EE2026 Digital Design

BOOLEAN ALGEBRA, LOGIC GATES

Chua Dingjuan elechuad@nus.edu.sg

BOOLEAN ALGEBRA

Postulates, Theorems, Laws, AND, OR, NOT, XOR, Minterm, Maxterm, SOP/POS, CSOP/CPOS

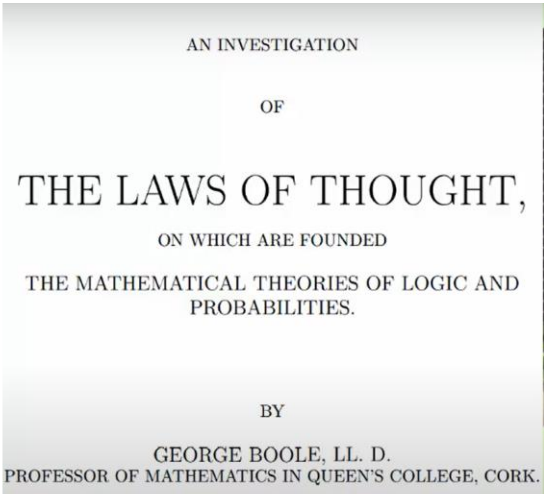
Outline

- What is Boolean Algebra?
- Theorems and Postulates
- Boolean functions and truth table
- Boolean function simplification using algebra manipulation

What is Boolean Algebra?

Brief History:

- Boolean was developed in 1854 by George Boole (An English mathematician, philosopher, and logician)
- Huntington formulated the postulates in 1904 as the formal definition
- Boolean Algebra is the mathematical foundation for digital system design, including computers
- It was first applied to the practical problem (Analysis of networks of relays) in late 1930s by C.E Shannon (MIT) who later introduced “Switching algebra” in 1938
- Switching algebra is a Boolean algebra in which the number of elements is precisely two


AN INVESTIGATION

OF

THE LAWS OF THOUGHT,

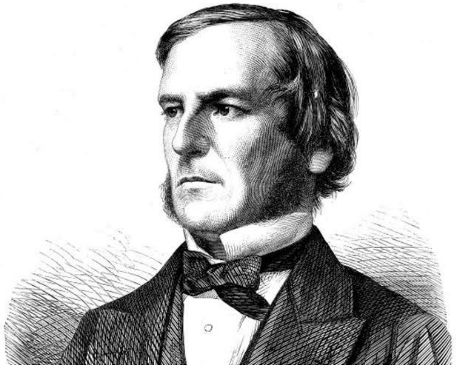
ON WHICH ARE FOUNDED

THE MATHEMATICAL THEORIES OF LOGIC AND PROBABILITIES.

BY

GEORGE BOOLE, LL. D.

PROFESSOR OF MATHEMATICS IN QUEEN'S COLLEGE, CORK.



Boolean Algebra

- Boolean algebra is a two-valued type of switching algebra
- Switching algebra represents bistable electrical switching circuits (On or Off)
- Boolean algebra is defined by a set of elements, **B**, and there are two main operators (***AND, OR***)
  - Binary operators (two arguments involved)
    - ***AND → "."***
    - ***OR → "+"***
  - Plus, one unary operator (only one argument involved)
    - ***NOT → "$\overline{\quad}$"*** (*Complement* operator represented by an overbar)

- Boolean algebra satisfies six Huntington postulates

Ref - Postulates of Boolean Algebra

There are Six Huntington Postulates that define the Boolean Algebra:

1. Closure - For all elements x and y in the set $\boldsymbol{B}$
   i. $x + y \ is\ an\ element\ of\ \boldsymbol{B}\ and$
   ii. $x \cdot y\ is\ an\ element\ of\ \boldsymbol{B}$

2. There exists a 0 and 1 element in ***B***, such that
   i. $x + 0 = x$
   ii. $x \cdot 1 = x$

3. Commutative Law
   i. $x + y = y + x$
   ii. $x \cdot y = y \cdot x$

4. Distributive Law
   i. $x \cdot (y + z) = x \cdot y + x \cdot z$ $(\cdot\ over\ +)$
   ii. $x + (y \cdot z) = (x + y) \cdot (x + z)$ $(+\ over\ \cdot)$

5. For every element x in the set ***B***, there exists an element $\bar{x}$ in the set ***B***, such that
   i. $x + \bar{x} = 1$
   ii. $x \cdot \bar{x} = 0$
   ($\bar{x}$ is called the **complement** of x)

6. There exist at least two distinct elements in the set ***B***

Ref - Boolean vs. Ordinary Algebra

Boolean algebra	Ordinary algebra
No associative law. But it can be derived from the other postulates	Associative law is included: $a+(b+c)=(a+b)+c$
Distributive law: $x+(y\cdot z)=(x+y)\cdot(x+z)$ valid	Not valid
No additive or multiplicative inverses, therefore there are no subtraction and division operation	Subtraction and division operations exist
Complement operation available	No complement operation
Boolean algebra: Undefined set of elements; Switching algebra: a two-valued Boolean algebra, whose element set only has two elements, 0 and 1.	Dealing with real numbers and constituting an infinite set of elements

The Three Operators in Two-Valued Boolean Algebra (*B={0,1}*)

OR: $A + B$

A	B	$A+B$
0	0	0
0	1	1
1	0	1
1	1	1

$$0 + 0 = 0$$
$$0 + 1 = 1$$
$$1 + 0 = 1$$
$$1 + 1 = 1$$

AND: $A \cdot B$

A	B	$A \cdot B$
0	0	0
0	1	0
1	0	0
1	1	1

$$0 \cdot 0 = 0$$
$$0 \cdot 1 = 0$$
$$1 \cdot 0 = 0$$
$$1 \cdot 1 = 1$$

NOT: $\overline{A}$

A	$\overline{A}$
0	1
1	0

$$\mathrm{A} = 0 \rightarrow \overline{A} = 1$$
$$\mathrm{A} = 1 \rightarrow \overline{A} = 0$$

Priority: NOT has highest precedence, followed by AND and OR

NOT(A · B + C) = NOT((A · B) + C)

Theorems of Boolean Algebra

#	Theorem		
1	$A + A = A$	$A \cdot A = A$	Tautology Law
2	$A + 1 = 1$	$A \cdot 0 = 0$	Union Law
3	$\overline{(\bar{A})} = A$		Involution Law
4	$A + (B + C)$ $= (A + B) + C$	$A \cdot (B \cdot C) = (A \cdot B) \cdot C$	Associative Law
5	$\overline{A + B} = \bar{A} \cdot \bar{B}$	$\overline{A \cdot B} = \bar{A} + \bar{B}$	De Morgan's Law
6	$A + A \cdot B = A$	$A \cdot (A + B) = A$	Absorption Law
7	$A + \bar{A} \cdot B = A + B$	$A \cdot (\bar{A} + B) = A \cdot B$	
8	$AB + A\bar{B} = A$	$(A + B)(A + \bar{B}) = A$	Logical adjacency
9	$AB + \bar{A}C + BC$ $= AB + \bar{A}C$	$(A + B)(\bar{A} + C)(B + C)$ $= (A + B)(\bar{A} + C)$	Consensus Law

Duality (OR and AND, 0 and 1 can be interchanged)

Boolean Functions and Truth Table

- A Boolean function expresses the logical relationship between binary variables.
- It can be evaluated by determining the binary value of the expression for all possible values of the variables

Truth table is a tabular technique for listing all possible combinations of input variables and the values of function for each combination

$$F_1 = A + B$$

A	B	F_1
0	0	0
0	1	1
1	0	1
1	1	1

$$F_3 = A + BC$$

A	B	C	F_3
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

Examples - Truth Table

Prove the De Morgan's Law:

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

A	B	$\overline{\boldsymbol{A}+\boldsymbol{B}}$	$\bar{\boldsymbol{A}} \cdot \bar{\boldsymbol{B}}$
0	0	1	1
0	1	0	0
1	0	0	0
1	1	0	0

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

A	B	$\overline{\boldsymbol{A} \cdot \boldsymbol{B}}$	$\bar{\boldsymbol{A}} + \bar{\boldsymbol{B}}$
0	0	1	1
0	1	1	1
1	0	1	1
1	1	0	0

Prove : $A + \bar{A} \cdot B = A + B$

A	B	$\boldsymbol{A} + \bar{\boldsymbol{A}} \cdot \boldsymbol{B}$	$\boldsymbol{A} + \boldsymbol{B}$
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	1

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

A	B	$\boldsymbol{A} \cdot (\boldsymbol{A} + \boldsymbol{B})$	$\boldsymbol{A}$
0	0	0	0
0	1	0	0
1	0	1	1
1	1	1	1

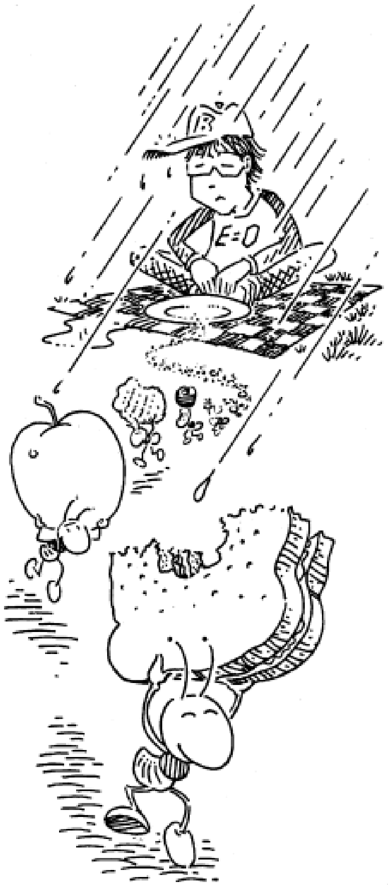
Truth Table – examples (cont.)

Prove : $A + (B \cdot C) = (A + B) \cdot (A + C)$

$\boldsymbol{A}$	$\boldsymbol{B}$	$\boldsymbol{C}$	$\boldsymbol{A} + (\boldsymbol{B} \cdot \boldsymbol{C})$	$(\boldsymbol{A} + \boldsymbol{B}) \cdot (\boldsymbol{A} + \boldsymbol{C})$
0	0	0	0	0
0	0	1	0	0
0	1	0	0	0
0	1	1	1	1
1	0	0	1	1
1	0	1	1	1
1	1	0	1	1
1	1	1	1	1

But how do we use these ideas?

Design Example



Ben Bitdiddle is having a picnic. He won't enjoy it if it rains or if there are ants. Design a circuit that will output TRUE only if Ben enjoys the picnic.

Solution

Inputs : A (Ants), B (Rain)

Output : E (Ben's Enjoyment)

Boolean Algebra Version

E = ?

Ben enjoys his picnic *if* there is no rain **and** no ants :

E = 1 *if* $A = 0$ **and** $B = 0$

Truth Table Version

A	B	*E*
0	0	
0	1	
1	0	
1	1	

E = ?

Minterm and Maxterm

A	B	F	Minterm	Maxterm
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	0	$A \cdot B$	$\bar{A} + \bar{B}$

minterm $\mathrm{A}.\mathrm{B} = 1$ for
A = 1 and B = 1

maxterm $\bar{A} + \bar{B} = 0$
for A = 1 and B = 1

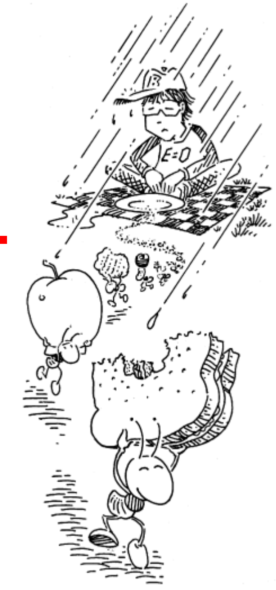
Minterm

- ***Minterm*** is a *product term* that contains all variables in the function
- AND all the variables
- If the variable in truth table is "0", take its complement in the minterm
- Minterm is equal to 1 for that set of given input

Maxterm

- ***Maxterm*** is a *sum term* that contains all variables in the function
- OR all the variables
- If the variable in truth table is "1", take its complement in the maxterm
- Maxterm is equal to 0 for that set of given input

Truth Table → CSOP or CPOS



A Boolean function is in canonical form if it is expressed as

- a ***sum*** of ***minterms*** (Canonical Sum Of Products - CSOP) or
- a ***product*** of ***maxterms*** (Canonical Product Of Sums - CPOS)

Using Ben's Bitdiddle's truth table :

A	B	E	minterm	maxterm
0	0	1	$\bar{A} \cdot \bar{B}$	$A + B$
0	1	0	$\bar{A} \cdot B$	$A + \bar{B}$
1	0	0	$A \cdot \bar{B}$	$\bar{A} + B$
1	1	0	$A \cdot B$	$\bar{A} + \bar{B}$

A truth table expressed in either CSOP or CPOS. (How should we choose?)

- CSOP → sum the minterms that make output = 1

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

- CPOS → product the maxterms that make output = 0

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

Are the above two functions equivalent? How do we check?

SOP and POS → Truth Table

Are the following two Boolean functions same?

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

Let's use truth tables to check:

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

A	B	E_1
0	0	
0	1	
1	0	
1	1	

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

A	B	E_2
0	0	
0	1	
1	0	
1	1	

SOP: If any *PRODUCT* in SOP is "1", the function is "1". Otherwise, the function is "0"

POS: If any *SUM* in POS is "0", the function is "0". Otherwise, the function is "1"

SOP and POS are different ways to present the same Boolean function

SOP and POS → Truth Table

Are the following two Boolean functions same?

$$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$$

$$\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$$

Can we prove this via Boolean algebra?

$\boldsymbol{E_1(A,B)} = \bar{A} \cdot \bar{B}$ $\quad\quad$ $\boldsymbol{E_2(A,B)} = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$

Example : Truth Table → SOP → POS

Truth table:

A	B	C	F_1	$\overline{F_1}$
0	0	0	0	1
0	0	1	0	1
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	1	0
1	1	0	1	0
1	1	1	0	1

Use SOP:

start from CSOP of NOT(F) (otherwise, complemented POS is obtained from SOP)

$$\overline{\boldsymbol{F_1}}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) = \overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\,\overline{\boldsymbol{C}} + \overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\boldsymbol{C} + \overline{\boldsymbol{A}}\boldsymbol{B}\boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}\boldsymbol{C}$$

Apply De Morgan's Law:

$$\begin{aligned}\boldsymbol{F_1}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) &= \overline{\overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\,\overline{\boldsymbol{C}} + \overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\boldsymbol{C} + \overline{\boldsymbol{A}}\boldsymbol{B}\boldsymbol{C} + \boldsymbol{A}\boldsymbol{B}\boldsymbol{C}} \\ &= \overline{\overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\,\overline{\boldsymbol{C}}} \cdot \overline{\overline{\boldsymbol{A}}\,\overline{\boldsymbol{B}}\boldsymbol{C}} \cdot \overline{\overline{\boldsymbol{A}}\boldsymbol{B}\boldsymbol{C}} \cdot \overline{\boldsymbol{A}\boldsymbol{B}\boldsymbol{C}} \\ &= (\boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C})(\boldsymbol{A}+\boldsymbol{B}+\overline{\boldsymbol{C}})(\boldsymbol{A}+\overline{\boldsymbol{B}}+\overline{\boldsymbol{C}})(\overline{\boldsymbol{A}}+\overline{\boldsymbol{B}}+\overline{\boldsymbol{C}})\end{aligned}$$

Use POS directly from truth table:

clearly the same

$$\begin{aligned}\boldsymbol{F_1}(\boldsymbol{A},\boldsymbol{B},\boldsymbol{C}) \\ &= (\boldsymbol{A}+\boldsymbol{B}+\boldsymbol{C})(\boldsymbol{A}+\boldsymbol{B}+\overline{\boldsymbol{C}})(\boldsymbol{A}+\overline{\boldsymbol{B}}+\overline{\boldsymbol{C}})(\overline{\boldsymbol{A}}+\overline{\boldsymbol{B}}+\overline{\boldsymbol{C}})\end{aligned}$$

POS can be obtained from SOP (and vice versa) by starting from complemented SOP of F and applying the De Morgan's Law

Example-1: Non-Canonical → Canonical Form via Truth Table

Example: For the given Boolean function below, find a canonical *minterm* and *maxterm* expression.

1) obtain the truth table from the given function
2) find *minterm* or *maxterm* expression from truth table (CSOP or CPOS)

$F(x,y,z) = \bar{x} + y\bar{z}$ ← Non Canonical form

Truth table:

x	y	z	F
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	0

Canonical *minterm* expression:

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

(only contains the *minterms* that make the function = 1)

Canonical *maxterm* expression:

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

(only contains the *maxterms* that make the function = 0)

Example-2: Non-Canonical → Canonical Form via Postulates and Theorems

Example: For the given Boolean functions below, convert it to canonical *minterm or maxterm* expression.

(*Using postulates/theorem to expand the given function to canonical form)

SOP → CSOP:
(CSOP – Canonical SOP)

$$
\begin{aligned}
F(x,y,z) &= \bar{x}y + xz \\
&= \bar{x}y \cdot 1 + x \cdot 1 \cdot z \\
&= \bar{x}y(\mathrm{z} + \bar{z}) + x(y + \bar{y})z \\
&= \bar{x}yz + \bar{x}y\bar{z} + xyz + x\bar{y}z
\end{aligned}
$$

For missing literals, complete minterms through postulates:
$\mathrm{A} \cdot 1 = A \quad and \quad A + \bar{A} = 1$

SOP → CPOS:
(CSOP – Canonical POS)

$$
\begin{aligned}
F(x,y,z) &\overset{1)}{=} \bar{x}y + xz \\
&= \overline{\overline{\bar{x}y + xz}} \quad \text{a} \\
&= \overline{(x + \bar{y})(\bar{x} + \bar{z})} \quad \text{b} \\
&= \overline{x\bar{x} + x\bar{z} + \bar{x}\bar{y} + \bar{y}\bar{z}} \quad \text{c} \\
&= (x + y)(\bar{x} + z)(y + z) \quad \text{d} \\
2)\ &= (x + y + z) \cdot (x + y + \bar{z}) \cdot (\bar{x} + y + z) \cdot (\bar{x} + \bar{y} + z) \\
&\quad \cdot (x + y + z) \cdot (\bar{x} + y + z)
\end{aligned}
$$

1) express SOP as POS
   - a) complement twice
   - b) apply De Morgan's law
   - c) expand
   - d) re-apply De Morgan's law
2) for missing literals, complete maxterms through distribution postulate

Use distribution postulate:
$A + (\boldsymbol{B}\boldsymbol{C}) = (A + \boldsymbol{B})(A + \boldsymbol{C})$
(A = incomplete sum,
C=NOT(B)=missing literal)

$x + y = (x + y) + \boldsymbol{z}\bar{\boldsymbol{z}}$
$= (x + y + \boldsymbol{z})(x + y + \bar{\boldsymbol{z}})$ ⟶

Pre-Lab Exercise

The following task is required to be implemented :

- When switch **A** turns on, only **LED1** lights up.
- When switch **B** turns on, only **LED2** lights up.
- When both switches **A** and **B** turn on, **LED1**, **LED2**, and **LED3** light up.

INPUT		OUTPUT			MINTERM
A	B	LED1	LED2	LED3	
0	0				$\bar{A}\bar{B}$
0	1				$\bar{A}B$
1	0				$A\bar{B}$
1	1				AB

LED1 =

LED2 =

LED3 =

Summary

- Postulates and theorems of Boolean algebra
- Three binary operators: AND, OR and NOT
- Boolean Functions
- Truth table and Boolean function evaluation using truth table
- Boolean function in SOP or POS form
- Obtain SOP or POS from truth table
- Minterm and maxterm
- Canonical form of Boolean function
- Convert non-canonical form to canonical SOP or POS expressions.

LOGIC GATES

AND, OR, NOT, XOR gates

Outline

Logic gate introduction

- AND/NAND, OR/NOR, NOT/Buffer, XOR/NXOR
- different levels of description (Boolean, truth table, graphical, Verilog)

Implementation of Boolean function using gates

- different levels of description (Boolean, graphical, Verilog)

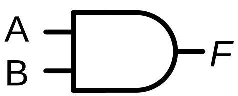
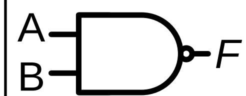
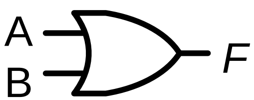
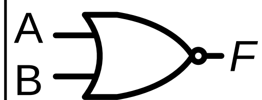
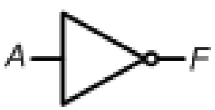
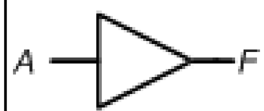
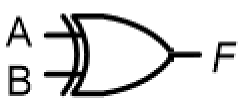
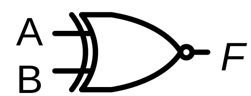
Design simplification by algebra manipulation

Commercial logic gates

Logic Gate Introduction

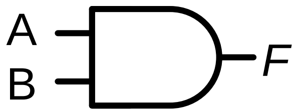
Logic gates are digital circuits that implement the Boolean operations.

Basic Logic Gates:

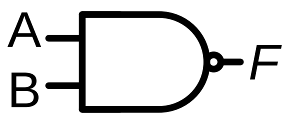
Gate	Symbol	Function (F)	Verilog Operator	Gate	Symbol	Function (F)	Verilog Operator
AND		$A \cdot B$	F = A & B	NAND		$\overline{A \cdot B}$	F = ~(A & B)
OR		$A + B$	F = A | B	NOR		$\overline{A + B}$	F = ~(A | B)
NOT		$\bar{A}$	F = ~A	Buffer		A	F = A
XOR		$A \oplus B$	F = A ^ B	XNOR		$\overline{A \oplus B}$	F = ~(A ^ B)

Verilog Bit-wise Operator Precedence : ~, &, ^, |

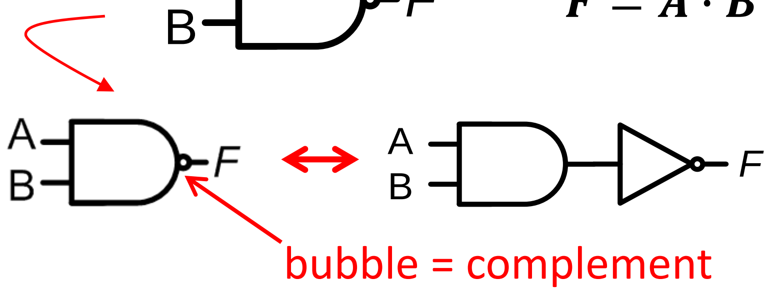
AND and NAND Gates



$$F = A \cdot B$$



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



Truth Table (AND, NAND):

		AND	NAND
A	B	$A \cdot B$	$\overline{A \cdot B}$
0	0	0	1
1	0	0	1
0	1	0	1
1	1	1	0

AND

F is TRUE only when both ***A*** and ***B*** are TRUE

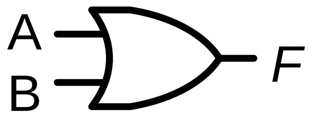
```
module andgate(A, B, F);
      input A, B;
      output F;
      assign F = A & B;
endmodule
```

NAND

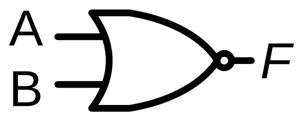
- ***F*** is FALSE only if both ***A*** and ***B*** are TRUE

```
module nandgate(A, B, F);
      input A, B;
      output F;
      assign F =
endmodule
```

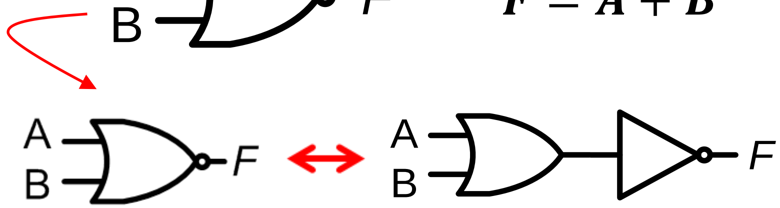
OR and NOR Gates



$$F = A + B$$



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



Truth Table (OR, NOR):

		OR	NOR
A	B	$A + B$	$\overline{A + B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	1	0

OR

- ***F*** is FALSE only when both ***A*** and ***B*** are FALSE

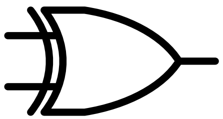
NOR

- ***F*** is TRUE only if both ***A*** and ***B*** are FALSE

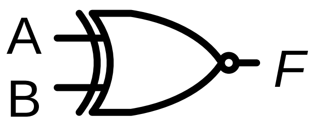
```
module orgate(A, B, F);
    input A, B;
    output F;
    assign F = A | B;
endmodule
```

```
module norgate(A, B, F);
    input A, B;
    output F;
    assign F =
endmodule
```

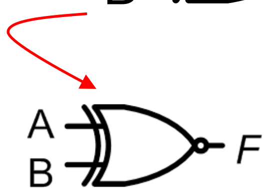
XOR and XNOR Gates

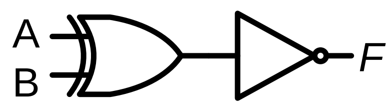


$$F = A\overline{B} + \overline{A}B = A \oplus B$$



$$F = \overline{A \oplus B}$$





Truth Table (XOR, XNOR):

		XOR	XNOR
A	B	$A \oplus B$	$\overline{A \oplus B}$
0	0	0	1
1	0	1	0
0	1	1	0
1	1	0	1

XOR

- ***F*** is TRUE if ***A*** ≠ ***B***

XNOR

- ***F*** is TRUE if ***A*** = ***B***

```
module xorgate(A, B, F);
      input A, B;
      output F;
      assign F = A ^ B;
endmodule
```

```
module xnorgate(A, B, F);
      input A, B;
      output F;
      assign F = ~(A ^ B);
endmodule
```

Cont’d Ben Bitdiddle’s Example

We developed the following two Boolean expressions for Ben :

$\boldsymbol{E}(\boldsymbol{A},\boldsymbol{B}) = \bar{A} \cdot \bar{B}$

$\boldsymbol{E}(\boldsymbol{A},\boldsymbol{B}) = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$

How do we implement them?

$\boldsymbol{E_1}(\boldsymbol{A},\boldsymbol{B}) = \bar{A} \cdot \bar{B}$

$\boldsymbol{E_2}(\boldsymbol{A},\boldsymbol{B}) = (A + \bar{B}).(\bar{A} + B).(\bar{A} + \bar{B})$

Implementation using Logic Gates – Sketch Method

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(w, x, y, z) = \bar{w}\bar{x}z + \bar{w}xz + wyz + xyz$$

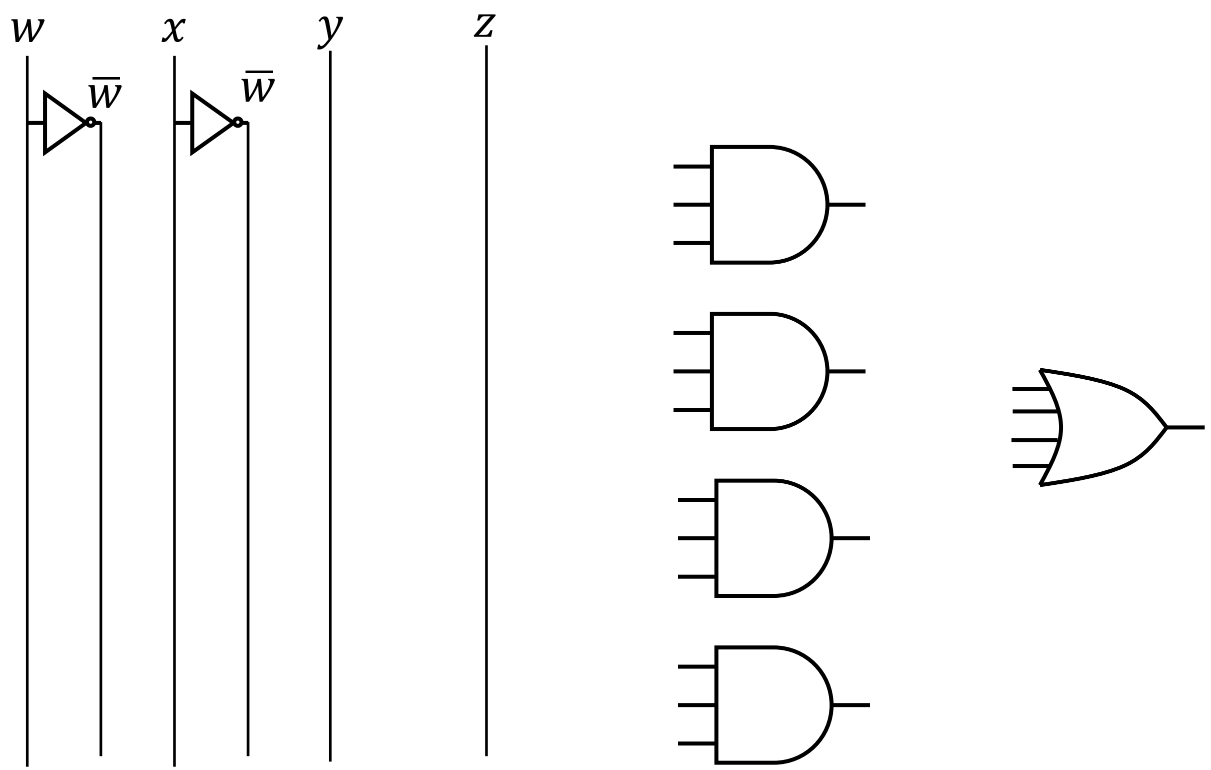
Input signals needed?

$w, \bar{w}, x, \bar{x}, y, z$

Types / # of Gates needed?

AND – 3 inputs

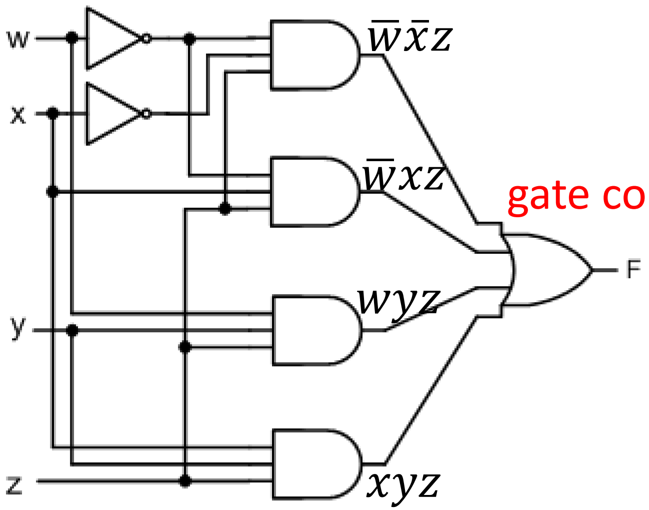
OR – 4 inputs



Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

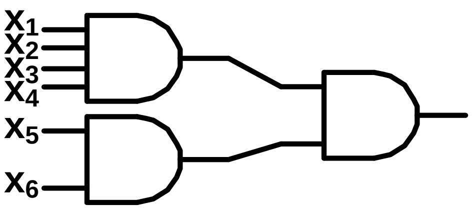
$$F(w,x,y,z) = \bar{w}\bar{x}z + \bar{w}xz + wyz + xyz$$



gate count = 7

if AND5 or more is needed: two-level ANDing (same for OR):

$x_1 \cdot x_2 \cdot x_3 \cdot x_4 \cdot x_5 \cdot x_6 = (x_1 \cdot x_2 \cdot x_3 \cdot x_4) \cdot (x_5 \cdot x_6)$



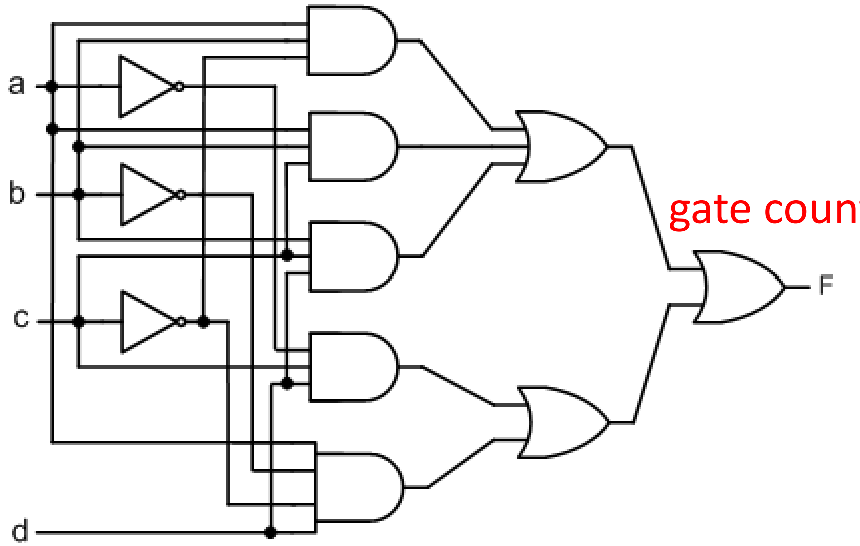
parentheses (~w & x & z) not needed in SOP, as precedence order is ~, &, ^, |

```
module func(w,x,y,z,F);
    input w, x, y, z;
    output F;
    assign F = ~w & ~x & z | ~w & x & z | w & y & z | x & y & z;
endmodule
```

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c,d) = ab\bar{c} + abc + bcd + \bar{a}cd + a\bar{b}\bar{c}d$$

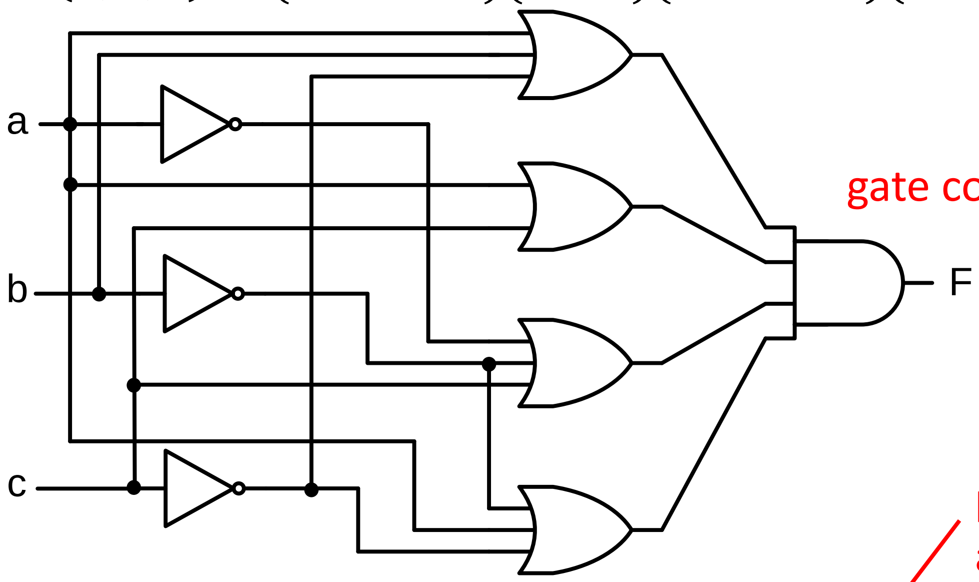



```
module func(a,b,c,d,F);
    input a, b, c, d;
    output F;
    assign F = a & b & ~c | a & b & c | b & c & d | ~a & c & d | a & ~b & ~c & d;
endmodule
```

Implementation of Boolean Function using Logic Gates

- Implement the following Boolean functions to logic gates, assume that the maximum number of inputs of a gate is 4.

$$F(a,b,c) = (a+b+\bar{c})(a+c)(\bar{a}+\bar{b}+c)(a+\bar{b}+\bar{c})$$



gate count = 8

parentheses (~a | ~b | c) needed in POS, as precedence order is ~, &, ^, |

```
module func(a,b,c,F);
    input a, b, c;
    output F;
    assign F = (a | b | ~c) & (a | c) & (~a | ~b | c) & (a | ~b | ~c);
endmodule
```

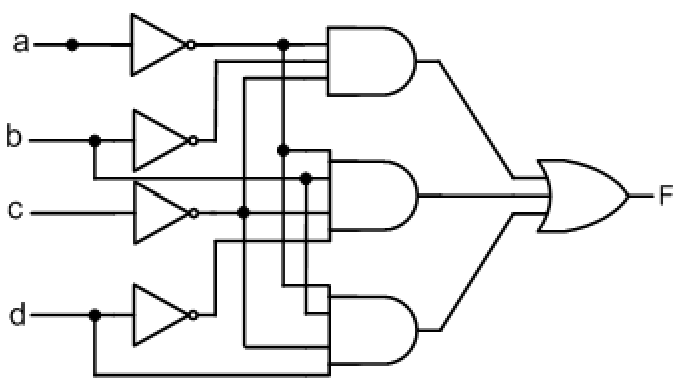
Boolean Function Simplification using Algebra Manipulation

- To reduce the hardware cost, the Boolean function can be simplified before implemented using logic gates
- A simplified Boolean Function contains a minimal number of terms such that no other expression with fewer literals and terms will represent the original function
- Simplification can be done by
  - Algebraic manipulation using postulates and theorem
  - Karnaugh Map

Boolean Function Simplification

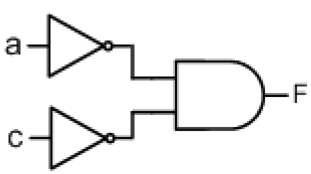
$$
\begin{aligned}
F(a,b,c,d) &= \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c}\bar{d} + \bar{a}b\bar{c}d \\
&= \bar{a}\bar{b}\bar{c} + \bar{a}b\bar{c}\left(\bar{d} + d\right) \leftarrow \quad (A + \bar{A} = 1) \\
&= \bar{a}\bar{c}\left(\bar{b} + b\right) \leftarrow \quad (A + \bar{A} = 1) \\
&= \bar{a}\bar{c}
\end{aligned}
$$

Before simplification:



gate count = 8

After simplification:



gate count = 3

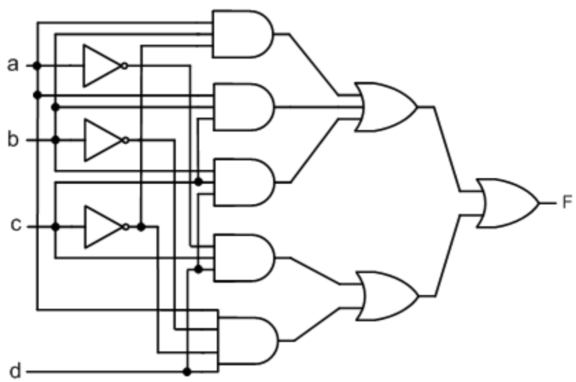
(62.5% reduction!)

Boolean Function Simplification

(Relook at the second example):

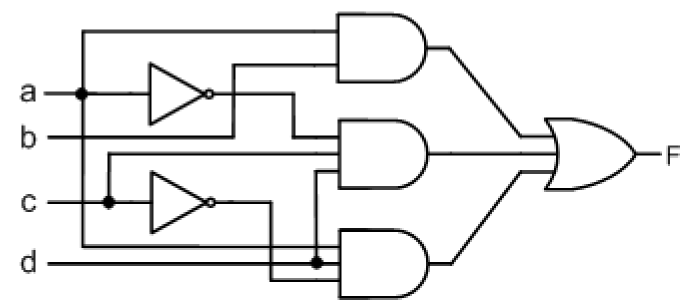
$$F(a,b,c,d) = ab\bar{c} + abc + bcd + \bar{a}cd + a\bar{b}\bar{c}d$$
$$= ab(\bar{c} + c) + bcd + \bar{a}cd + a\bar{b}\bar{c}d \qquad \leftarrow \quad (A + \bar{A} = 1)$$
$$= a\left[b + \bar{b}(\bar{c}d)\right] + bcd + \bar{a}cd \qquad \leftarrow \quad (A + \bar{A} \cdot B = A + B)$$
$$= a(b + \bar{c}d) + bcd + \bar{a}cd$$
$$= [ab + \bar{a}(cd) + b(cd)] + a\bar{c}d \qquad \leftarrow \quad (AB + \bar{A}C + BC = AB + \bar{A}C) \text{ - consensus}$$
$$= ab + \bar{a}cd + a\bar{c}d$$

Before simplification:



gate count = 11

After simplification:



gate count = 6

(45.5% reduction!)

Some Guidelines for Simplification of Boolean Function (in SOP)

Three most used theorems:

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

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

(3) $AB + \bar{A}C + BC = AB + \bar{A}C$ (Consensus)

Apply (1) until it cannot be applied further

Apply (2) until it cannot be applied further

Go back to (1) and then (2) until they can no longer be applied

Apply (3) until it cannot be applied further

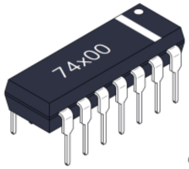
Go back to (1), (2) and then (3) until none of them can be applied

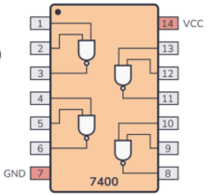
It can then be assumed that the function is simplified

Empirical: the result is usually close to minimal, but may not be the minimal

Cumbersome: other methods are much easier and quicker

Bubble Pushing Rule to Rearrange Logic and Transform (N)AND/(N)OR



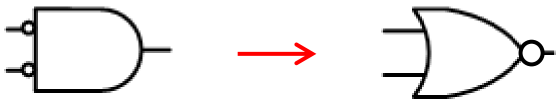


- NAND and NOR are universal gates – They can be used to implement any function!
- Graphic manipulations helps us to do NAND/NOR only implementations :

- bubbles at the input of an AND gate can be “pushed” at its output, and the gate is transformed into a NOR gate (similarly, NAND becomes OR)

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

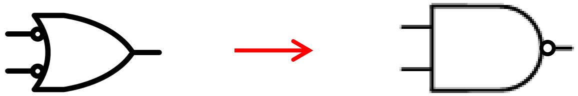
De Morgan’s law



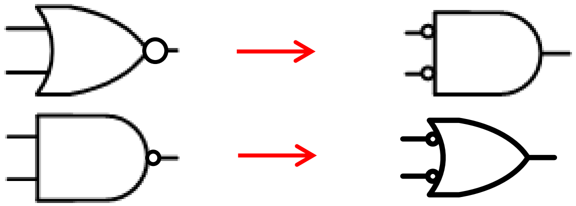
- bubbles at the input of an OR gate can be “pushed” at its output, and the gate is transformed into a NAND gate (similarly, NOR becomes AND)

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

De Morgan’s law



- and vice versa, of course:



alternate gate representations

- two adjacent bubbles gets simplified

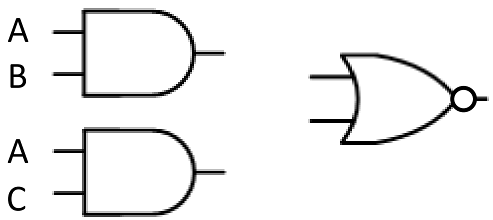
$$F=\overline{\overline{A}}=A$$

Example – Bubble Pushing

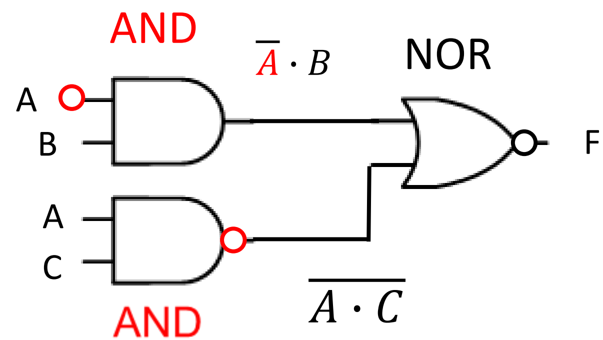
Implement the following Boolean function using only **NOR** gates and inverters.

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

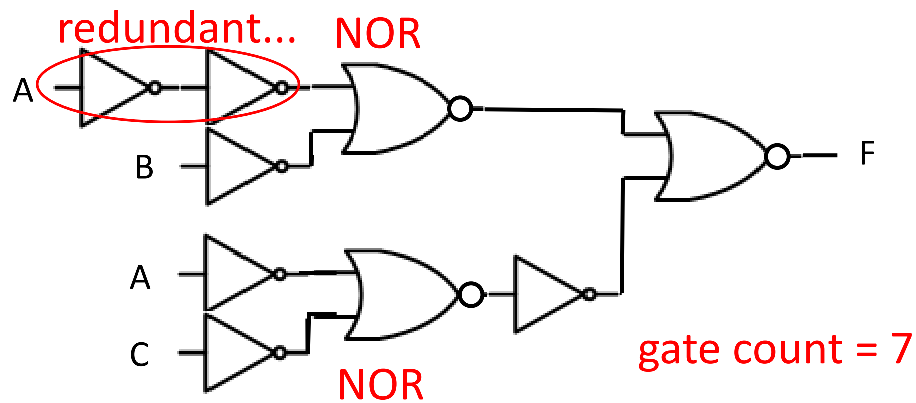
Step 1:



Step 2:
(add the negation where needed for the correct function)



Step 3:
(i) Replace AND gate with NOR gate
(ii) balance the bubbles using inverters to maintain the correct functionality



Positive & Negative Logic

Positive and Negative Logic

Positive and negative logic map the physical voltage (H, L) in a gate correspondence to a logic value

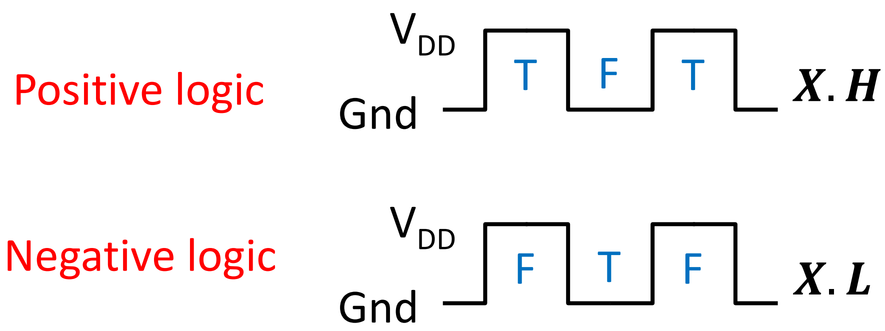
Positive logic (Active high)

- Voltage "**H**" (i.e. V_{DD}) → interpreted as logic "**1**" or "**True**"
- Voltage "**L**" (i.e. Gnd or 0V) → interpreted as logic "**0**" or "**False**"

Negative Logic (Active low)

- Voltage "**L**" (i.e. Gnd or 0V) → interpreted as logic "**1**" or "**True**"
- Voltage "**H**" (i.e. V_{DD}) → interpreted as logic "**0**" or "**False**"

Example:



Conversion of a signal:

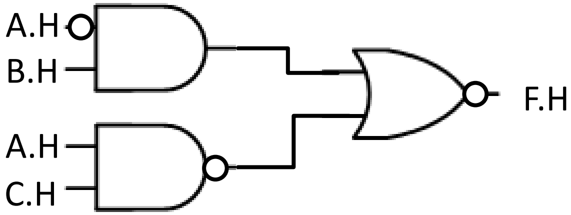
$$\boldsymbol{X.H = \overline{X}.L}$$

$$\boldsymbol{X.L = \overline{X}.H}$$

Example – Implementation in Positive Logic

Implement the following Boolean function in positive logic.

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



Implement the following Boolean function in mixed logic, where A , B are active low signals and C, F are active high.

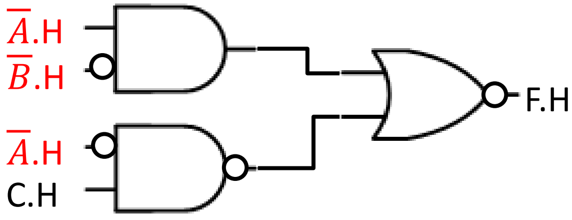
$$F = \overline{\left(\overline{A} \cdot B + \overline{A \cdot C}\right)}$$

Step 1:
(convert active low notation to active high)

A.L → $\overline{A}$.H
B.L → $\overline{B}$.H

A.L→$\overline{A}$.H
C.H

Step 2:
(implement in active high logic)



Summary

Logic gate is a circuit that implement Boolean operations

AND and NAND gates

OR and NOR gates

XOR and XNOR gates

Boolean function implementation using logic gates

Boolean function simplification using algebra postulates and theorems

Positive and negative logic

- Definition
- Physical gates with positive and negative logics
- Physical truth table and logic truth table
- Conversion between positive and negative logics
- Gates with mixed logic