# Introduction to Logic Gates

- We will introduce Boolean algebra and logic gates
- Logic gates are the building blocks of digital circuits

# Logic Variables

- Different names for the same thing
  - Logic variables
  - Binary variables
  - Boolean variables
- Can only take on 2 values, e.g.,
  - TRUE or False
  - ON or OFF
  - 1 or 0

# Logic Variables

- In electronic circuits the two values can be represented by e.g.,

    – High voltage for a 1

    – Low voltage for a 0

- Note that since only 2 voltage levels are used, the circuits have greater immunity to electrical noise

# Uses of Simple Logic

- Example – Heating Boiler
  - If chimney is not blocked and the house is cold and the pilot light is lit, then open the main fuel valve to start boiler.

    $b$ = chimney blocked

    $c$ = house is cold

    $p$ = pilot light lit

    $v$ = open fuel valve
  - So in terms of a logical (Boolean) expression

    $v$ = (NOT $b$) AND $c$ AND $p$

# Logic Gates

- Basic logic circuits with one or more inputs and one output are known as *gates*
- *Gates* are used as the building blocks in the design of more complex digital logic circuits

# Representing Logic Functions

- There are several ways of representing logic functions:
  - Symbols to represent the gates
  - Truth tables
  - Boolean algebra
- We will now describe commonly used gates
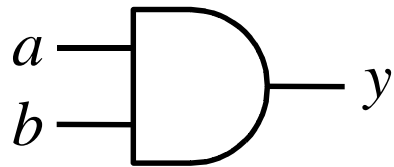
# NOT Gate

Symbol          Truth-table        Boolean

$$y = \overline{a}$$

| $a$ | $y$ |
|-----|-----|
| 0   | 1   |
| 1   | 0   |
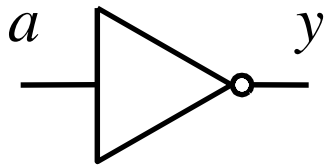
- A NOT gate is also called an 'inverter'
- $y$ is only TRUE if $a$ is FALSE
- Circle (or 'bubble') on the output of a gate implies that it as an inverting (or complemented) output

# AND Gate

| Symbol | Truth-table | Boolean |
|---|---|---|



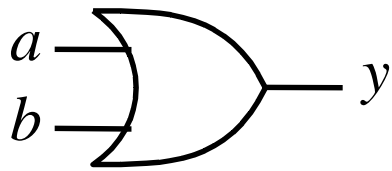| $a$ | $b$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

$$y = a.b$$

- $y$ is only TRUE only if $a$ is TRUE and $b$ is TRUE
- In Boolean algebra AND is represented by a dot .

# OR Gate

Symbol

Truth-table

Boolean

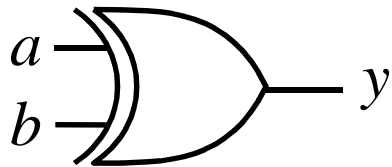| $a$ | $b$ | $y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

$$y = a + b$$

$a$ —

$b$ —

$y$

- $y$ is TRUE if $a$ is TRUE or $b$ is TRUE (or both)
- In Boolean algebra OR is represented by a plus sign $+$

# EXCLUSIVE OR (XOR) Gate

Symbol               Truth-table          Boolean

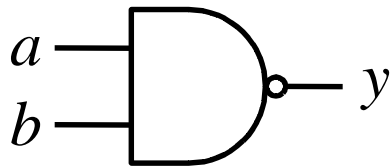| $a$ | $b$ | $y$ |
|-----|-----|-----|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

$$y = a \oplus b$$

- $y$ is TRUE if $a$ is TRUE or $b$ is TRUE (but not both)
- In Boolean algebra XOR is represented by an $\oplus$ sign

# NOT AND (NAND) Gate
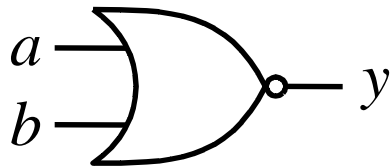
Symbol                Truth-table              Boolean

$$y = \overline{a.b}$$

| $a$ | $b$ | $y$ |
|-----|-----|-----|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

- $y$ is TRUE if $a$ is FALSE or $b$ is FALSE (or both)
- $y$ is FALSE only if $a$ is TRUE and $b$ is TRUE

# NOT OR (NOR) Gate

### Symbol

$a$ —⫐
$b$ —    ⟩o— $y$

### Truth-table

| $a$ | $b$ | $y$ |
|-----|-----|-----|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

### Boolean

$$y = \overline{a + b}$$

- $y$ is TRUE only if $a$ is FALSE and $b$ is FALSE
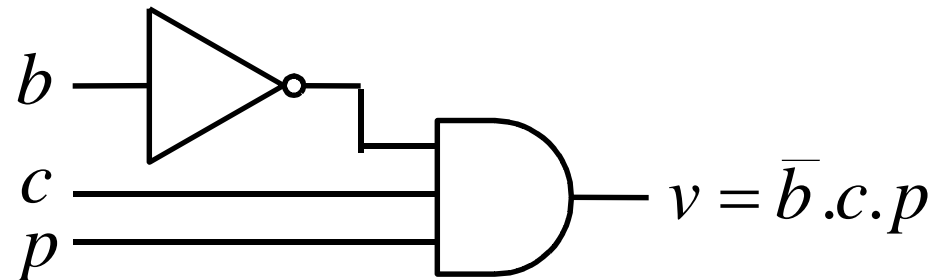- $y$ is FALSE if $a$ is TRUE or $b$ is TRUE (or both)

# Boiler Example

- If chimney is not blocked and the house is cold and the pilot light is lit, then open the main fuel valve to start boiler.

$b$ = chimney blocked       $c$ = house is cold

$p$ = pilot light lit       $v$ = open fuel valve

$$v = \bar{b}.c.p$$

# Boolean Algebra

- In this section we will introduce the laws of Boolean Algebra

- We will then see how it can be used to design *combinational logic* circuits

- Combinational logic circuits do not have an internal stored state, i.e., they have no memory. Consequently the output is solely a function of the current inputs.

- Later, we will study circuits having a stored internal state, i.e., sequential logic circuits.

# Boolean Algebra

OR

$$a + 0 = a$$
$$a + a = a$$
$$a + 1 = 1$$
$$a + \overline{a} = 1$$

AND

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

- AND takes precedence over OR, e.g.,

$$a.b + c.d = (a.b) + (c.d)$$

# Boolean Algebra

- Commutation

$$a + b = b + a$$
$$a.b = b.a$$

- Association

$$(a + b) + c = a + (b + c)$$
$$(a.b).c = a.(b.c)$$

- Distribution

$$a.(b + c + \ldots) = (a.b) + (a.c) + \ldots$$
$$a + (b.c\ldots) = (a + b).(a + c).\ldots \quad \text{NEW}$$

- Absorption

$$a + (a.c) = a \quad \text{NEW}$$
$$a.(a + c) = a \quad \text{NEW}$$

# Boolean Algebra - Examples

Show

$$a.(\bar{a} + b) = a.b$$

$$a.(\bar{a} + b) = a.\bar{a} + a.b = 0 + a.b = a.b$$

Show

$$a + (\bar{a}.b) = a + b$$

$$a + (\bar{a}.b) = (a + \bar{a}).(a + b) = 1.(a + b) = a + b$$

# Boolean Algebra

- A useful technique is to expand each term until it includes one instance of each variable (or its compliment). It may be possible to simplify the expression by cancelling terms in this expanded form e.g., to prove the absorption rule:

$$a + a.b = a$$

$$a.b + a.\bar{b} + a.b = a.b + a.\bar{b} = a.(b + \bar{b}) = a.1 = a$$

# Boolean Algebra - Example

Simplify

$x.y + \bar{y}.z + x.z + x.y.z$

$x.y.z + x.y.\bar{z} + x.\bar{y}.z + \bar{x}.\bar{y}.z + x.y.z + x.\bar{y}.z + x.y.z$

$x.y.z + x.y.\bar{z} + x.\bar{y}.z + \bar{x}.\bar{y}.z$

$x.y.(z + \bar{z}) + \bar{y}.z.(x + \bar{x})$

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

$x.y + \bar{y}.z$

# DeMorgan's Theorem

$$\overline{a + b + c + \ldots} = \overline{a}.\overline{b}.\overline{c}. \ldots$$
$$\overline{a.b.c. \ldots} = \overline{a} + \overline{b} + \overline{c} + \ldots$$

- In a simple expression like $a + b + c$ (or $a.b.c$) simply change all operators from OR to AND (or vice versa), complement each term (put a bar over it) and then complement the whole expression, i.e.,

$$a + b + c + \ldots = \overline{\overline{a}.\overline{b}.\overline{c}. \ldots}$$
$$a.b.c. \ldots = \overline{\overline{a} + \overline{b} + \overline{c} + \ldots}$$

# DeMorgan's Theorem

- For 2 variables we can show $\overline{a+b} = \overline{a}.\overline{b}$ and $\overline{a.b} = \overline{a} + \overline{b}$ using a truth table.

| $a$ | $b$ | $\overline{a+b}$ | $\overline{a.b}$ | $\overline{a}$ | $\overline{b}$ | $\overline{a}.\overline{b}$ | $\overline{a}+\overline{b}$ |
|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |

- Extending to more variables by induction

$$\overline{a+b+c} = \overline{(a+b)}.\overline{c} = (\overline{a}.\overline{b}).\overline{c} = \overline{a}.\overline{b}.\overline{c}$$

# DeMorgan's Examples

- Simplify $a.\overline{b} + a.\overline{(b + c)} + b.\overline{(b + c)}$

$$= a.\overline{b} + a.\overline{b}.\overline{c} + b.\overline{b}.\overline{c} \quad \text{(DeMorgan)}$$

$$= a.\overline{b} + a.\overline{b}.\overline{c} \quad \text{(b.\overline{b} = 0)}$$

$$= a.\overline{b} \quad \text{(absorbtion)}$$

# DeMorgan's Examples

- Simplify $(a.b.(c + \overline{b.d}) + \overline{a.b}).c.d$

$$= (a.b.(c + \overline{b} + \overline{d}) + \overline{a} + \overline{b}).c.d \qquad \text{(De Morgan)}$$

$$= (a.b.c + a.b.\overline{b} + a.b.\overline{d} + \overline{a} + \overline{b}).c.d \qquad \text{(distribute)}$$

$$= (a.b.c + a.b.\overline{d} + \overline{a} + \overline{b}).c.d \qquad (a.b.\overline{b} = 0)$$

$$= a.b.c.d + a.b.\overline{d}.c.d + \overline{a}.c.d + \overline{b}.c.d \qquad \text{(distribute)}$$

$$= a.b.c.d + \overline{a}.c.d + \overline{b}.c.d \qquad (a.b.\overline{d}.c.d = 0)$$

$$= (a.b + \overline{a} + \overline{b}).c.d \qquad \text{(distribute)}$$

$$= (a.b + \overline{a.b}).c.d \qquad \text{(DeMorgan)}$$

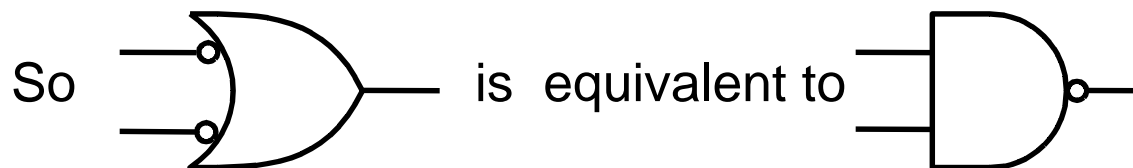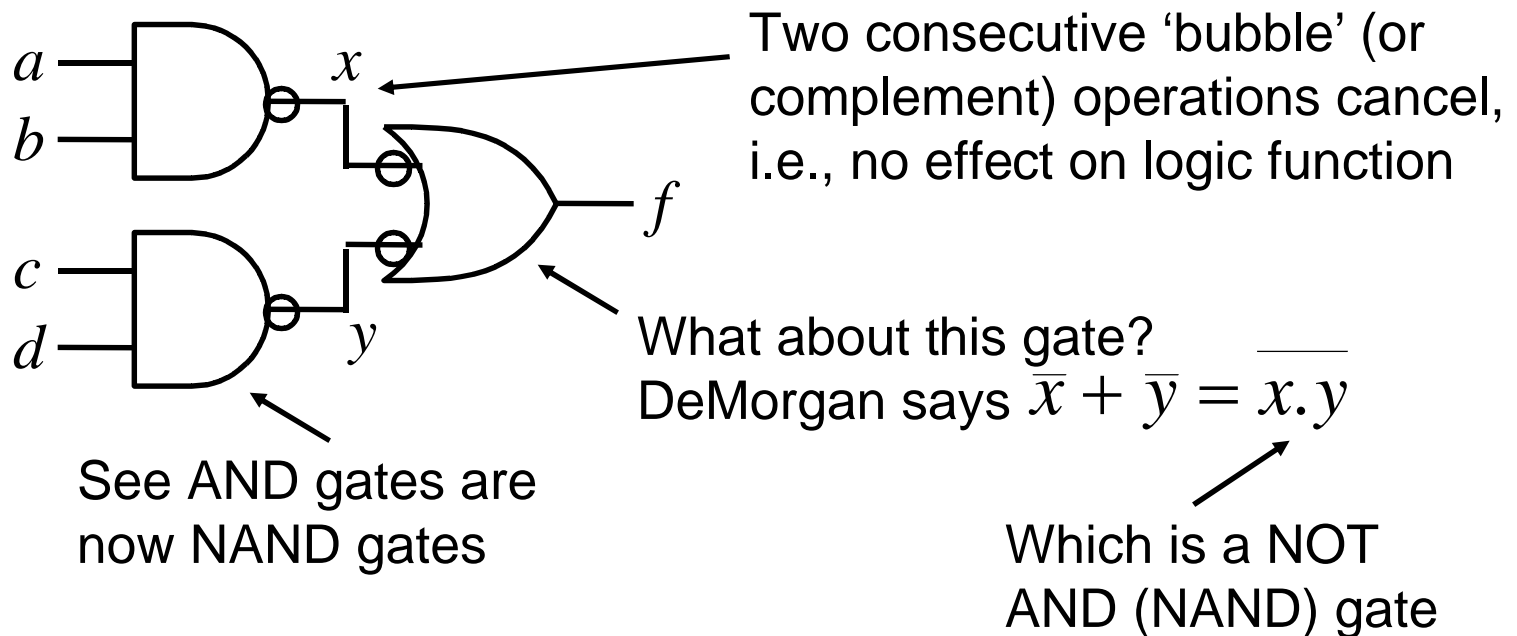$$= c.d \qquad (a.b + \overline{a.b} = 1)$$

# DeMorgan's in Gates

- To implement the function $f = a.b + c.d$ we can use AND and OR gates
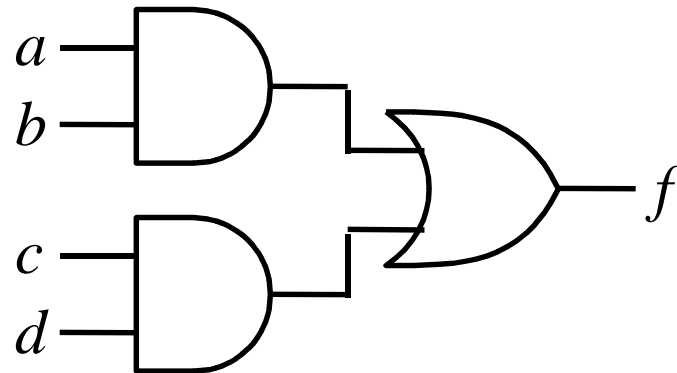


- However, sometimes we only wish to use NAND or NOR gates, since they are usually simpler and faster
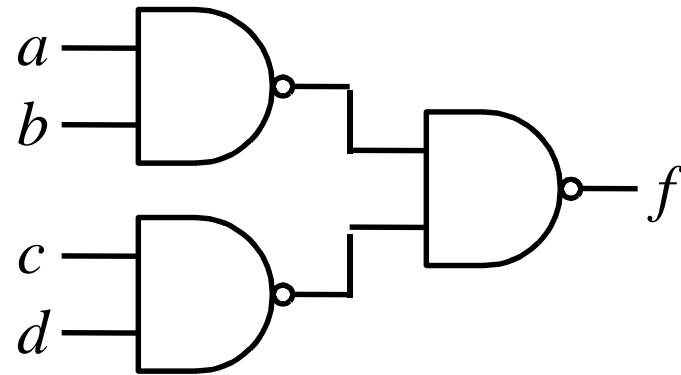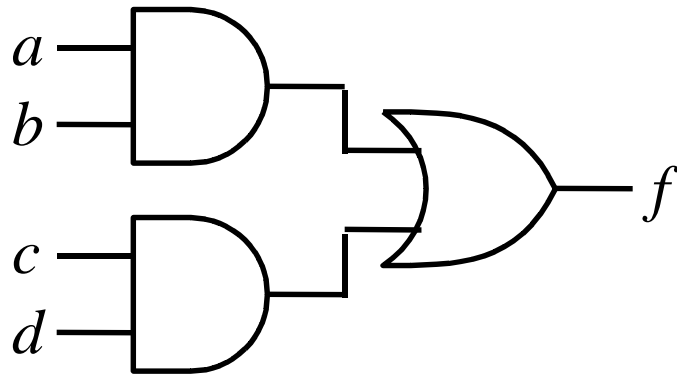
# DeMorgan's in Gates

- To do this we can use 'bubble' logic



Two consecutive 'bubble' (or complement) operations cancel, i.e., no effect on logic function

What about this gate?
DeMorgan says $\overline{x} + \overline{y} = \overline{x.y}$

See AND gates are now NAND gates

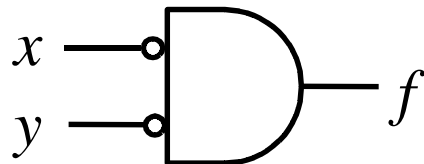Which is a NOT AND (NAND) gate

So ⟩ is equivalent to ⟩

# DeMorgan's in Gates

- So the previous function can be built using 3 NAND gates

# DeMorgan's in Gates

- Similarly, applying 'bubbles' to the input of an AND gate yields

$x$ ⎯ $f$
$y$ ⎯

Which is a NOT OR (NOR) gate

What about this gate?
DeMorgan says $\overline{x}.\overline{y} = \overline{x + y}$

So [gate] is equivalent to [gate]
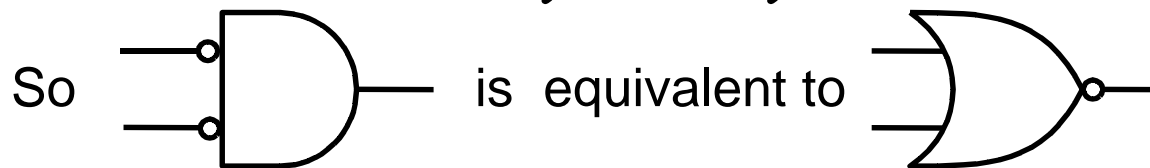
- Useful if trying to build using NOR gates