Logic Gates

Md. Rokanujjaman
Professor
Dept. of CSE, RU

Introduction

- Digital systems are concerned with digital signals
- Digital signals can take many forms
- Here we will concentrate on **binary signals** since these are the most common form of digital signals
  - can be used individually
    - perhaps to represent a single binary quantity or the state of a single switch
  - can be used in combination
    - to represent more complex quantities

Boolean Constants and Variables

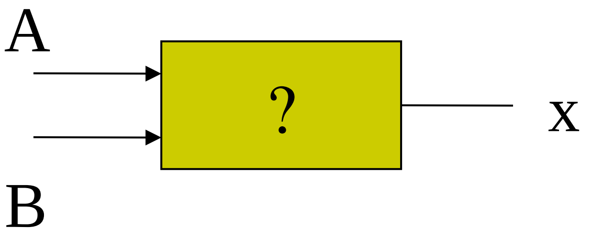
- Boolean 0 and 1 do not represent actual numbers but instead represent the state, or logic level.

Logic 0	Logic 1
False	True
Off	On
Low	High
No	Yes
Open switch	Closed switch

Truth Tables

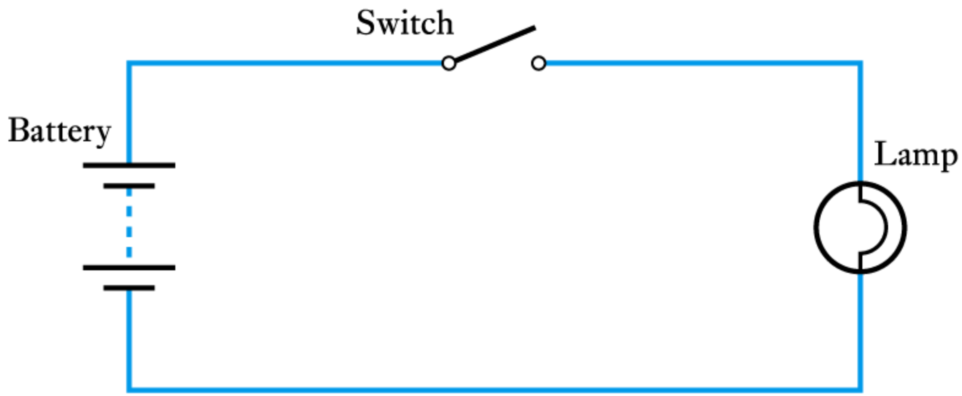
- A truth table is a means for describing how a logic circuit's output depends on the logic levels present at the circuit's inputs.

Inputs		Output
A	B	x
0	0	1
0	1	0
1	0	1
1	1	0



Binary Quantities and Variables

- A **binary quantity** is one that can take only 2 states



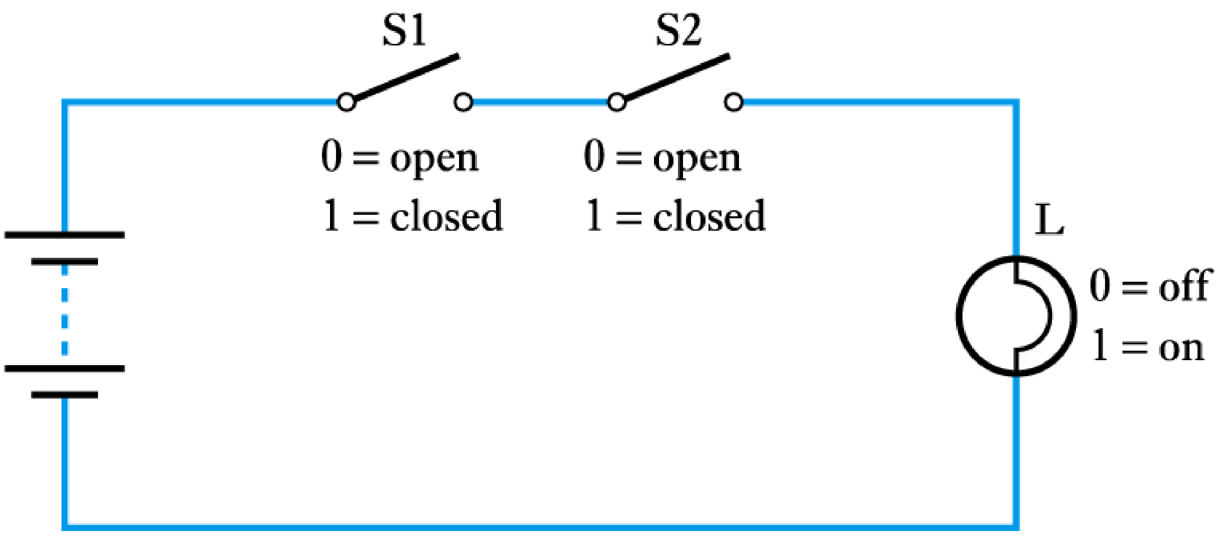
A simple binary arrangement

S	*L*
OPEN	OFF
CLOSED	ON

S	*L*
0	0
1	1

A truth table

- A binary arrangement with two switches in series



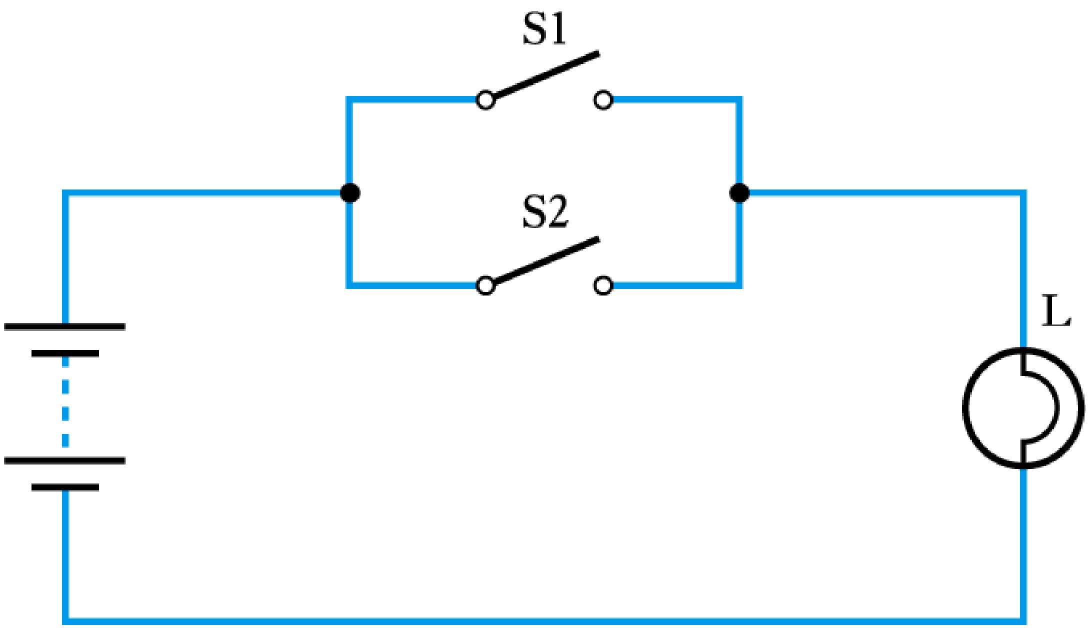
(a) Circuit

S1	S2	L
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

L = *S1* AND *S2*

- A binary arrangement with two switches in parallel



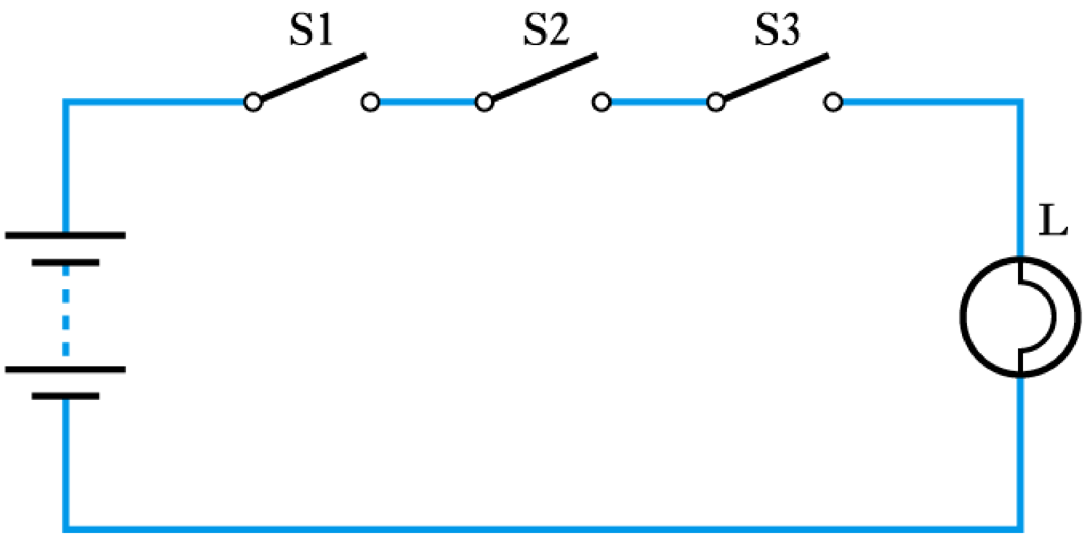
(a) Circuit

S1	S2	L
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

L = *S1* OR *S2*

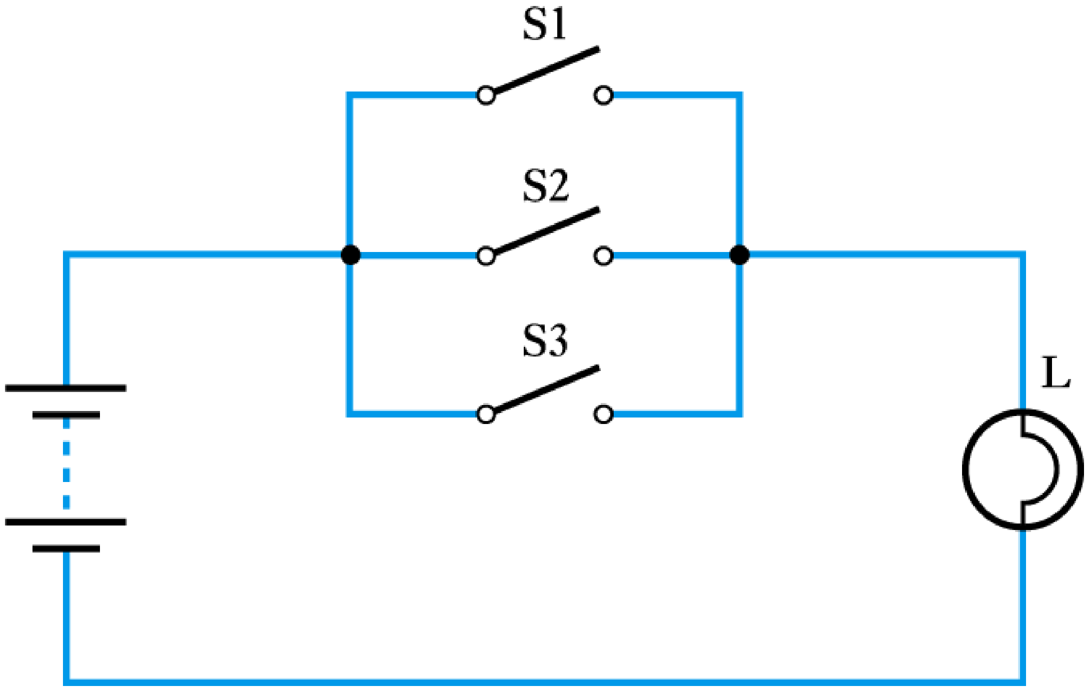
- Three switches in series



S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	1

L = *S1* AND *S2* AND *S3*

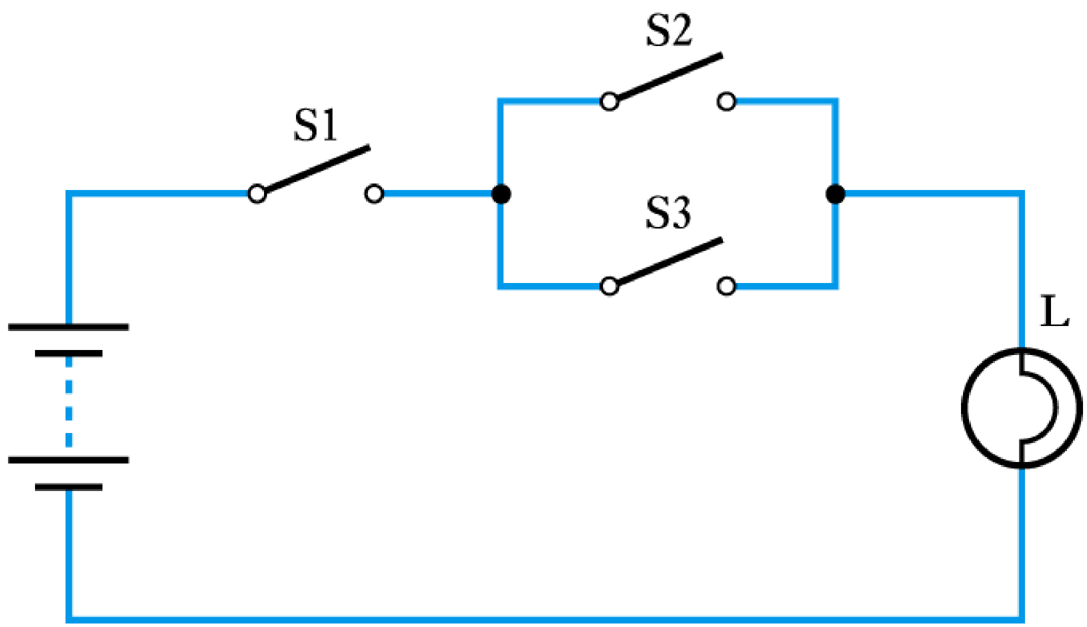
- Three switches in parallel



S1	S2	S3	L
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

$L = S1$ OR $S2$ OR $S3$

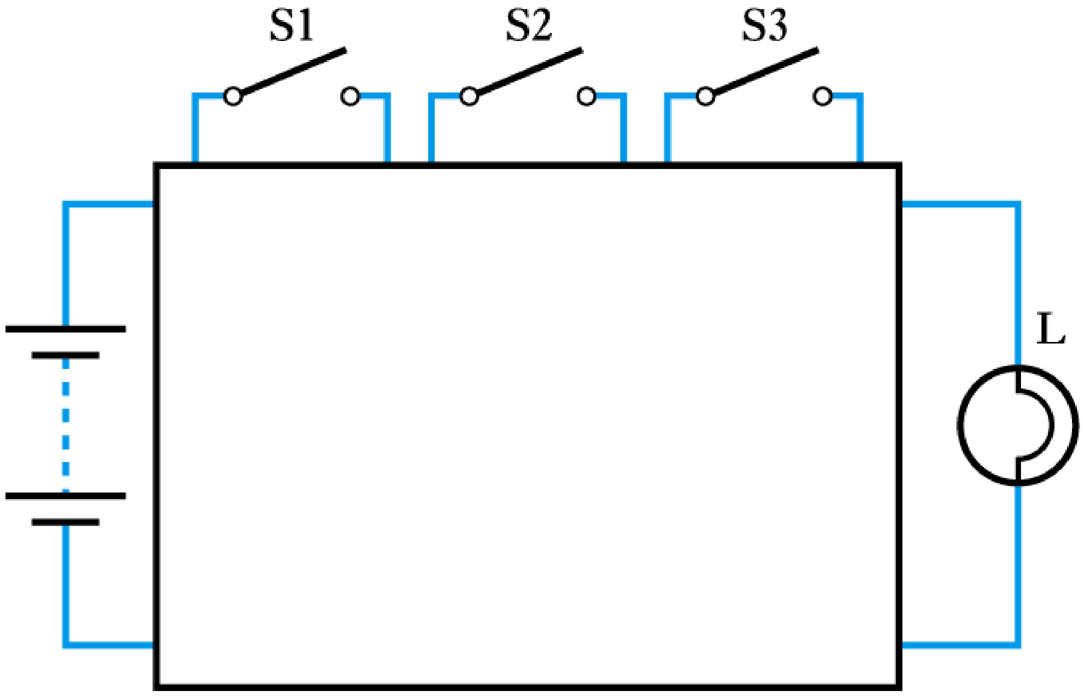
- A series/parallel arrangement



S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

L = *S1* AND (*S2* OR *S3*)

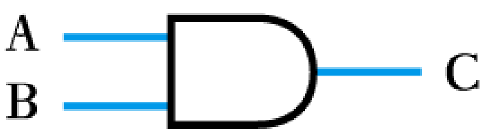
- Representing an unknown network



Logic Gates

- The building blocks used to create digital circuits are **logic gates**
- There are three elementary logic gates and a range of other simple gates
- Each gate has its own **logic symbol** which allows complex functions to be represented by a logic diagram
- The function of each gate can be represented by a **truth table** or using **Boolean notation**

- **The AND gate**



(a) Circuit symbol

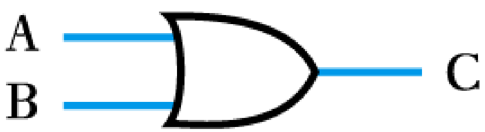
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

- **The OR gate**



(a) Circuit symbol

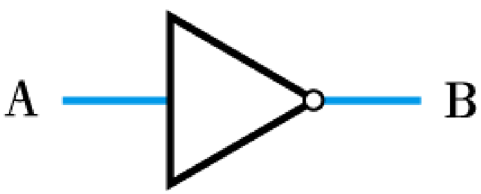
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$$C = A + B$$

(c) Boolean expression

- **The NOT gate (or inverter)**



(a) Circuit symbol

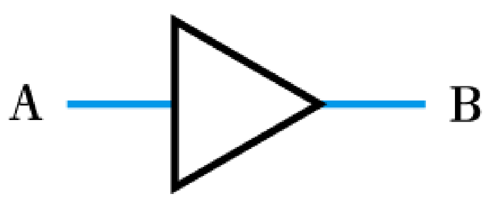
A	B
0	1
1	0

(b) Truth table

$$B = \overline{A}$$

(c) Boolean expression

- **A logic buffer gate**



(a) Circuit symbol

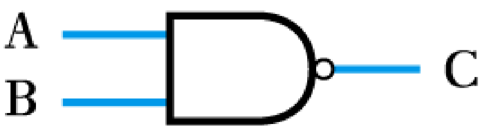
A	B
0	0
1	1

(b) Truth table

$$B = A$$

(c) Boolean expression

- **The NAND gate**



(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

- **The NOR gate**

(a) Circuit symbol

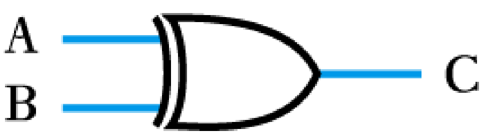
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

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

(c) Boolean expression

- **The Exclusive OR gate**



(a) Circuit symbol

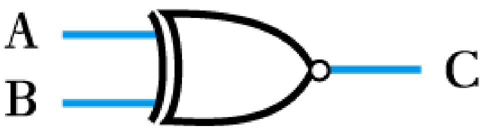
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

- **The Exclusive NOR gate**



(a) Circuit symbol

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

(b) Truth table

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

(c) Boolean expression

Boolean Algebra

- **Boolean Constants**
  - these are ‘0’ (false) and ‘1’ (true)
- **Boolean Variables**
  - variables that can only take the vales ‘0’ or ‘1’
- **Boolean Functions**
  - each of the logic functions (such as AND, OR and NOT) are represented by symbols as described above
- **Boolean Theorems**
  - a set of **identities** and **laws** – see text for details

- **Boolean identities**

AND Function	OR Function	NOT function
$0 \bullet 0 = 0$	$0 + 0 = 0$	$\overline{0} = 1$
$0 \bullet 1 = 0$	$0 + 1 = 1$	$\overline{1} = 0$
$1 \bullet 0 = 0$	$1 + 0 = 1$	$\overline{\overline{A}} = A$
$1 \bullet 1 = 1$	$1 + 1 = 1$	
$A \bullet 0 = 0$	$A + 0 = A$	
$0 \bullet A = 0$	$0 + A = A$	
$A \bullet 1 = A$	$A + 1 = 1$	
$1 \bullet A = A$	$1 + A = 1$	
$A \bullet A = A$	$A + A = A$	
$A \cdot \overline{A} = 0$	$A + \overline{A} = 1$	

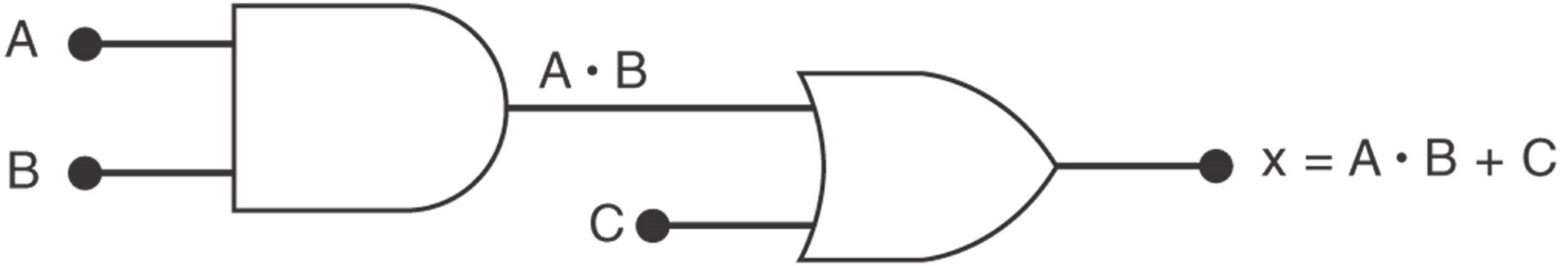
- **Boolean laws**

Commutative law $AB = BA$ $A + B = B + A$	**Absorption law** $A + AB = A$ $A(A + B) = A$
Distributive law $A(B + C) = AB + BC$ $A + BC = (A + B)(A + C)$	**De Morgan's law** $\overline{A + B} = \overline{A} \cdot \overline{B}$ $\overline{A \cdot B} = \overline{A} + \overline{B}$
Associative law $A(BC) = (AB)C$ $A + (B + C) = (A + B) + C$	**Note also** $A + \overline{A}B = A + B$ $A(\overline{A} + B) = AB$

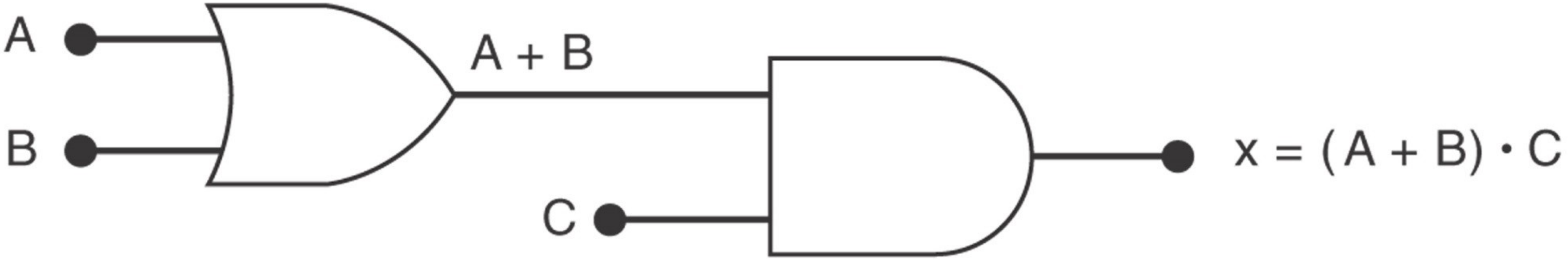
Combinational Logic

- Digital systems may be divided into two broad categories:
  - **combinational logic**
    - where the outputs are determined solely by the current states of the inputs
  - **sequential logic**
    - where the outputs are determined not only by the current inputs but also by the sequence of inputs that led to the current state
- In this lecture we will look at combination logic

Examples 1,2

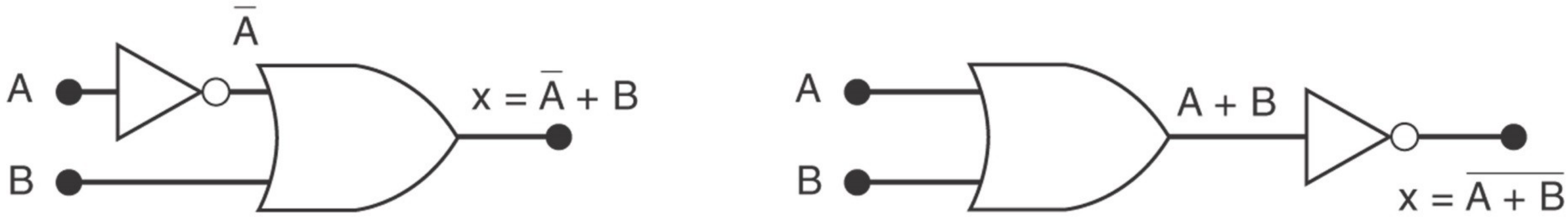


(a)



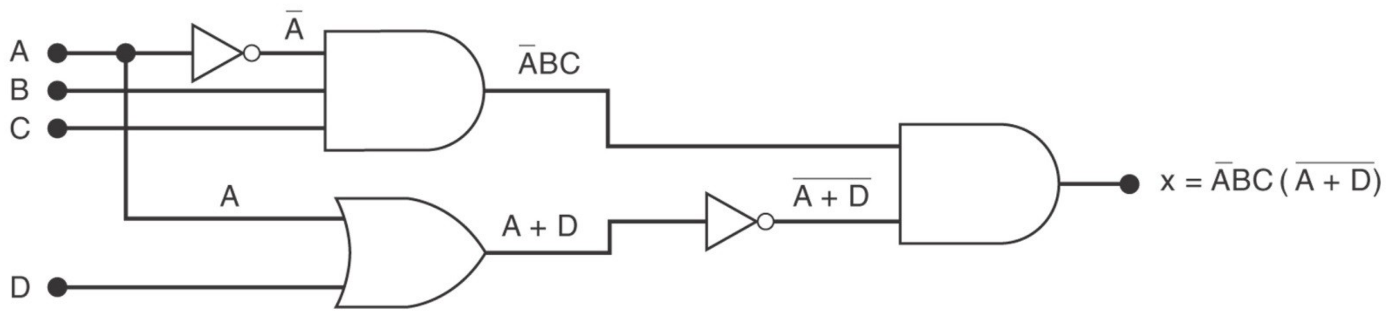
(b)

Examples 3

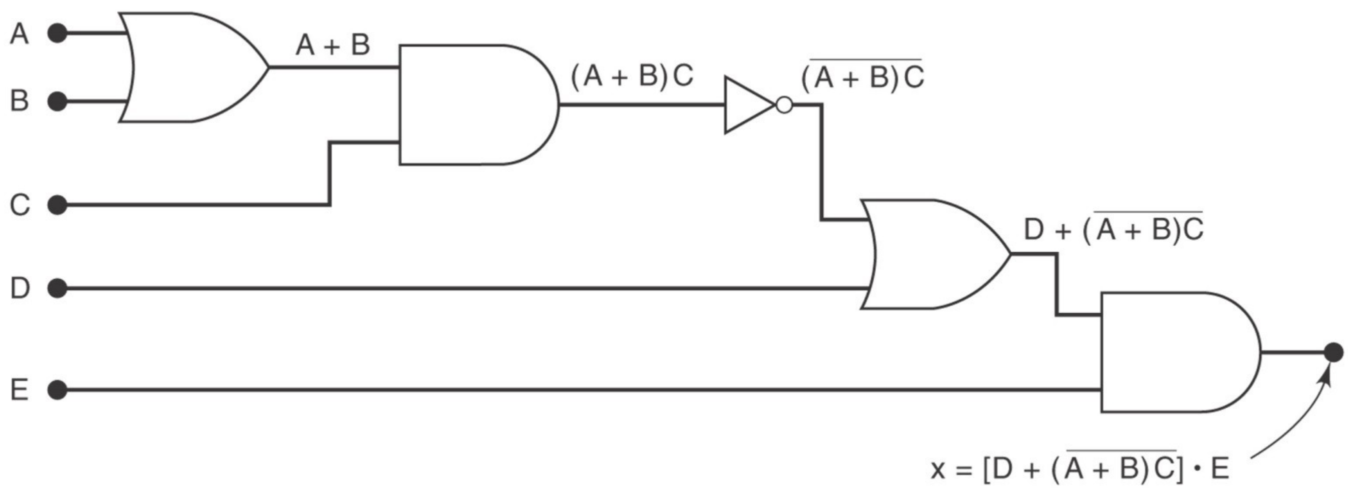


(a) (b)

Example 4


A
B
C
D
$\overline{A}$
$\overline{A}BC$
A
A + D
$\overline{A + D}$
$x = \overline{A}BC(\overline{A + D})$

(a)

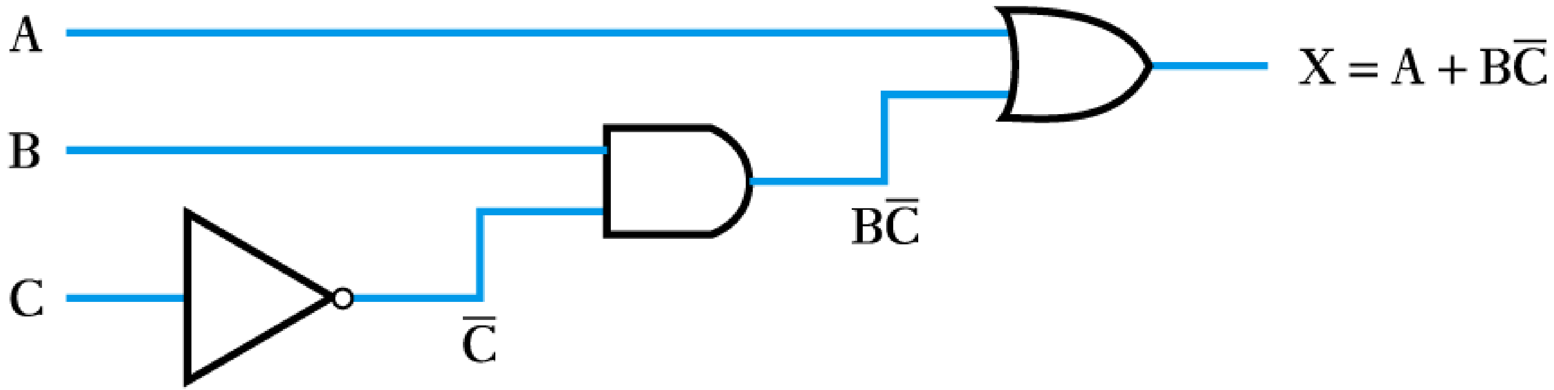

A
B
C
D
E
A + B
(A + B)C
$\overline{(A + B)C}$
$D + (\overline{A + B)C}$
$x = [D + (\overline{A + B)C}] \cdot E$

(b)

- **Implementing a function from a Boolean expression**

Example –

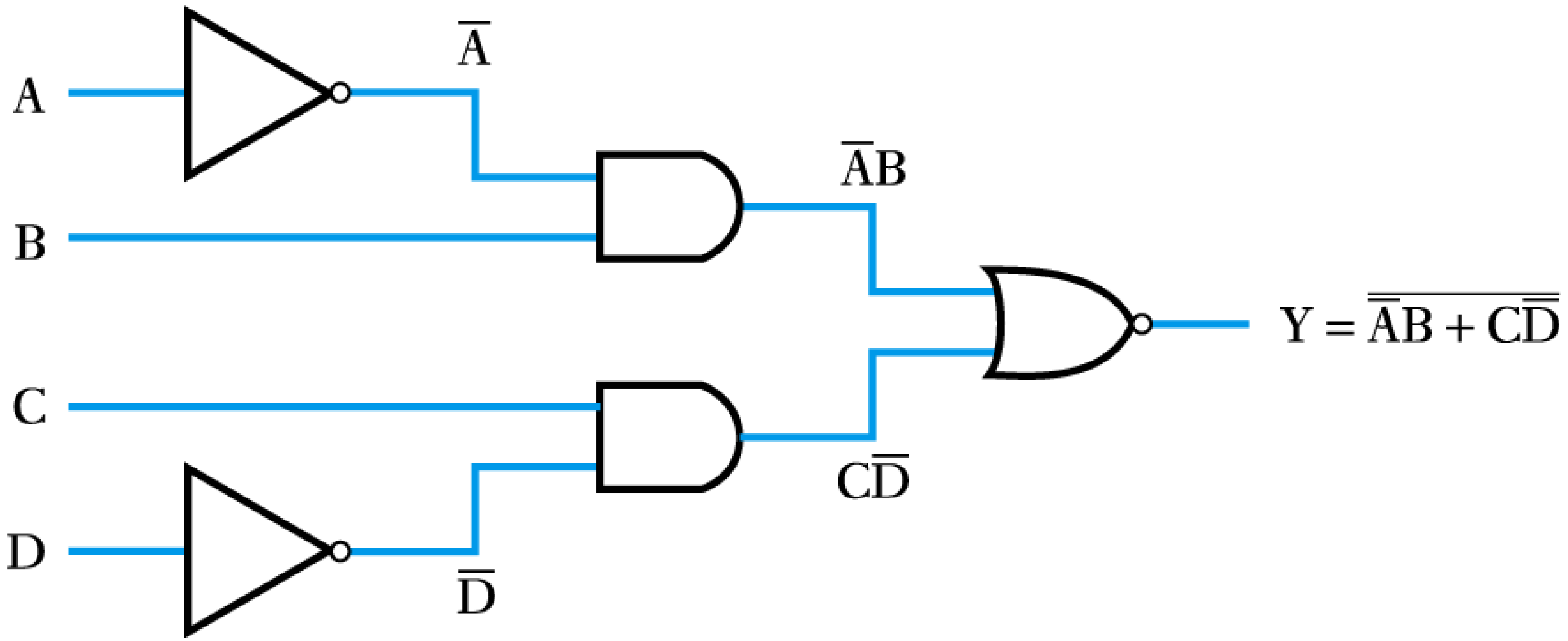
$$X = A + B\overline{C}$$



- **Implementing a function from a Boolean expression**

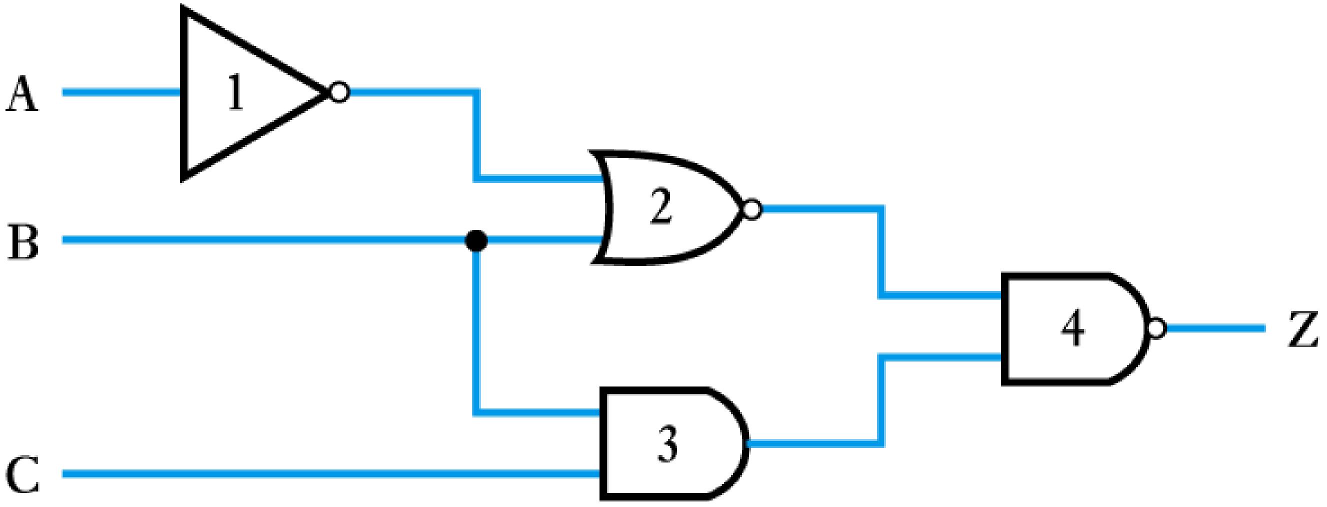
Example –

$$Y = \overline{\overline{A}\,B + C\,\overline{D}}$$



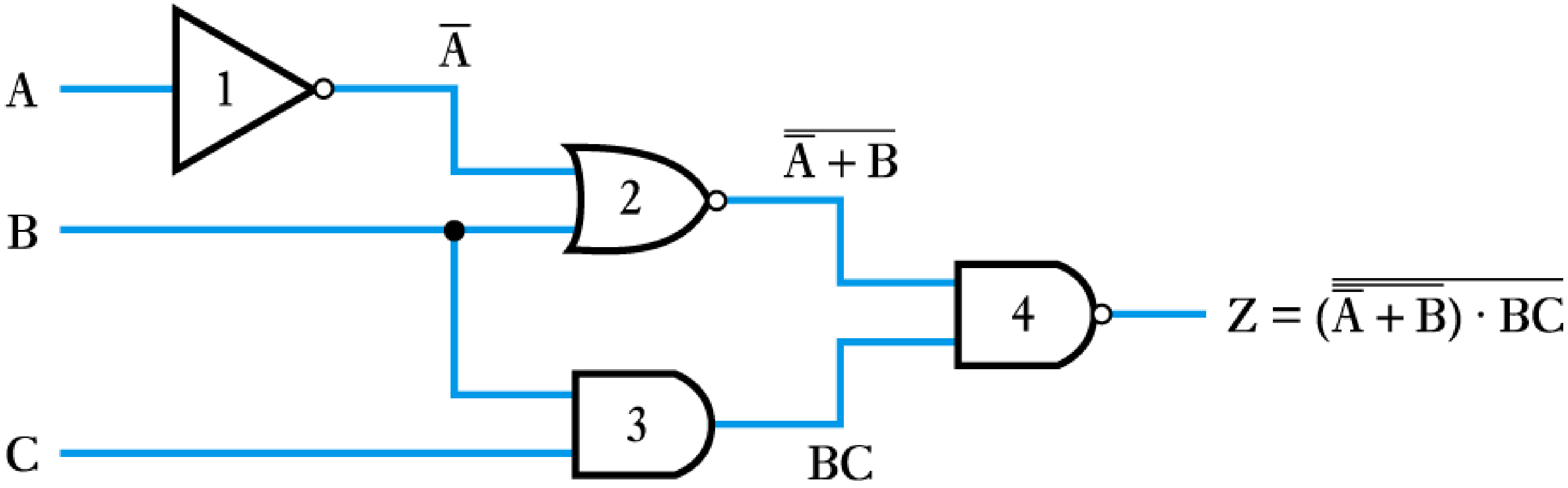
- **Generating a Boolean expression from a logic diagram**

Example –



Example (continued)

– work progressively from the inputs to the output adding logic expressions to the output of each gate in turn



- **Implementing a logic function from a description**

Example –

The operation of the Exclusive OR gate can be stated as:

"The output should be true if either of its inputs are true, but not if both inputs are true."

This can be rephrased as:

"*The output is true if A* OR *B is true*, AND *if A* AND *B are* NOT *true*."

We can write this in Boolean notation as

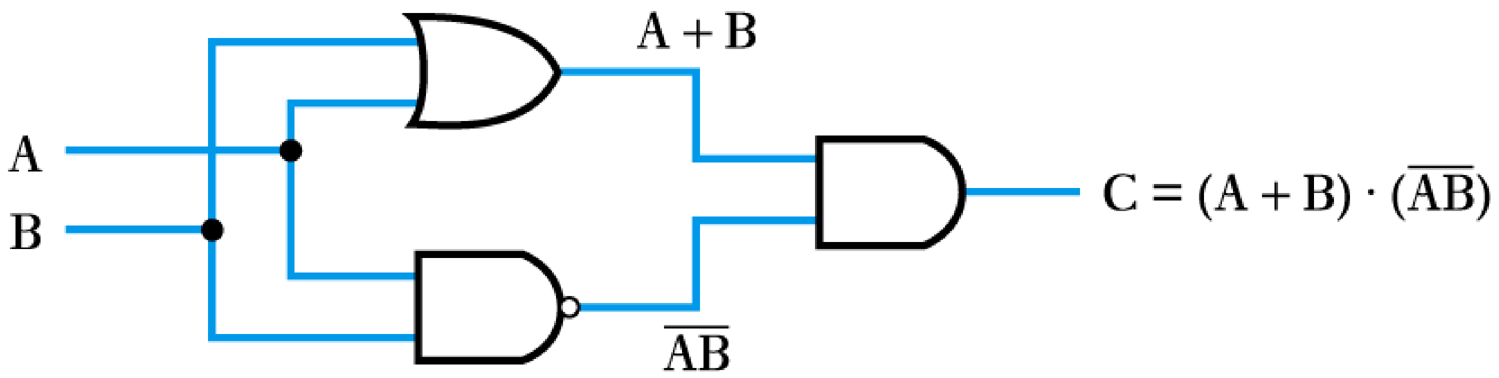
$$X = (A + B) \cdot \overline{(AB)}$$

Example (continued)

The logic function

$$X = (A + B) \cdot \overline{(AB)}$$

can then be implemented as before



- **Implementing a logic function from a truth table**

Example –

Implement the function of the following truth table

A	B	C	X
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	0

– first write down a Boolean expression for the output

– then implement as before

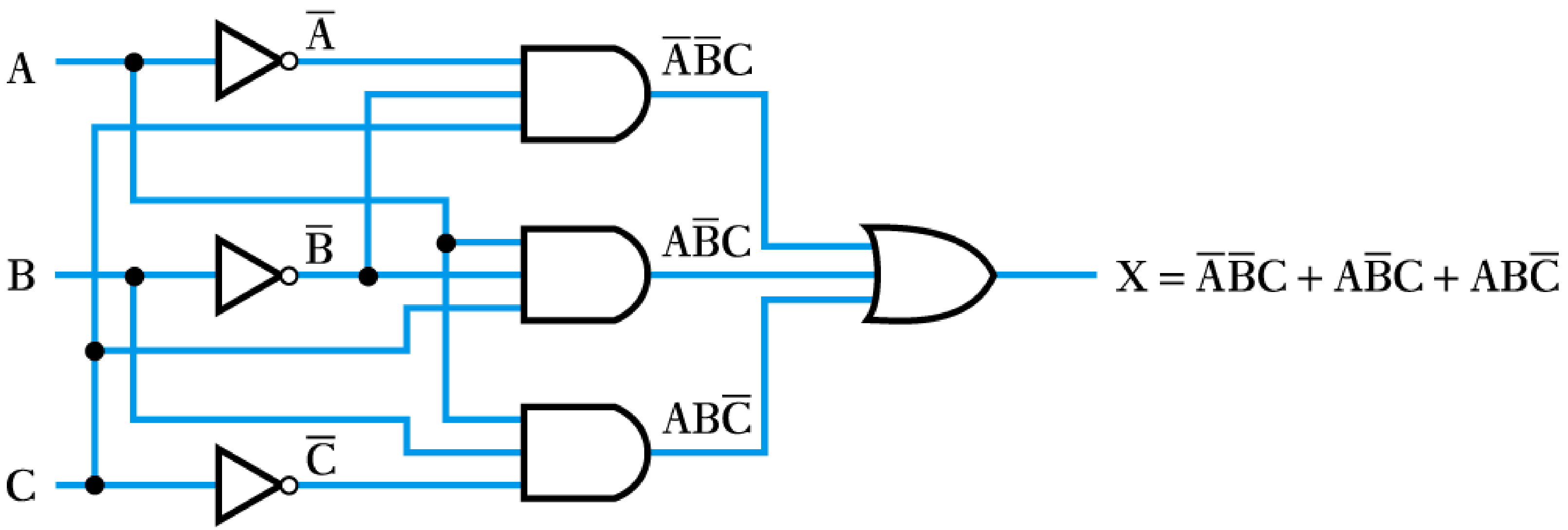
– in this case

$$X = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,C + A\,B\,\overline{C}$$

Example (continued)

The logic function $X = \overline{A}\,\overline{B}\,C + A\,\overline{B}\,C + A\,B\,\overline{C}$

can then be implemented as before



- In some cases it is possible to *simplify* logic expressions using the rules of Boolean algebra

Example –

$X = ABC + \overline{A}\,BC + AC + A\,\overline{C}$ can be simplified to $X = BC + A$

hence the following circuits are equivalent