Boolean algebra

First Class

2016-2017

Boolean algebra

Objective

- Understand the relationship between Boolean logic and digital computer circuits.
- Learn how to design simple logic circuits.
- Understand how digital circuits work together to form complex computer systems.

Boolean algebra

- Mathematician **George Boole** invented Boolean logic operations system in **1813 – 1864**. Boolean logic is also known as Boolean algebra. It is a mathematics of digital systems.

Boolean algebra

- Boolean algebra is a mathematical system for the manipulation of variables that can have one of two values.
  - In formal logic, these values are “true” and “false.”
  - In digital systems, these values are “on” and “off,” 1 and 0, or “high” and “low.”
- Boolean expressions are created by performing operations on Boolean variables.
  - Common Boolean operators include AND, OR, and NOT.

Boolean Algebra Operation

- The **complement** is denoted by a bar . It is defined by

$\overline{0} = 1$ and $\overline{1} = 0$.

- The **Boolean sum**, denoted by **(+)** or by **OR**, has the following values:

$$1 + 1 = 1, \quad 1 + 0 = 1, \quad 0 + 1 = 1, \quad 0 + 0 = 0$$

- The **Boolean product**, denoted by **(·)** or by **AND**, has the following values:

$$1 \cdot 1 = 1, \quad 1 \cdot 0 = 0, \quad 0 \cdot 1 = 0, \quad 0 \cdot 0 = 0$$

Truth Tables

Inputs		Outputs
x	*y*	*xy*
0	0	0
0	1	0
1	0	0
1	1	1

xy = x **AND** y = x * y
AND is true only if **both** inputs are true

Inputs		Outputs
x	*y*	*x* + *y*
0	0	0
0	1	1
1	0	1
1	1	1

x + y = x **OR** y
OR is true if **either** inputs are true

Inputs	Outputs
x	$\bar{x}$
0	1
1	0

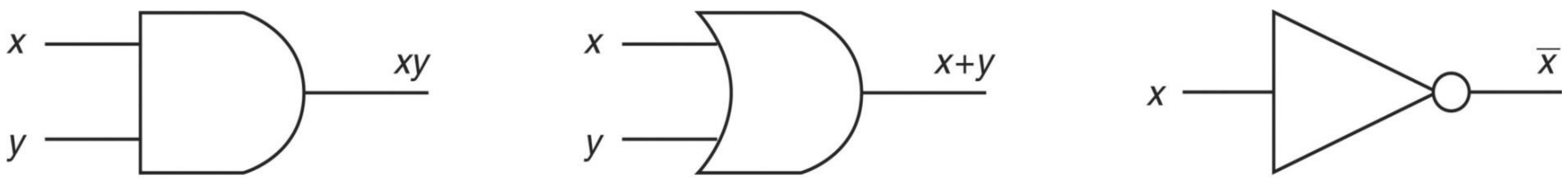
$\bar{x}$ (bar) = **NOT** x
NOT inverts the bit

NOR is NOT of OR, **NAND** is NOT of AND, **XOR** is true if both inputs differ

x	*y*	*x* NOR *y*
0	0	1
0	1	0
1	0	0
1	1	0

Logic Gates

Here we see the logic gates that represent the Boolean operations previously discussed

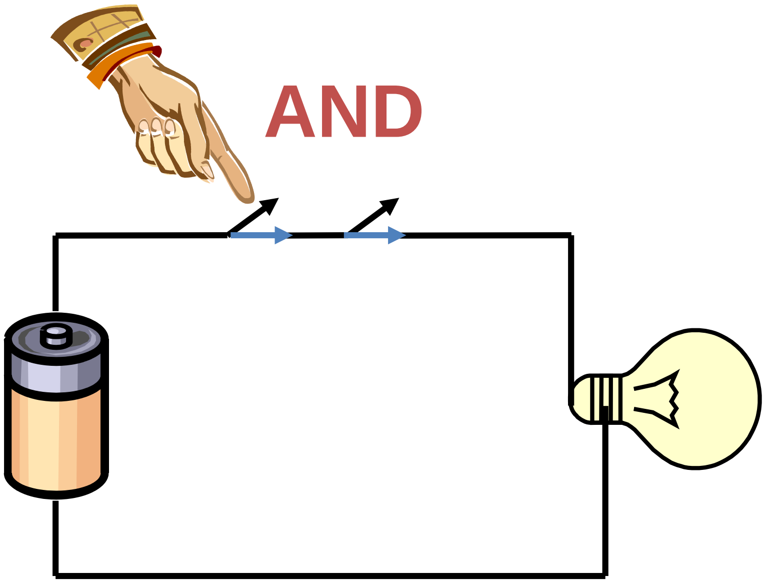


-Boolean multiplier　　　　-Boolean adder

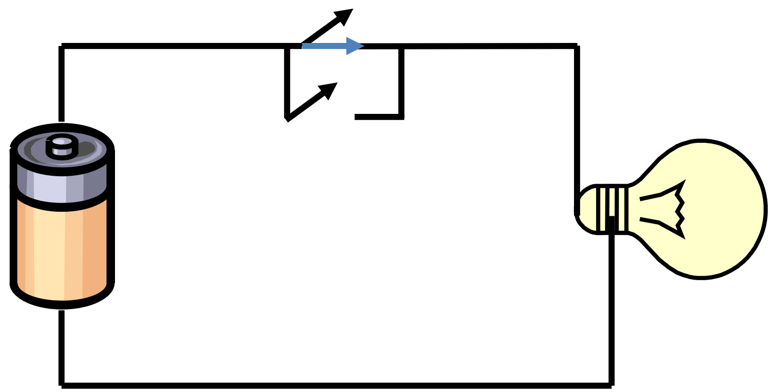
AND Gate　　　　OR Gate　　　　NOT Gate

Switching Circuits

AND



OR



Boolean Addition & Multiplication

- **Example 1:** Determine the value of A, B, C and D that make the sum term $A + \overline{B} + C + \overline{D}$ equal to 0.

Solution: To gat 0, all the terms should be 0. So $A = 0$, $\overline{B} = 0$, $C = 0$, $\overline{D} = 0$, $\longrightarrow$ $0 + \overline{1} + 0 + \overline{1} = 0$.

- **Example 2:** Determine the value of A, B, C and D that make the product term $A\overline{B}C\overline{D}$ equal to 1.

Solution: To gat 1, all the terms should be 1.

$$A\overline{B}C\overline{D} = 1 . \overline{0} . 1 . \overline{0} = 1$$

Boolean Addition & Multiplication

- **Example: Find the value (F) if $F(x,y,z) = x\bar{z} + y$**

Solution:

- As with common arithmetic, Boolean operations have rules of precedence.
- The NOT operator has highest priority, followed by AND and then OR.
- This is how we chose the (shaded) function subparts in our table.

$F(x,y,z) = x\bar{z}+y$

x	y	z	$\bar{z}$	$x\bar{z}$	$x\bar{z}+y$
0	0	0	1	0	0
0	0	1	0	0	0
0	1	0	1	0	1
0	1	1	0	0	1
1	0	0	1	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	0	0	1

Boolean Algebra

- Digital computers contain circuits that implement Boolean functions.
- The simpler that we can make a Boolean function, the smaller the circuit that will result.
  - Simpler circuits are cheaper to build, consume less power, and run faster than complex circuits.
- With this in mind, we always want to reduce our Boolean functions to their simplest form.
- So that, there are a number of Boolean identities (rules) that help us to do this.

Simplification of Boolean Functions

- An implementation of a Boolean Function requires the use of logic gates.
- A smaller number of gates, with each gate (other then Inverter) having less number of inputs, may reduce the cost of the implementation.
- There are 2 methods for simplification of Boolean functions.
  - ❖ The algebraic method by using Identities
  - ❖ The graphical method by using Karnaugh Map method

Basic Boolean Identities (Rules)

Identity Name	AND Form	OR Form
Identity Law	$1x = x$	$0+x = x$
Null (or Dominance) Law	$0x = 0$	$1+x = 1$
Idempotent Law	$xx = x$	$x+x = x$
Inverse Law	$x\overline{x} = 0$	$x+\overline{x} = 1$
Commutative Law	$xy = yx$	$x+y = y+x$
Associative Law	$(xy)z = x(yz)$	$(x+y)+z = x+(y+z)$
Distributive Law	$x+yz = (x+y)(x+z)$	$x(y+z) = xy+xz$
Absorption Law	$x(x+y) = x$	$x+xy = x$
DeMorgan's Law	$(\overline{xy}) = \overline{x}+\overline{y}$	$(\overline{x+y}) = \overline{x}\overline{y}$
Double Complement Law	$\overline{\overline{x}} = x$	

Basic Boolean Identities (Rules)

- **Example:** simplify using Boolean identities

$$F(X,Y,Z) = (X + Y)(X + \overline{Y})\,\overline{(X\overline{Z})}$$

$(X + Y)(X + \overline{Y})\,\overline{(X\overline{Z})}$	Idempotent Law (Rewriting)
$(X + Y)(X + \overline{Y})(\overline{X} + Z)$	DeMorgan's Law
$(XX + X\overline{Y} + XY + Y\overline{Y})(\overline{X} + Z)$	Distributive Law
$((X + Y\overline{Y}) + X(Y + \overline{Y}))(\overline{X} + Z)$	Commutative & Distributive Laws
$((X + 0) + X(1))(\overline{X} + Z)$	Inverse Law
$X(\overline{X} + Z)$	Idempotent Law
$X\overline{X} + XZ$	Distributive Law
$0 + XZ$	Inverse Law
XZ	Idempotent Law

Basic Boolean Identities (Rules)

- **Example:** $xy+\overline{x}z+yz = xy+\overline{x}z+yz*1$ (identity) =

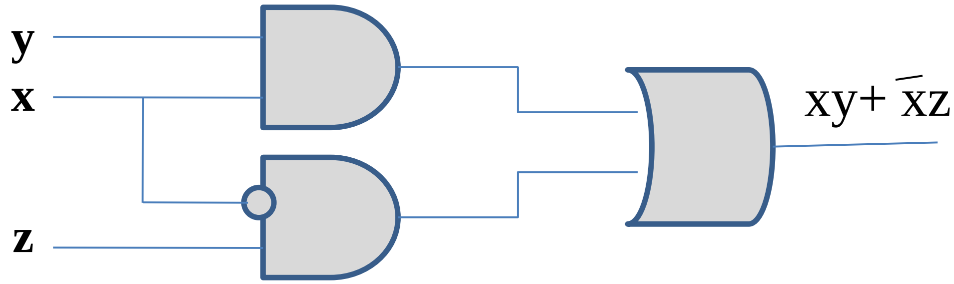
  $xy+\overline{x}z+yz*(x+\overline{x})$ (inverse) =

  $xy+\overline{x}z+xyz+\overline{x}yz$ (distributive) =

  $xy(1+z)+\overline{x}z(y+1)$ (distributive) =

  $xy(1)+\overline{x}z(1)$ (null) = $xy*1+\overline{x}z*1$

  (absorption)= $xy+\overline{x}z$ (identity)



DeMorgan’s law

- DeMorgan’s law can be extended to any number of variables.
- Replace each variable by its complement and change all ANDs to ORs and all ORs to ANDs.
- Thus, we find the complement of:

$$F(X,Y,Z) = (XY) + (\overline{X}Z) + (Y\overline{Z})$$

$$\overline{F}(X,Y,Z) = \overline{(XY) + (\overline{X}Z) + (Y\overline{Z})}$$

$$= \overline{(XY)}\,\overline{(\overline{X}Z)}\,\overline{(Y\overline{Z})}$$

$$= (\overline{X}+\overline{Y})(X+\overline{Z})(\overline{Y}+Z)$$

Basic Boolean Identities (Rules)

- **Example:** simplify using Boolean algebra

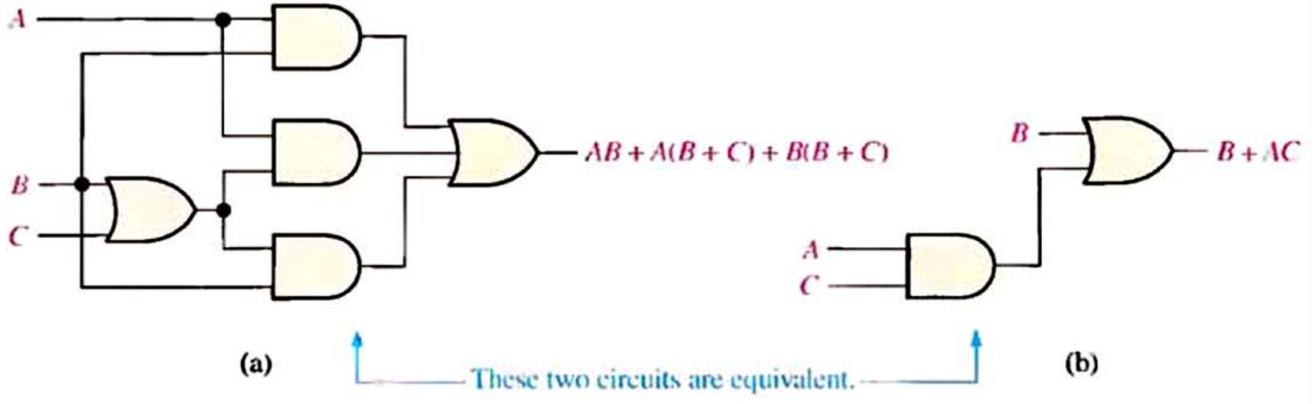
AB + A(B+C) + B(B+C)

AB + AB + AC + BB + BC …. (distributed law)

AB + AC + B + BC …. (AB + AB = AB & BB=B)

AB + AC + B ….. (B + BC =B)

B + AC …. (AB + B =B)



Boolean Algebra

- There are two canonical forms for Boolean expressions: sum-of-products (**SOP**) and product-of-sums (**POS**).
  – Recall the Boolean product is the AND operation and the Boolean sum is the OR operation.
- In the sum-of-products form, ANDed variables are ORed together.
  – For example: $F(x,y,z) = xy + xz + yz$
- In the product-of-sums form, ORed variables are ANDed together:
  – For example: $F(x,y,z) = (x+y)(x+z)(y+z)$

Boolean Algebra

- It is easy to convert a function to sum-of-products form using its truth table.
- We are interested in the values of the variables that make the function true (=1).
- Using the truth table, we list the values of the variables that result in a true function value.
- Each group of variables is then ORed together.

$$F(x,y,z) = x\bar{z}+y$$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

- The sum-of-products form for our function is:

$$F(x,y,z) = \bar{x}y\bar{z} + \bar{x}yz + x\bar{y}\bar{z} + xy\bar{z} + xyz$$

We note that this function is not in simplest terms. Our aim is only to rewrite our function in canonical sum-of-products form.

$$F(x,y,z) = x\bar{z} + y$$

x	y	z	$x\bar{z}+y$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

- **Example1:** Convert the following Boollean algebra to sum of product forms: $\overline{(\overline{A + B}) + C}$

Solution: $\overline{(\overline{A + B}) + C} = (\overline{\overline{A + B}}) \cdot \overline{C} = (A + B)\,\overline{C} = A\overline{C} + B\overline{C}$

- **Example2:** From the truth table,

determine the standard SOP

expression and the equivalent

standard POS expression

INPUTS			OUTPUT
A	*B*	*C*	*X*
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	0
1	1	0	1
1	1	1	1

Boolean Algebra

- Solution: There are four 1s in the output column and the corresponding binary values are 011, 100, 110, 111. Convert these binary values to produce terms as follows:

$011 \longrightarrow \bar{A}BC,\quad 100 \longrightarrow A\bar{B}\bar{C},\quad 110 \longrightarrow AB\bar{C},\quad 111 \longrightarrow ABC$

The resulting standard **SOP** expression for the output X is

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

- For the **POS** expression the output is **0** for the binary values 000, 001 010, 101. Convert these binary values to sum terms:

$000 \longrightarrow A+B+C,\ 001 \longrightarrow A+B+\bar{C},\ 010 \longrightarrow A+\bar{B}+C,\ 101 \longrightarrow \bar{A}+B+\bar{C}$

The resulting standard POS expression for the output X is:

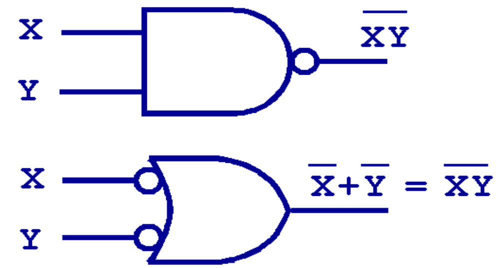
$X = (A+B+C)(A+B+\bar{C})(A+\bar{B}+C)(\bar{A}+B+\bar{C})$

Logic Gates

NAND and NOR are two very important gates. Their symbols and truth tables are shown at the right.

X NAND Y

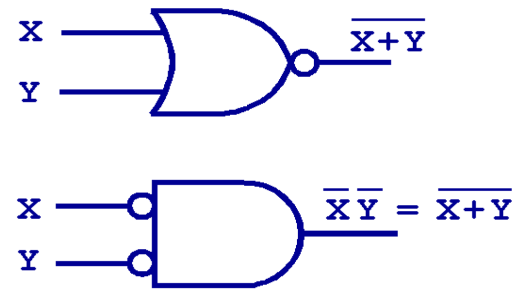
X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0



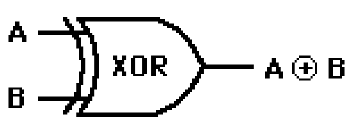
NAND

X NOR Y

X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0



NOR

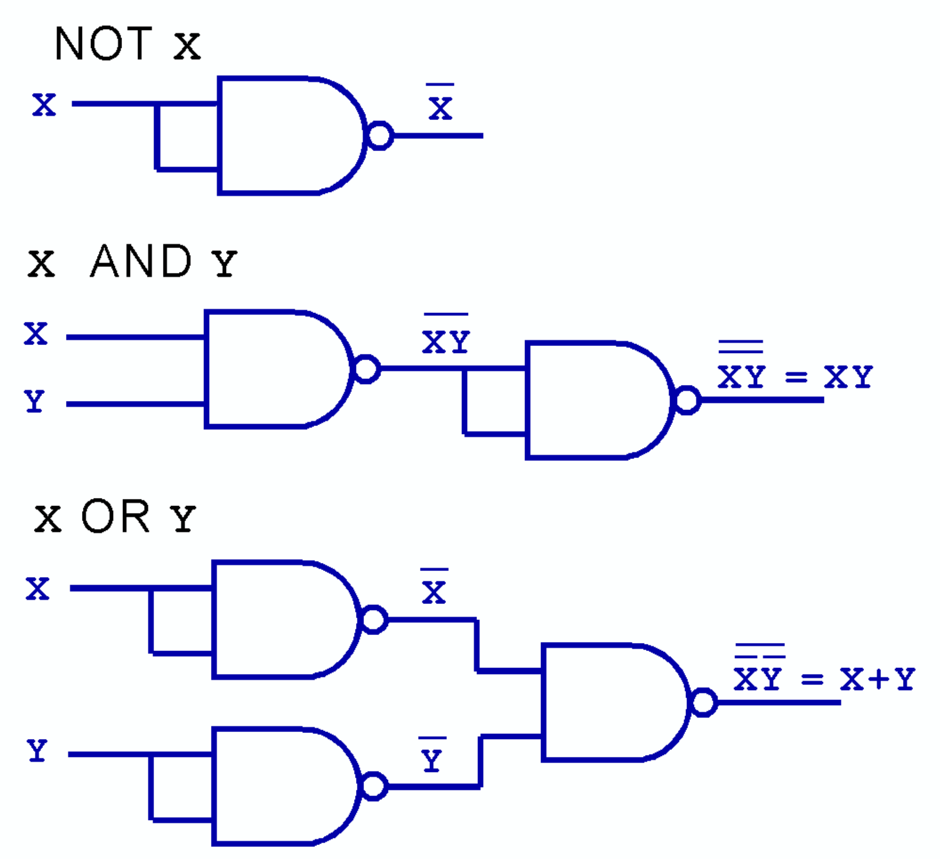


A	B	Out
0	0	0
0	1	1
1	0	1
1	1	0

XOR looks like OR but with the added curved line

Logic Gates

- NAND and NOR are known as ***universal gates*** because they are inexpensive to manufacture, and any Boolean function can be constructed using only NAND or only NOR gates.

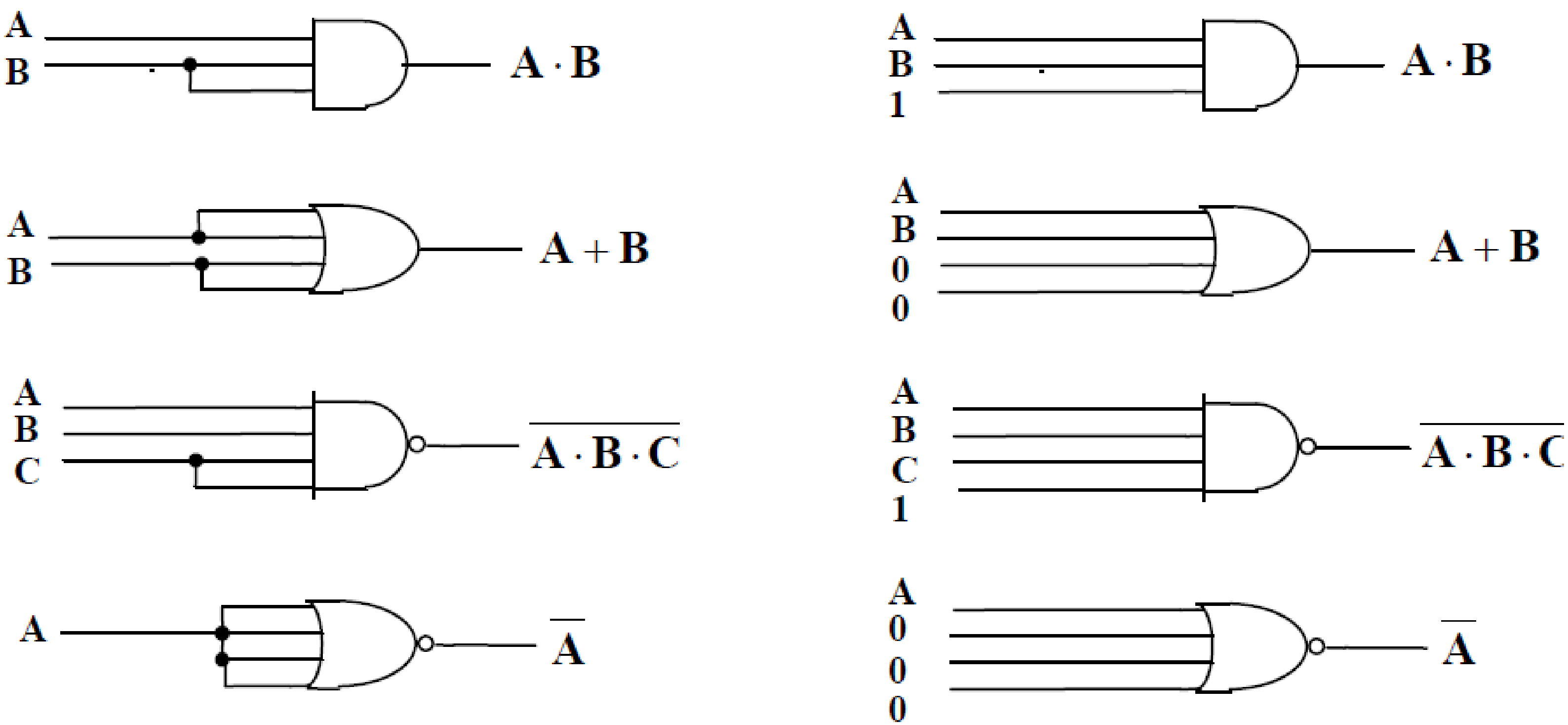


Logic Gates

- **Fan-in** is the number of inputs a gate can handle. Physical logic gates with a large fan-in tend to be slower than those with a small fan-in. This is because the complexity of the input circuitry increases the input capacitance of the device. Using logic gates with higher fan-in will help reducing the depth of a logic circuit.
- The fan-out of a logic gate output is the number of gate inputs it can feed or connect to.
- The maximum fan-out of an output measures its load-driving capability: it is the greatest number of inputs of gates of the same type to which the output can be safely connected.

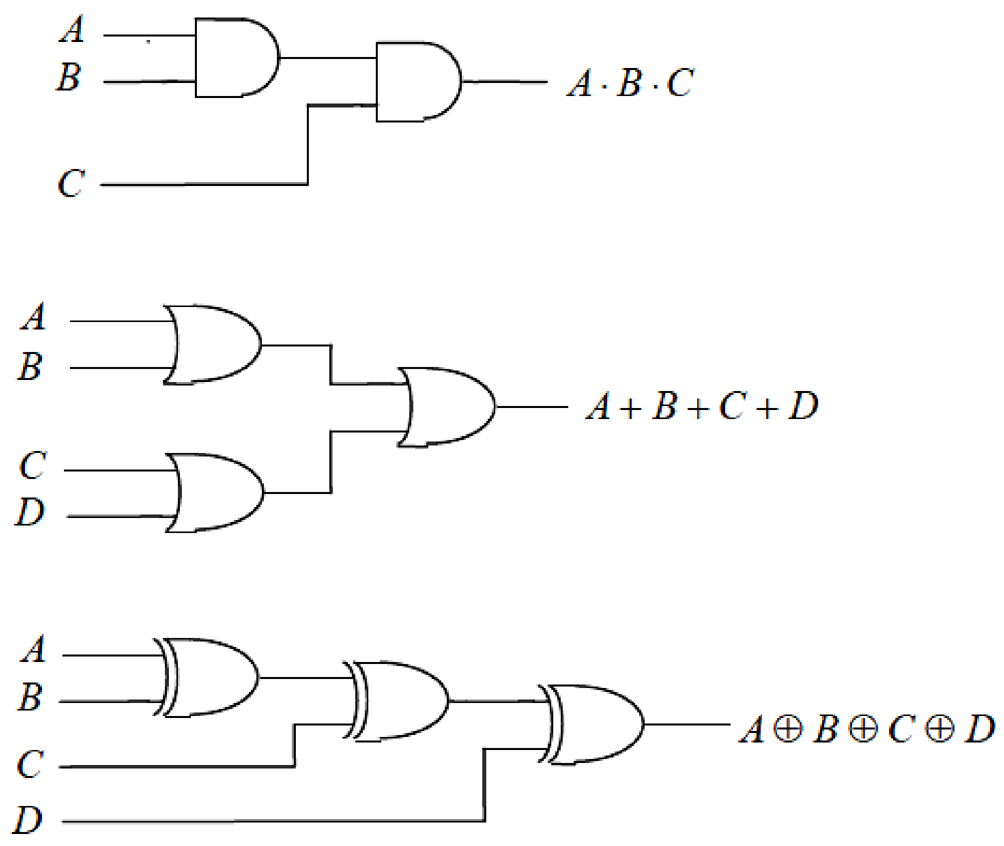
Logic Gates

- To reduce the Fan-in for the gate, we do:

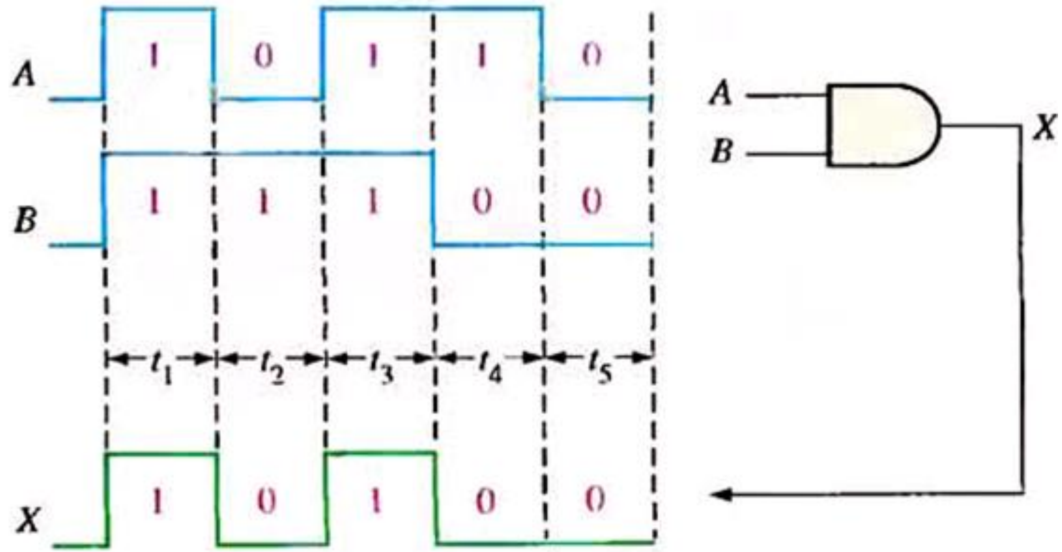


Logic Gates

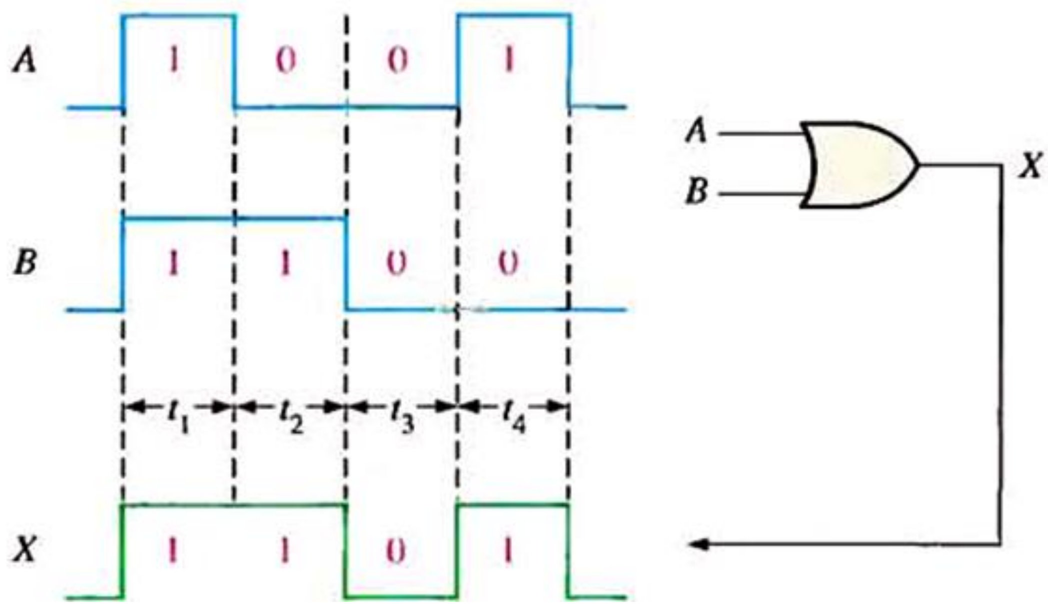
- If the available gates with limited inputs number, so we do:



Timing Diagram for AND Gate



- Timing Diagram for OR Gate



Example: If A is 10001, and B is 00100 are applied to a NOR gate, what is the resulting output waveform?