	x	F	0	1	1	0									
x	F																	
0	1																	
1	0																	
Buffer		$F = x$	<table><tr><th>x</th><th>F</th></tr><tr><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td></tr></table>	x	F	0	0	1	1									
x	F																	
0	0																	
1	1																	


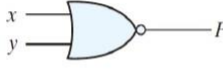
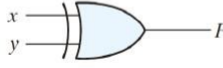
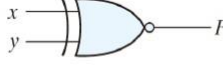
NAND		$F = (xy)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	1																
0	1	1																
1	0	1																
1	1	0																
NOR		$F = (x + y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	0
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	0																
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>0</td></tr><tr><td>0</td><td>1</td><td>1</td></tr><tr><td>1</td><td>0</td><td>1</td></tr><tr><td>1</td><td>1</td><td>0</td></tr></table>	x	y	F	0	0	0	0	1	1	1	0	1	1	1	0
x	y	F																
0	0	0																
0	1	1																
1	0	1																
1	1	0																
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	<table><tr><th>x</th><th>y</th><th>F</th></tr><tr><td>0</td><td>0</td><td>1</td></tr><tr><td>0</td><td>1</td><td>0</td></tr><tr><td>1</td><td>0</td><td>0</td></tr><tr><td>1</td><td>1</td><td>1</td></tr></table>	x	y	F	0	0	1	0	1	0	1	0	0	1	1	1
x	y	F																
0	0	1																
0	1	0																
1	0	0																
1	1	1																

Fig 4.1: Gates

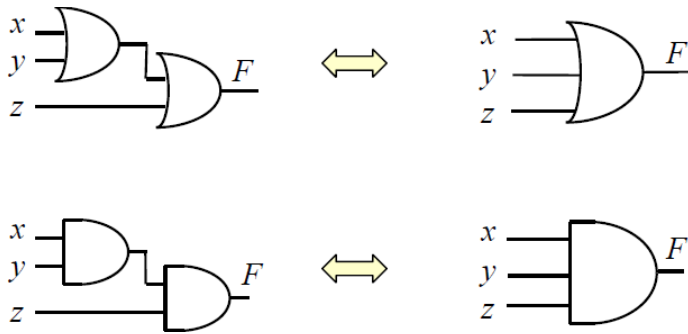
Multiple input gates

AND and OR are commutative and associative.

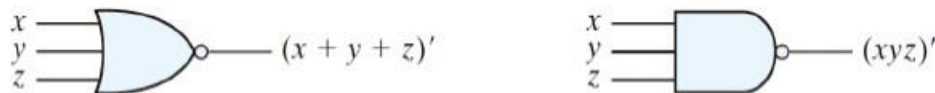
$$x + y = y + x \quad \text{and} \quad x \cdot y = y \cdot x$$

$$(x + y) + z = x + (y + z) \quad \text{and} \quad (x \cdot y) \cdot z = x \cdot (y \cdot z)$$

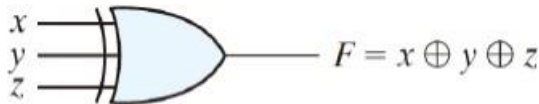
So 3 input AND and OR gates are as shown:



3-input NAND and NOR gate:



3-input XOR gate is



4.3 Combinational and sequential logic

Digital logic circuits are classified into two: Combinational and Sequential

Combinational logic circuits are memoryless logic circuits in which output at any instant of time depends only on the present inputs without regard to previous inputs.

Sequential circuits are circuits with memory, whose outputs depend not only on the present combination of inputs but also on past inputs.

4.3.1 Combinational logic circuit

A combinational circuit consists of input variables, logic gates and output variables.

Logic gates accept signals from inputs and generate output signals.

Both input and output data are represented by binary signals. For n input variables, there are 2^n possible combinations of binary input values. For each possible input combination, there is only one possible output combination. Each output function is expressed in terms of n input variables.

Each input variable to a combinational circuit may have one or two wires. Variable is represented either in normal form (unprimed) or in complemented form (primed). To implement primed variables, an inverter is required.

Design procedure of combinational circuits

Design procedure involves the following steps:

- 1) The problem is stated.
- 2) The number of available input variables and required output variables is determined.
- 3) The input and output variables are assigned letter symbols.
- 4) Truth table that defines the relationship between input and output variables is determined.
- 5) Simplified Boolean function for each output is obtained.
- 6) Logic diagram is drawn.

Truth table for a combinational circuit consists of input and output columns. 1s and 0s in input columns are obtained from 2^n binary combinations available for input variables. Output may be either 0 or 1 for every valid input combination.

Output Boolean functions from truth table are simplified using laws of Boolean Algebra, K-map method or tabular method.

Practically design method will have to consider constraints such as:

- a) Minimum number of gates
- b) Minimum number of inputs to a gate
- c) Minimum propagation time of signal through the circuit.
- d) Minimum number of interconnections
- e) Limitations of driving capabilities of each gate.

Logic diagram is helpful in visualising gate implementation of expressions.

Adders:

Digital systems like computers perform all arithmetic operations like addition, subtraction, multiplication and division. The most basic operation is binary addition.

4.3.2 Half adder

Half adder is a combinational circuit that performs the addition of two bits and produces sum S and carry C as the outputs.

- 1) Addition of 0 and 0 produces sum = 0, carry = 0.
- 2) Addition of 0 and 1 produces sum = 1, carry = 0.
- 3) Addition of 1 and 1 produces sum = 0, carry = 1.

Truth table

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

From truth table, $S = A'B + AB' = A \text{ xor } B$

$C = AB$

Logic circuit for half adder:

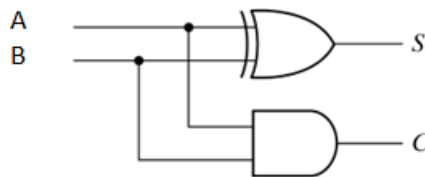


Fig 4.2 Implementation of half adder

4.3.3 Full adder:

Full adder is a combinational circuit that performs the addition of three bit: A, B and Cin and produces sum S and carry Cout as the outputs.

Truth table for Full adder:

A	B	Cin	S	Cout
0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

$$\begin{aligned}
 S &= A'B'Cin + A'BCin' + AB'Cin' + ABCin \\
 &= Cin (AB + A'B') + Cin' (A'B + AB') \\
 &= Cin (A \text{ xnor } B) + Cin' (A \text{ xor } B) \\
 &= A \text{ xor } B \text{ xor } Cin
 \end{aligned}$$

$$\begin{aligned}
 Cout &= A'BCin + AB'Cin + ABCin' + ABCin \\
 &= AB (Cin + Cin') + Cin (A \text{ xor } B) \\
 &= AB.1 + Cin (A \text{ xor } B) = AB + Cin (A \text{ xor } B)
 \end{aligned}$$

$$S = A \text{ xor } B \text{ xor } Cin$$

$$Cout = AB + Cin (A \text{ xor } B)$$

Implementation of Full adder with two half adders and OR gate.

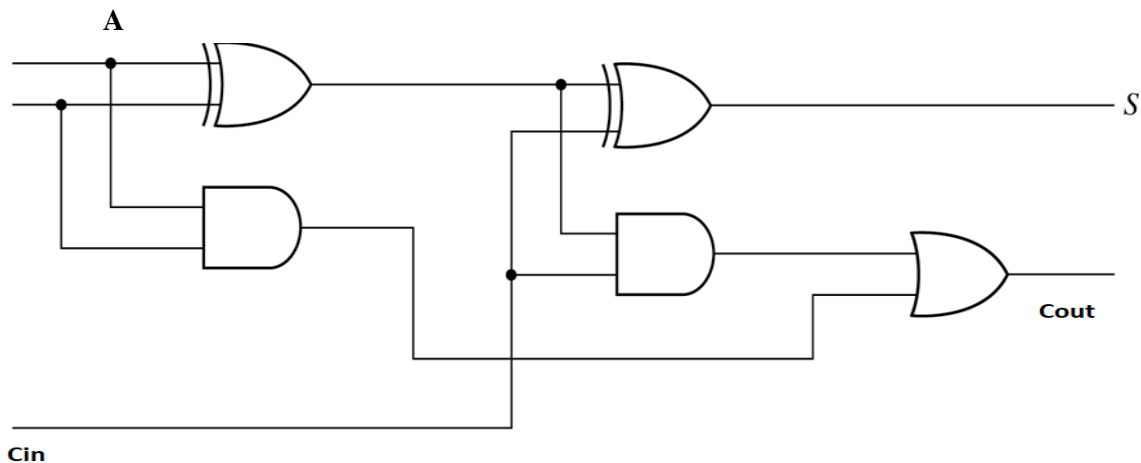


Fig 4.3 Implementation of Full Adder

4.3.4 Multiplexers

- Multiplexing means transmitting a large number of information units over a smaller number of channels or lines.
- A digital multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line.
- The selection of a particular input line is controlled by a set of selection lines.
- Normally, there are 2^n input lines and n selection lines whose bit combinations determine which input is selected.
- Multiplexers are used in communication systems to increase the efficiency of the system. Multiplexers are used in telephone networks for the integration of several audio signals on a single transmission line. To maintain large amounts of data, multiplexers are also used in computer memory systems.
- A **4-to-1-line multiplexer**. Each of the four input lines, I_0 to I_3 is applied to one input of an AND gate. Selection lines S_1 and S_0 are decoded to select a particular AND gate.

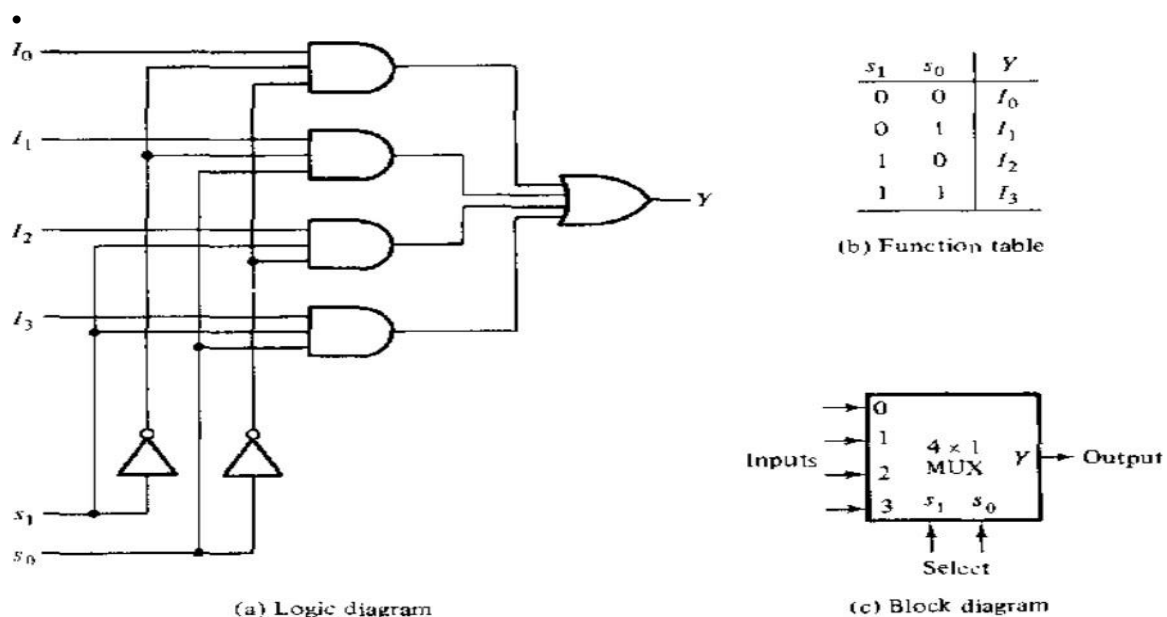


Fig 4.4 (a) Multiplexer logic diagram (b) Function table (c) Block diagram

4.3.5 Decoders

- A decoder is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines.
- If the n -bit decoded information has unused or don't-care combinations, the decoder output will have fewer than 2^n outputs.
- The decoders presented here are called n -to- m -line decoders, where $m \leq 2^n$.
- Their purpose is to generate the 2^n (or fewer) minterms of n input variables. The

name decoder is also used in conjunction with some code converters such as a BCD-to-seven segment decoder

- Decoders are used to convert an analog signal into digital data, which can then be processed by a computer.

3 to 8 Decoder

- A 3 to 8 decoder has three inputs (A, B, C) and eight outputs (D0 to D7).
- Based on the 3 inputs one of the eight outputs is selected.
- The truth table for 3 to 8 decoder is shown in the below table.
- From the truth table, it is seen that only one of eight outputs (D0 to D7) is selected based on three select inputs.
- From the truth table, the logic expressions for outputs can be written as follows:

A	B	C	D0	D1	D2	D3	D4	D5	D6	D7
0	0	0	1	0	0	0	0	0	0	0
0	0	1	0	1	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0	0
0	1	1	0	0	0	1	0	0	0	0
1	0	0	0	0	0	0	1	0	0	0
1	0	1	0	0	0	0	0	1	0	0
1	1	0	0	0	0	0	0	0	1	0
1	1	1	0	0	0	0	0	0	0	1

- Boolean equations for the output can be written as follows

$$D_0 = \bar{A}\bar{B}\bar{C}, \quad D_1 = \bar{A}\bar{B}C, \quad D_2 = \bar{A}B\bar{C},$$

$$D_3 = \bar{A}BC, \quad D_4 = A\bar{B}\bar{C}, \quad D_5 = A\bar{B}C,$$

$$D_6 = ABC, \quad D_7 = \bar{A}BC$$

Implementation of 3:8 decoder circuit

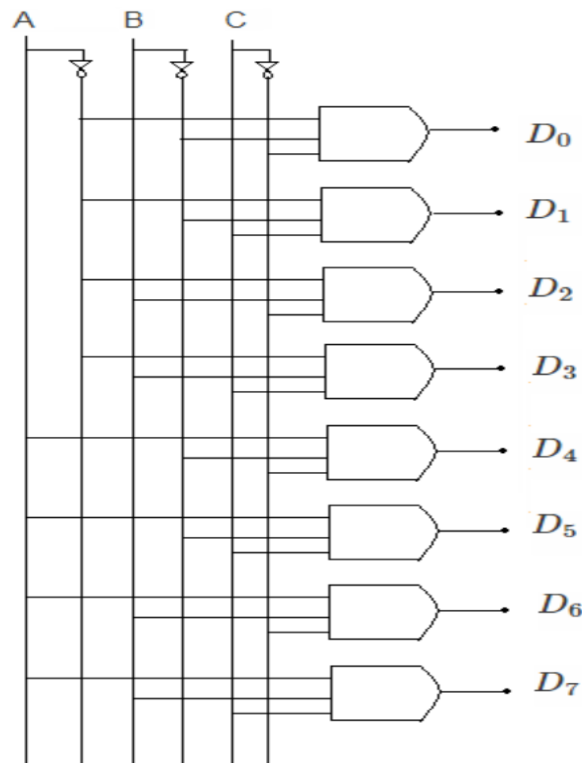


Fig 4.5 Implementation of 3:8 Decoder

4.3.6 Encoders

- An encoder is a digital circuit that performs the inverse operation of a decoder.
- An encoder has 2^n (or fewer) input lines and n output lines. The output lines generate the binary code corresponding to the input value.
- An encoder is a digital circuit that converts a set of binary inputs into a unique binary code.
- The binary code represents the position of the input and is used to identify the specific input that is active.
- Encoders are commonly used in digital systems to convert a parallel set of inputs into a serial code.

Octal to Binary Encoders

- The basic principle of an encoder is to assign a unique binary code to each possible input. For example, a 2-to-4-line encoder has 2 input lines and 4 output lines and assigns a unique 4-bit binary code to each of the $2^2 = 4$ possible input combinations.
- The output of an encoder is usually active low, meaning that only one output is active (low) at any given time, and the remaining outputs are inactive (high). The active low output is selected based on the binary code assigned to the active input.

- Truth table of 8:3 Octal to Binary Encoders is written as follows :

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	x	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Fig 4.6 Truth table of 8:3 Encoder

Implementation of 8:3 Octal to Binary Encoders is written as follows

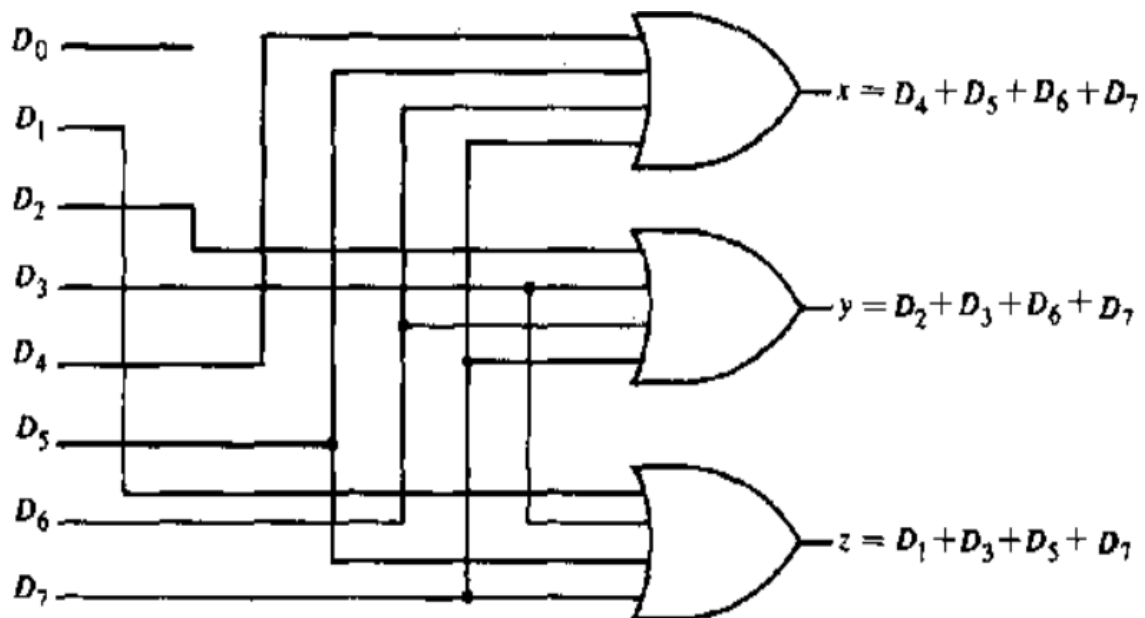


Fig 4.7 Implementation of 8:3 Encoder

Differences between Combinational and Sequential Circuits

Metrics /Parameters	Combinational Circuit	Sequential Circuit
Usage of memory or Storage area	It does not require any memory element to store the output.	It requires a memory element to store the previous state output.
Feedback	It requires no feedback for generating the next output	It requires feedback as it relies on the previous output/feedback and the current input.
Speed	It performs faster in comparison with sequential circuits.	Its performance is slow because it uses memory elements.
Elementary blocks	It uses logic gates as an elementary block.	It uses flipflops as elementary blocks.
Complexity	It is easy to use and handle	It is complex to use and handle in comparison to combinational circuits.
Example	Encoder ,Decoder, Multiplexer	Flip-flops and Counters.

4.4 Sequential Circuits

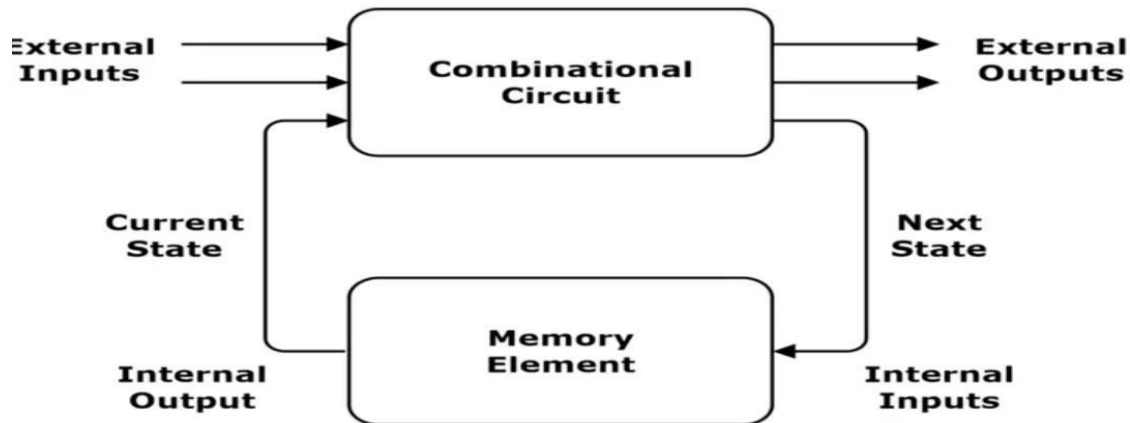


Fig 4.8 Block diagram of sequential circuits

Types of Sequential Circuits

- **Asynchronous sequential circuits:** They do not use the clock signals instead they make use of the pulses. The un-clocked flip-flops are the memory elements of asynchronous circuits. Though many differences between combinational and sequential circuits the asynchronous sequential circuits are similar to combinational circuits along with feedback.
- **Synchronous Sequential circuits:** They make use of clock signals and synchronize the memory element's states with the clock signals. The output is stored in the memory elements called latches or flip-flops (Bi-stable multivibrator).

Differences between Flip-flop and Latch

SNO	Flip-flop	Latch
1	Flip-flop is a bistable device i.e., it has two stable states that are represented as 0 and 1.	Latch is also a bistable device whose states are also represented as 0 and 1.
2	It checks the inputs but changes the output only at times defined by the clock signal or any other control signal.	It checks the inputs continuously and responds to the changes in inputs immediately.
3	It is a edge triggered device.	It is a level triggered device.
4	Gates like NOR, NOT, AND, NAND are building blocks of flip flops.	These are also made up of gates.
5	They are classified into asynchronous or synchronous	There is no such classification in latches.
6	It forms the building blocks of many sequential circuits like counters.	These can be used for the designing of sequential circuits but are not generally preferred.
7	a, Flip-flop always have a clock signal	Latches doesn't have a clock signal
8	Flip-flop can be build from Latches	Latches can be build from gates
9	ex:D Flip-flop, JK Flip-flop	ex:SR Latch, D Latch

4.4.1 SR flipflop Using NAND gates

The NAND gate SR flip flop is a basic flip flop which provides feedback from both of its outputs back to its opposing input. This circuit is used to store the single data bit in the memory circuit. So, the SR flip flop has a total of three inputs, i.e., 'S' and 'R', and current output 'Q'.

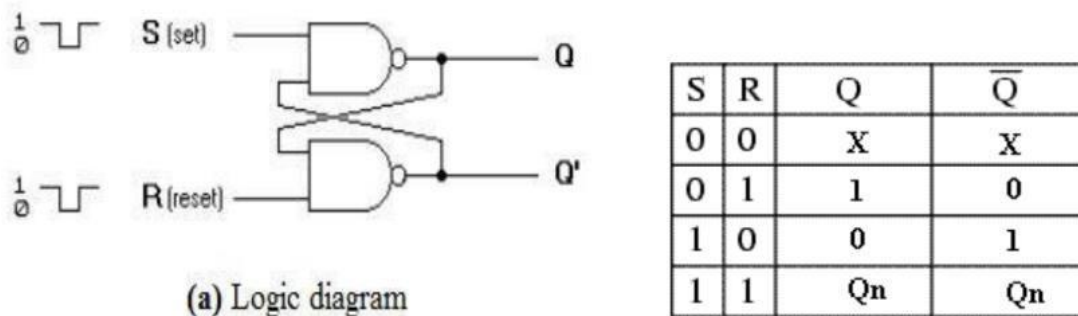


Fig 4.9 Logic diagram of SR flipflop using NAND gates

Note: If any one of the inputs is zero in NAND gate the output will be equal to one.

Case 1): when $S=0$ and $R=0$

- Input to first NAND gate is $\{0, Q'\}$ then the output $Q=1$
- Input to second NAND Gate is $\{Q, 0\} = \{1, 0\}$ then the output $Q'=1$

Here $Q=Q'=1$; Q and Q' Should be complement of each other, hence **$S=0$ and $R=0$ is a Invalid state.**

Case 2): When $S=0$ and $R=1$

- Input to first NAND gate is $\{0, Q'\}$ then the output $Q=1$
- Input to second NAND gate is $\{Q, 1\} = \{1, 1\}$ then the output $Q' = 0$

Here $Q=1$ and $Q'=0$; Q and Q' should be complement of each other , hence

$S=0$ and $R=1$ gives the output as $Q=1$

Case 3): When $S=1$ and $R=0$

- Input to second NAND gate is $\{Q, 0\}$ then the output $Q'=1$
- Input to first NAND gate is $\{1, Q'\} = \{1, 1\}$ then the output $Q = 0$

Here $Q=0$ and $Q'=1$; Q and Q' should be complement of each other , hence

$S=1$ and $R=0$ gives the output as $Q=0$.

Case 4): when $S=1$ and $R=1$

The output of SR latch cannot be predicted

Therefore

1. Assume $Q=1$ and $Q'=0$

- Input To First NAND gate $\{1, Q'\} = \{1, 0\}$, then the output $Q=1$
- Input to second NAND gate $\{Q, 1\} = \{1, 1\}$, then the output $Q'=0$

2. Assume $Q=0$ and $Q'=1$

- Input to first NAND gate $\{1, Q'\} = \{1, 1\}$, then the output $Q=0$
- Input to second NAND gate $\{Q, 1\} = \{0, 1\}$, then the output $Q'=1$

In both the cases predicted Q and Q' value is Equal to the Output Value of Q and Q' .

Therefore, when $R=1$ and $S=1$ there will be No change in State.

SR – Latch (Flip Flop Using NOR gate):

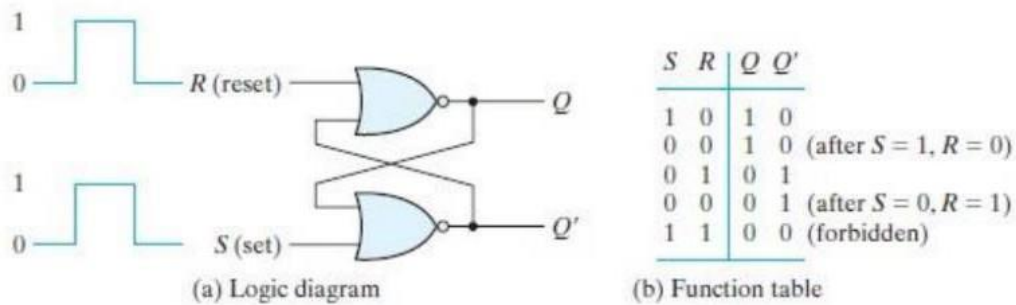


Fig 4.10 Logic diagram of SR flipflop using NOR gates

Note: If any one of the inputs is one in the NOR gate the output will be equal to zero.

Case 1): when S=0 and R=0

The output of SR latch cannot be predicted

Therefore

1) Assume Q=1 and Q'=0

- Input To First NOR gate $\{0, Q'\} = \{0, 0\}$, then the output Q=1
- Input to second NOR gate $\{Q, 0\} = \{1, 0\}$, then the output Q'=0

2) Assume Q=0 and Q'=1

- Input to first NOR gate $\{0, Q'\} = \{0, 1\}$, then the output Q=0
- Input to second NOR gate $\{Q, 0\} = \{0, 0\}$, then the output Q'=1

In both cases predicted Q and Q' value is Equal to the Output Value of Q and Q' . Therefore, when R=0 and S=0 there will be No change in State.

Case 2): When R=1 and S=0

- Input to first NOR gate is $\{1, Q'\}$ then the output Q=0
- Input to second NOR gate is $\{Q, 1\} = \{1, 1\}$ then the output Q' = 1

Here Q=0 and Q'=1; Q and Q' should be the complement of each other, hence S= 0 and R=1 give the output as Q=0

Case 3): When R=0 and S=1

- Input to second NOR gate is $\{Q, 1\}$ then the output Q'=0
- Input to first NOR gate is $\{1, Q'\} = \{1, 0\}$ then the output Q = 1

Here $Q=0$ and $Q'=1$; Q and Q' should be complement of each other , hence

$S=1$ and $R=0$ gives the output as $Q=1$.

Case 4): when $S=1$ and $R=1$

- Input to first NOR gate is $\{1, Q'\}$ then the output $Q=0$
- Input to second NOR Gate is $\{Q, 1\} = \{0, 1\}$ then the output $Q'=0$

Here $Q=Q'=0$; Q and Q' Should be complement of each other , hence $S=1$ and $R=1$ is a Invalid state.

RS FLIPFLOP WITH CLOCK PULSE

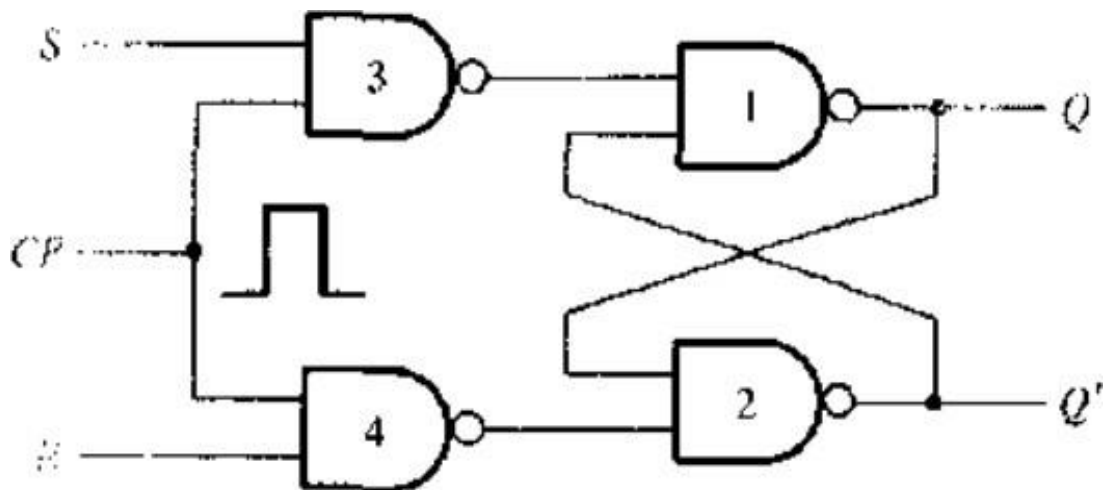


Fig 4.11 Logic diagram of RS flipflop with clock pulse

4.4.2 D flipflop

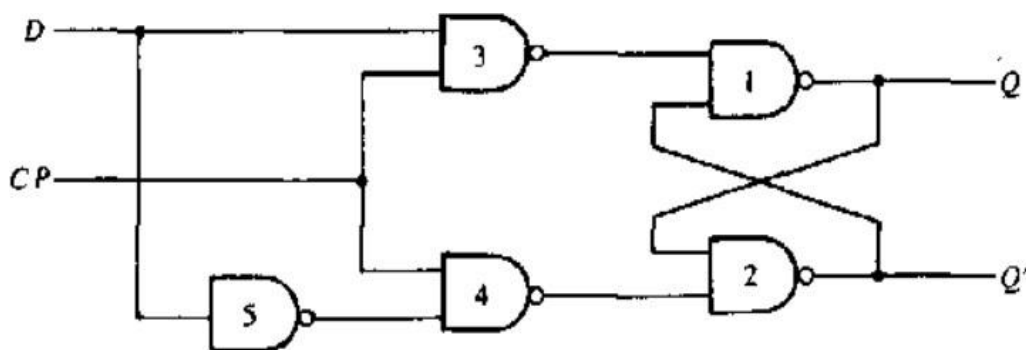


Fig 4.12 Logic diagram of D flipflop with clock pulse

- One way to eliminate the undesirable condition of the indeterminate state in the RS flipflop is to ensure that inputs S and R are never equal to 1 at the same time. This is done in the D flip-flop.
- The D input goes directly to the S input and its complement is applied to the R input. As long as the pulse input is at 0, the outputs of gates 3 and 4 are at the 1 level and the circuit cannot change state regardless of the value of D.
- The D input is sampled when CP = 1. If D is 1, the Q output goes to 1, placing the circuit in the set state. If D is 0, output Q goes to 0 and the circuit switches to the clear state.
- The D flip-flop receives the designation from its ability to hold data in its internal storage. This type of flip-flop is sometimes called a gated D-latch. The CP input is often given the designation G (for gate) to indicate that this input enables the gated latch to make possible data entry into the circuit.
- The binary information present at the data input of the D flip-flop is transferred to the Q output when the CP input is enabled. The output follows the data input as long as the pulse remains in its 1 state.
- When the pulse goes to 0, the binary information that was present at the data input at the time the pulse transition occurred is retained at the Q output until the pulse input is enabled again.
- The basic application of an FF is to store the data. Then it may be used to design registers for storing multi-bit data. This includes SISO (Serial in Serial Output), SIPO, PISO, PIPO, bidirectional shift registers, and universal shift registers.

References

1. M. Morris Mano. (2008). Digital Logic and Computer Design (3rd ed) PHI Learning.