Boolean Algebra and Logic Gates

Boolean Algebra

- ❖ Two-valued Boolean algebra is also called switching algebra
- ❖ A set of two values: *B* = {0, 1}
- ❖ Three basic operations: **AND**, **OR**, and **NOT**
- ❖ The **AND** operator is denoted by a dot (**·**)
  - ✧ $x \cdot y$ or xy is read: x **AND** y
- ❖ The **OR** operator is denoted by a plus (**+**)
  - ✧ $x + y$ is read: x **OR** y
- ❖ The **NOT** operator is denoted by (**'**) or an overbar ($\overline{\ }$).
  - ✧ x' or $\overline{x}$ is the complement of x

Importance of Boolean Algebra

- Our objective is to learn how to design digital circuits
- These circuits use signals with two possible values
- Logic **0** is a **low** voltage signal (around 0 volts)
- Logic **1** is a **high** voltage signal (e.g. 5 or 3.3 volts)
- Having only two logic values (0 and 1) simplifies the implementation of the digital circuit

Postulates of Boolean Algebra

1. Closure: the result of any Boolean operation is in *B* = {0, 1}
2. Identity element with respect to + is 0: $x + 0 = 0 + x = x$

   Identity element with respect to · is 1: $x \cdot 1 = 1 \cdot x = x$
3. Commutative with respect to +: $x + y = y + x$

   Commutative with respect to ·: $x \cdot y \ = \ y \cdot x$
4. · is distributive over +: $x \cdot (y + z) = (x \cdot y) + (x \cdot z)$

   + is distributive over ·: $x + (y \cdot z) = (x + y) \cdot (x + z)$
5. For every x in B, there exists x' in B (called complement of x) such that: $x + x' = 1$ and $x \cdot x' = 0$

AND, OR, and NOT Operators

- The following tables define $x \cdot y$, $x + y$, and x'
- $x \cdot y$ is the **AND** operator
- $x + y$ is the **OR** operator
- x' is the **NOT** operator

x y	x·y
0 0	0
0 1	0
1 0	0
1 1	1

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

x	x'
0	1
1	0

Boolean Functions

- Boolean functions are described by expressions that consist of:
  - Boolean variables, such as: x, y, etc.
  - Boolean constants: 0 and 1
  - Boolean operators: AND (·), OR (+), NOT (')
  - Parentheses, which can be nested
- Example: $f = x(y + w'z)$
  - The dot operator is implicit and need not be written
- Operator precedence: to avoid ambiguity in expressions
  - Expressions within parentheses should be evaluated first
  - The NOT (') operator should be evaluated second
  - The AND (·) operator should be evaluated third
  - The OR (+) operator should be evaluated last

Truth Table

- A truth table can represent a Boolean function
- List all possible combinations of 0's and 1's assigned to variables
- If n variables then 2^n rows
- Example: Truth table for $f = xy' + x'z$

x	y	z	y'	xy'	x'	x'z	f = xy'+x'z
0	0	0	1	0	1	0	0
0	0	1	1	0	1	1	1
0	1	0	0	0	1	0	0
0	1	1	0	0	1	1	1
1	0	0	1	1	0	0	1
1	0	1	1	1	0	0	1
1	1	0	0	0	0	0	0
1	1	1	0	0	0	0	0

DeMorgan's Theorem

- $(x + y)' = x'\,y'$
- $(x\,y)' = x' + y'$

Can be verified Using a Truth Table

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

Identical **Identical**

- Generalized DeMorgan's Theorem:
- $(x_1 + x_2 + \cdots + x_n)' = x_1' \cdot x_2' \cdot \cdots \cdot x_n'$
- $(x_1 \cdot x_2 \cdot \cdots \cdot x_n)' = x_1' + x_2' + \cdots + x_n'$

Complementing Boolean Functions

- What is the complement of $f = x'yz' + xy'z'$?
- Use DeMorgan's Theorem:
  - Complement each variable and constant
  - Interchange AND and OR operators
- So, what is the complement of $f = x'yz' + xy'z'$?

  Answer: $f' = (x + y' + z)(x' + y + z)$
- **Example 2:** Complement $g = (a' + bc)d' + e$
- **Answer:** $g' = (a(b' + c') + d)e'$

Algebraic Manipulation of Expressions

- The objective is to acquire skills in manipulating Boolean expressions, to transform them into simpler form.
- **Example 1:** prove $x + xy = x$ (absorption theorem)
- **Proof:**

$x + xy = x \cdot 1 + xy$	$x \cdot 1 = x$
$= x \cdot (1 + y)$	Distributive $\cdot$ over +
$= x \cdot 1 = x$	$(1 + y) = 1$

- **Example 2:** prove $x + x'y = x + y$ (simplification theorem)
- **Proof:**

$x + x'y = (x + x')(x + y)$	Distributive + over $\cdot$
$= 1 \cdot (x + y)$	$(x + x') = 1$
$= x + y$	

Duality Principle

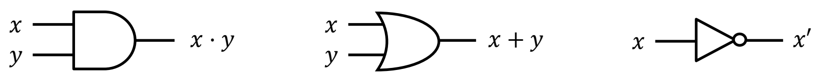
- The **dual** of a Boolean expression can be obtained by:
  - Interchanging AND ($\cdot$) and OR ($+$) operators
  - Interchanging 0's and 1's
- Example: the dual of $x(y + z')$ is $x + yz'$
  - The complement operator does not change
- The properties of Boolean algebra appear in **dual pairs**
  - If a property is proven to be true then its dual is also true

	Property	Dual Property
Identity	$x + 0 = x$	$x \cdot 1 = x$
Complement	$x + x' = 1$	$x \cdot x' = 0$
Distributive	$x\,(y + z) \;= xy + xz$	$x + yz \;= (x + y)(x + z)$

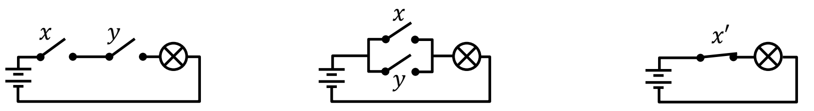
Summary of Boolean Algebra

	Property	Dual Property
Identity	$x + 0 = x$	$x \cdot 1 = x$
Complement	$x + x' = 1$	$x \cdot x' = 0$
Null	$x + 1 = 1$	$x \cdot 0 = 0$
Idempotence	$x + x = x$	$x \cdot x = x$
Involution	$(x')' = x$	
Commutative	$x + y \;=\; y + x$	$x\,y \;=\; y\,x$
Associative	$(x + y) + z \;= x + (y + z)$	$(x\,y)\,z \;= x\,(y\,z)$
Distributive	$x\,(y + z) \;= xy + xz$	$x + yz \;= (x + y)(x + z)$
Absorption	$x + xy \;=\; x$	$x(x + y) \;=\; x$
Simplification	$x + x'y \;=\; x + y$	$x(x' + y) \;=\; xy$
De Morgan	$(x + y)' \;=\; x'\,y'$	$(x\,y)' = x' + y'$

Logic Gates and Symbols



AND gate | OR gate | NOT gate (inverter)



AND: Switches in series
logic 0 is open switch

OR: Switches in parallel
logic 0 is open switch

NOT: Switch is normally
closed when x is 0

❖ In the earliest computers, relays were used as mechanical switches controlled by electricity (coils)

❖ Today, tiny transistors are used as electronic switches that implement the logic gates (CMOS technology)

Truth Table and Logic Diagram

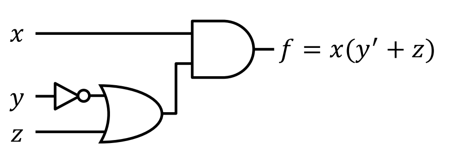
❖ Given the following logic function: $f = x(y' + z)$

❖ Draw the corresponding truth table and logic diagram

Truth Table

x y z	y'+z	f = x(y'+z)
0 0 0	1	0
0 0 1	1	0
0 1 0	0	0
0 1 1	1	0
1 0 0	1	1
1 0 1	1	1
1 1 0	0	0
1 1 1	1	1

Logic Diagram



Truth Table and Logic Diagram describe the same function f. Truth table is unique, but logic expression and logic diagram are not. This gives flexibility in implementing logic functions.

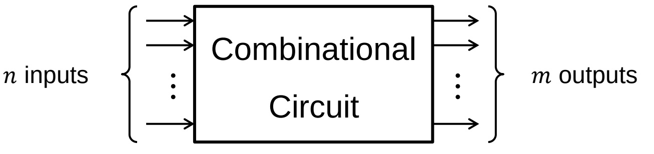
Combinational Circuit

- A combinational circuit is a block of logic gates having:

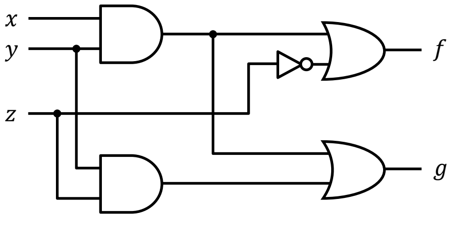
  n inputs: $x_1, x_2, \dots, x_n$

  m outputs: $f_1, f_2, \dots, f_m$

- Each output is a function of the input variables
- Each output is determined from present combination of inputs
- Combination circuit performs operation specified by logic gates



Example of a Simple Combinational Circuit



- The above circuit has:
  - Three inputs: x, y, and z
  - Two outputs: f and g
- What are the logic expressions of f and g ?
- **Answer:** $f = xy + z'$

  $g = xy + yz$

From Truth Tables to Gate Implementation

❖ Given the truth table of a Boolean function f, how do we implement the truth table using logic gates?

Truth Table

x y z	f
0 0 0	0
0 0 1	0
0 1 0	1
0 1 1	1
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	1

What is the logic expression of f?

What is the gate implementation of f?

To answer these questions, we need to define Minterms and Maxterms

Minterms and Maxterms

❖ **Minterms** are AND terms with every variable present in either true or complement form

❖ **Maxterms** are OR terms with every variable present in either true or complement form

Minterms and Maxterms for 2 variables x and y

x	y	index	Minterm	Maxterm
0	0	0	$m_0 = x'y'$	$M_0 = x + y$
0	1	1	$m_1 = x'y$	$M_1 = x + y'$
1	0	2	$m_2 = xy'$	$M_2 = x' + y$
1	1	3	$m_3 = xy$	$M_3 = x' + y'$

❖ For n variables, there are 2^n Minterms and Maxterms

Minterms and Maxterms for 3 Variables

x	y	z	index	Minterm	Maxterm
0	0	0	0	$m_0 = x'y'z'$	$M_0 = x + y + z$
0	0	1	1	$m_1 = x'y'z$	$M_1 = x + y + z'$
0	1	0	2	$m_2 = x'yz'$	$M_2 = x + y' + z$
0	1	1	3	$m_3 = x'yz$	$M_3 = x + y' + z'$
1	0	0	4	$m_4 = xy'z'$	$M_4 = x' + y + z$
1	0	1	5	$m_5 = xy'z$	$M_5 = x' + y + z'$
1	1	0	6	$m_6 = xyz'$	$M_6 = x' + y' + z$
1	1	1	7	$m_7 = xyz$	$M_7 = x' + y' + z'$

Maxterm M_i is the **complement** of Minterm m_i

$M_i = m_i{}'$ and $m_i = M_i{}'$

Purpose of the Index

- Minterms and Maxterms are designated with an index
- The **index** for the Minterm or Maxterm, expressed as a **binary number**, is used to determine whether the variable is shown in the true or complemented form
- For Minterms:
  - '**1**' means the variable is **Not Complemented**
  - '**0**' means the variable is **Complemented**
- For Maxterms:
  - '**0**' means the variable is **Not Complemented**
  - '**1**' means the variable is **Complemented**

Sum-Of-Minterms (SOM) Canonical Form

Truth Table

x y z	f	Minterm
0 0 0	0	
0 0 1	0	
0 1 0	1	$m_2 = x'yz'$
0 1 1	1	$m_3 = x'yz$
1 0 0	0	
1 0 1	1	$m_5 = xy'z$
1 1 0	0	
1 1 1	1	$m_7 = xyz$

Sum of Minterm entries that evaluate to '**1**'

Focus on the '**1**' entries

$$f = m_2 + m_3 + m_5 + m_7$$

$$f = \sum(2, 3, 5, 7)$$

$$f = x'yz' + x'yz + xy'z + xyz$$

Examples of Sum-Of-Minterms

- $f(a, b, c, d) = \sum(2, 3, 6, 10, 11)$
- $f(a, b, c, d) = m_2 + m_3 + m_6 + m_{10} + m_{11}$
- $f(a, b, c, d) = a'b'cd' + a'b'cd + a'bcd' + ab'cd' + ab'cd$
- $g(a, b, c, d) = \sum(0, 1, 12, 15)$
- $g(a, b, c, d) = m_0 + m_1 + m_{12} + m_{15}$
- $g(a, b, c, d) = a'b'c'd' + a'b'c'd + abc'd' + abcd$

Product-Of-Maxterms (POM) Canonical Form

Truth Table

x y z	f	Maxterm
0 0 0	0	$M_0 = x + y + z$
0 0 1	0	$M_1 = x + y + z'$
0 1 0	1	
0 1 1	1	
1 0 0	0	$M_4 = x' + y + z$
1 0 1	1	
1 1 0	0	$M_6 = x' + y' + z$
1 1 1	1	

Product of Maxterm entries that evaluate to '**0**'

Focus on the '**0**' entries

$$f = M_0 \cdot M_1 \cdot M_4 \cdot M_6$$

$$f = \prod(0, 1, 4, 6)$$

$$f = (x + y + z)(x + y + z')(x' + y + z)(x' + y' + z)$$

Examples of Product-Of-Maxterms

- $f(a, b, c, d) = \prod(1, 3, 11)$
- $f(a, b, c, d) = M_1 \cdot M_3 \cdot M_{11}$
- $f(a, b, c, d) = (a + b + c + d')(a + b + c' + d')\ (a' + b + c' + d')$
- $g(a, b, c, d) = \prod(0, 5, 13)$
- $g(a, b, c, d) = M_0 \cdot M_5 \cdot M_{13}$
- $f(a, b, c, d) = (a + b + c + d)(a + b' + c + d')\ (a' + b' + c + d')$

Conversions between Canonical Forms

❖ The same Boolean function f can be expressed in two ways:

- ✧ Sum-of-Minterms $f = m_0 + m_2 + m_3 + m_5 + m_7 = \sum(0,2,3,5,7)$
- ✧ Product-of-Maxterms $f = M_1 \cdot M_4 \cdot M_6 = \prod(1,4,6)$

Truth Table

x y z	f	Minterms	Maxterms
0 0 0	1	$m_0 = x'y'z'$	
0 0 1	0		$M_1 = x + y + z'$
0 1 0	1	$m_2 = x'yz'$	
0 1 1	1	$m_3 = x'yz$	
1 0 0	0		$M_4 = x' + y + z$
1 0 1	1	$m_5 = xy'z$	
1 1 0	0		$M_6 = x' + y' + z$
1 1 1	1	$m_7 = xyz$	

To convert from one canonical form to another, interchange the symbols $\sum$ and $\prod$ and list those numbers missing from the original form.

Function Complement

Truth Table

x y z	f	f'
0 0 0	1	0
0 0 1	0	1
0 1 0	1	0
0 1 1	1	0
1 0 0	0	1
1 0 1	1	0
1 1 0	0	1
1 1 1	1	0

Given a Boolean function f

$$f(x,y,z) = \sum(0,2,3,5,7) = \prod(1,4,6)$$

Then, the complement f' of function f

$$f'(x,y,z) = \prod(0,2,3,5,7) = \sum(1,4,6)$$

The complement of a function expressed by a Sum of Minterms is the Product of Maxterms with the same indices. Interchange the symbols $\sum$ and $\prod$, but keep the same list of indices.

Summary of Minterms and Maxterms

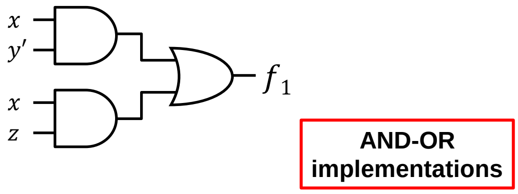
- There are 2^n Minterms and Maxterms for Boolean functions with n variables, indexed from 0 to $2^n - 1$
- Minterms correspond to the **1-entries** of the function
- Maxterms correspond to the **0-entries** of the function
- Any Boolean function can be expressed as a Sum-of-Minterms and as a Product-of-Maxterms
- For a Boolean function, given the list of Minterm indices one can determine the list of Maxterms indices (and vice versa)
- The complement of a Sum-of-Minterms is a Product-of-Maxterms with the same indices (and vice versa)

Sum-of-Products and Products-of-Sums

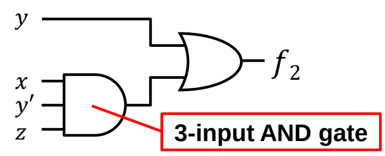
- Canonical forms contain a larger number of literals
  - Because the Minterms (and Maxterms) must contain, by definition, all the variables either complemented or not
- Another way to express Boolean functions is in **standard** form
- Two standard forms: **Sum-of-Products** and **Product-of -Sums**
- Sum of Products (**SOP**)
  - Boolean expression is the **ORing** (sum) of **AND terms** (products)
  - Examples: $f_1 = xy' + xz$ $\qquad f_2 = y + xy'z$
- Products of Sums (**POS**)
  - Boolean expression is the **ANDing** (product) of **OR terms** (sums)
  - Examples: $f_3 = (x + z)(x' + y')$ $\qquad f_4 = x(x' + y' + z)$

Two-Level Gate Implementation

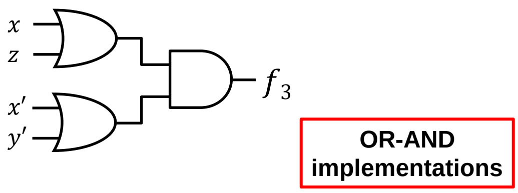
$f_1 = xy' + xz$



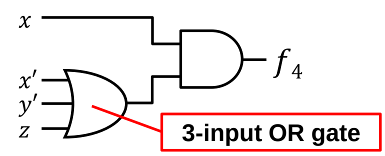
$f_2 = y + xy'z$



$f_3 = (x + z)(x' + y')$



$f_4 = x(x' + y' + z)$

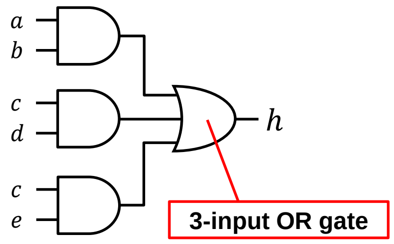


Two-Level vs. Three-Level Implementation

- $h = ab + cd + ce$ (6 literals) is a sum-of-products
- h may also be written as: $h = ab + c(d + e)$ (5 literals)
- However, $h = ab + c(d + e)$ is a non-standard form
  - $h = ab + c(d + e)$ is not a sum-of-products nor a product-of-sums

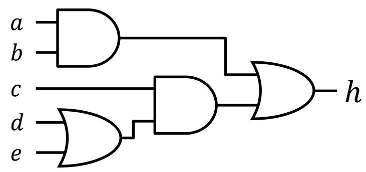
2-level implementation

$h = ab + cd + ce$



3-level implementation

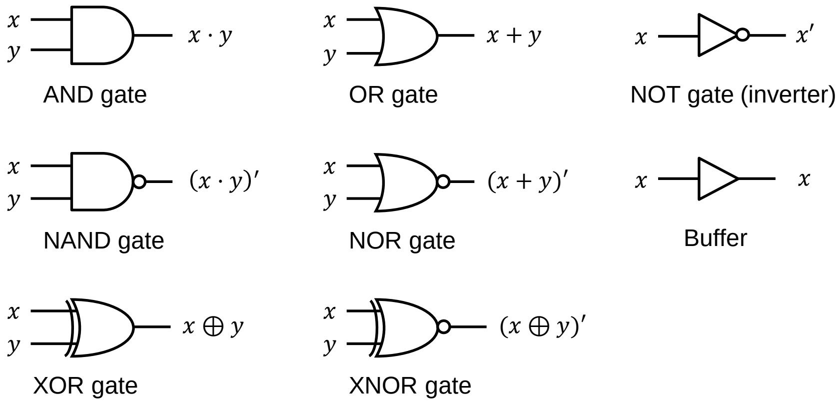
$h = ab + c(d + e)$



Additional Logic Gates and Symbols

❖ Why?

- ✧ Low cost implementation
- ✧ Useful in implementing Boolean functions

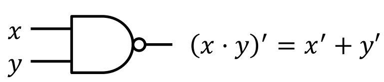


NAND Gate

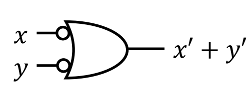
❖ The NAND gate has the following symbol and truth table

❖ NAND represents **N**OT **AND**

❖ The small bubble circle represents the invert function



NAND gate



Another symbol for NAND

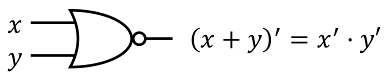
x	y	NAND
0	0	1
0	1	1
1	0	1
1	1	0

NOR Gate

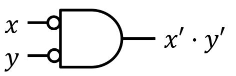
❖ The NOR gate has the following symbol and truth table

❖ NOR represents **NOT OR**

❖ The small bubble circle represents the invert function



NOR gate



Another symbol for NOR

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

The NAND Gate is Universal

❖ NAND gates can implement any Boolean function

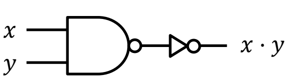
❖ NAND gates can be used as inverters, or to implement AND/OR

❖ A single-input NAND gate is an inverter

x NAND x = $(x \cdot x)' = x'$

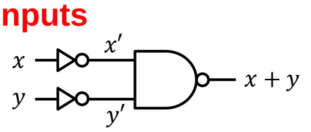
❖ AND is equivalent to NAND with **inverted output**

$(x$ NAND $y)'$ = $((x \cdot y)')' = x \cdot y$ (AND)



❖ OR is equivalent to NAND with **inverted inputs**

$(x'$ NAND $y')$ = $(x' \cdot y')' = x + y$ (OR)



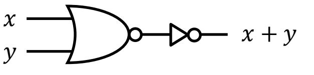
The NOR Gate is also Universal

- NOR gates can implement any Boolean function
- NOR gates can be used as inverters, or to implement AND/OR
- A single-input NOR gate is an inverter

  x NOR x = $(x + x)'$ = x'

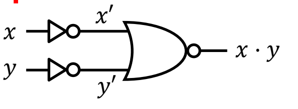
- OR is equivalent to NOR with **inverted output**

  (x NOR y)$'$ = $((x + y)')'$ = $x + y$ (OR)



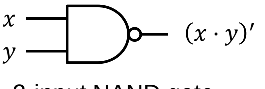
- AND is equivalent to NOR with **inverted inputs**

  (x' NOR y') = $(x' + y')'$ = $x \cdot y$ (AND)

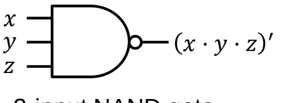


Multiple-Input NAND / NOR Gates

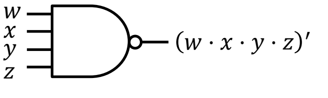
NAND/NOR gates can have multiple inputs, similar to AND/OR gates



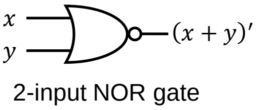
2-input NAND gate



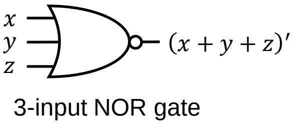
3-input NAND gate



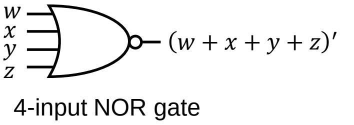
4-input NAND gate



2-input NOR gate



3-input NOR gate



4-input NOR gate

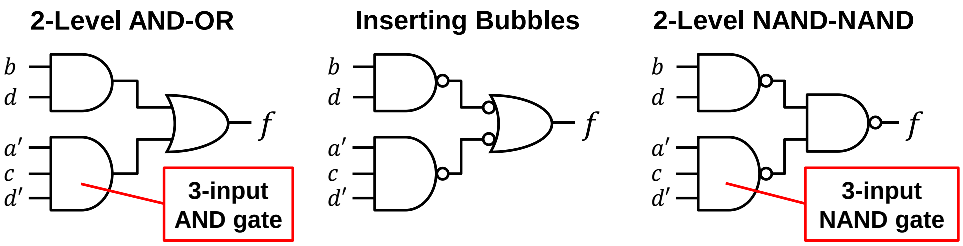
Note: a 3-input NAND is a single gate, NOT a combination of two 2-input gates.
The same can be said about other multiple-input NAND/NOR gates.

NAND – NAND Implementation

❖ Consider the following sum-of-products expression:

$f = bd + a'cd'$

❖ A 2-level **AND-OR** circuit can be converted easily to a 2-level **NAND-NAND implementation**



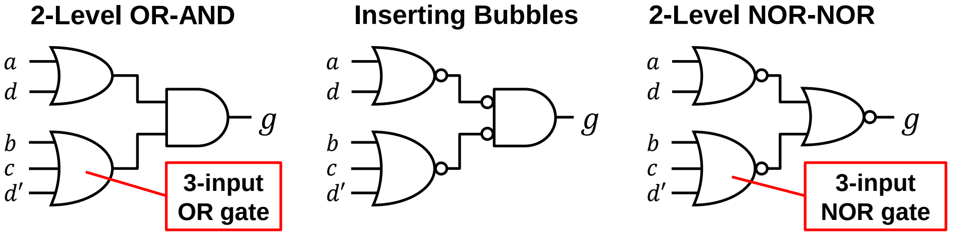
Two successive bubbles on same line cancel each other

NOR – NOR Implementation

❖ Consider the following product-of-sums expression:

$g = (a + d)(b + c + d')$

❖ A 2-level **OR-AND** circuit can be converted easily to a 2-level **NOR-NOR implementation**



Two successive bubbles on same line cancel each other

Exclusive OR / Exclusive NOR

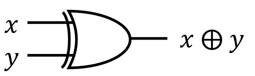
❖ Exclusive OR (XOR) is an important Boolean operation used extensively in logic circuits

❖ Exclusive NOR (XNOR) is the complement of XOR

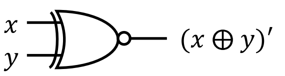
x y	XOR
0 0	0
0 1	1
1 0	1
1 1	0

x y	XNOR
0 0	1
0 1	0
1 0	0
1 1	1

XNOR is also known as **equivalence**



XOR gate



XNOR gate

XOR / XNOR Functions

❖ The XOR function is: $x \oplus y = xy' + x'y$

❖ The XNOR function is: $(x \oplus y)' = xy + x'y'$

❖ XOR and XNOR gates are complex

- ✧ Can be implemented as a true gate, or by
- ✧ Interconnecting other gate types

❖ XOR and XNOR gates do not exist for more than two inputs

- ✧ For 3 inputs, use two XOR gates
- ✧ The cost of a 3-input XOR gate is greater than the cost of two XOR gates

❖ Uses for XOR and XNOR gates include:

- ✧ Adders, subtractors, multipliers, counters, incrementers, decrementers
- ✧ Parity generators and checkers

XOR and XNOR Properties

- $x \oplus 0 = x$ $\quad$ $x \oplus 1 = x'$
- $x \oplus x = 0$ $\quad$ $x \oplus x' = 1$
- $x \oplus y = y \oplus x$
- $x' \oplus y' = x \oplus y$
- $(x \oplus y)' = x' \oplus y = x \oplus y'$

XOR and XNOR are **associative** operations

- $(x \oplus y) \oplus z = x \oplus (y \oplus z) = x \oplus y \oplus z$
- $\big((x \oplus y)' \oplus z\big)' = (x \oplus (y \oplus z)')' = x \oplus y \oplus z$