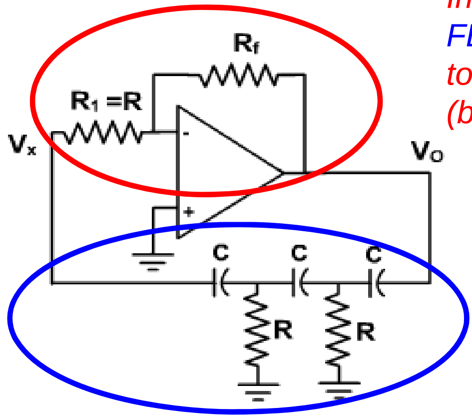
Last time: Key points oscillators

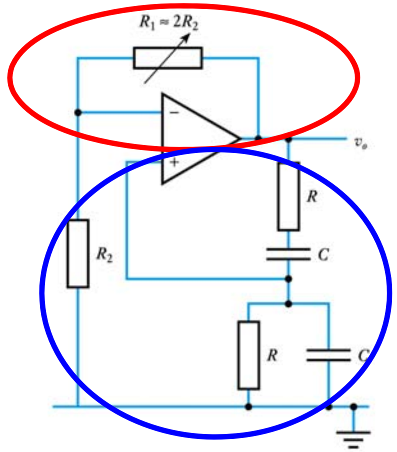
- Positive feedback is used in analogue and digital systems
- A primary use is in the production of oscillators
- The requirement for oscillation is that the loop gain *AB* must have a magnitude of 1, and a phase shift of 180° (or 180° plus some integer multiple of 360°)
- This can be achieved using a circuit that produces a phase shift of 180° subtracted from a non-inverting amplifier's feedback (or added to an inverting amplifier's feedback)
- Alternatively, it can be achieved using a circuit that produces a phase shift of 0° subtracted from an inverting amplifier's feedback (or added to a non-inverting amplifier's feedback)
- For good frequency stability we often use crystals
- Care must be taken to ensure the stability of all feedback systems

Sine-wave oscillator systems

- Two feedback loops
- FB$_1$ purely resistive gain
  - $V_{fb1} = G_1 \cdot V_o$, where G_1=constant
- FB$_2$ has $G_2(f)$ with phase(f)
  - V_{fb2} Cancels V_{fb1}, but only when V_o changes at $f=f_o$
- Near saturation, V_o slows or stops ($A \rightarrow 0 \Rightarrow f \rightarrow 0 \Rightarrow G_2 \rightarrow 0$)
- FB$_2$ no longer cancels FB$_1$ and V_o reverses



Inverting & FB$_2$ added to FB$_1$ (both at V_-)



Non-inverting & FB$_2$ subtracted from FB$_1$ (at V_+ and V_-)

How do we create a sine wave?

(*from TI- Application Report SLOA060 - March 2001*)

As the phase shift approaches 180° and $|A\beta| \rightarrow 1$, the output voltage of the now-unstable system tends to infinity but, of course, is limited to finite values by an energy-limited power supply.

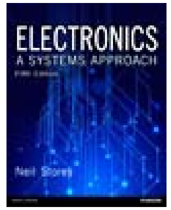
When the output voltage approaches either power rail, the active devices in the amplifiers change gain. This causes the value of A to change and forces $A\beta$ away from the singularity; thus the trajectory towards an infinite voltage slows and eventually halts.

At this stage, one of three things can occur:

(i) Nonlinearity in saturation or cutoff causes the system to become stable and lock up at the current power rail.

(ii) The initial change causes the system to saturate (or cutoff) and stay that way for a long time before it becomes linear and heads for the opposite power rail. This produces highly distorted oscillations (usually quasi-square waves), the resulting oscillators being called relaxation oscillators

(iii) The system stays linear and reverses direction, heading for the opposite power rail. **This produces a sine-wave oscillator**.

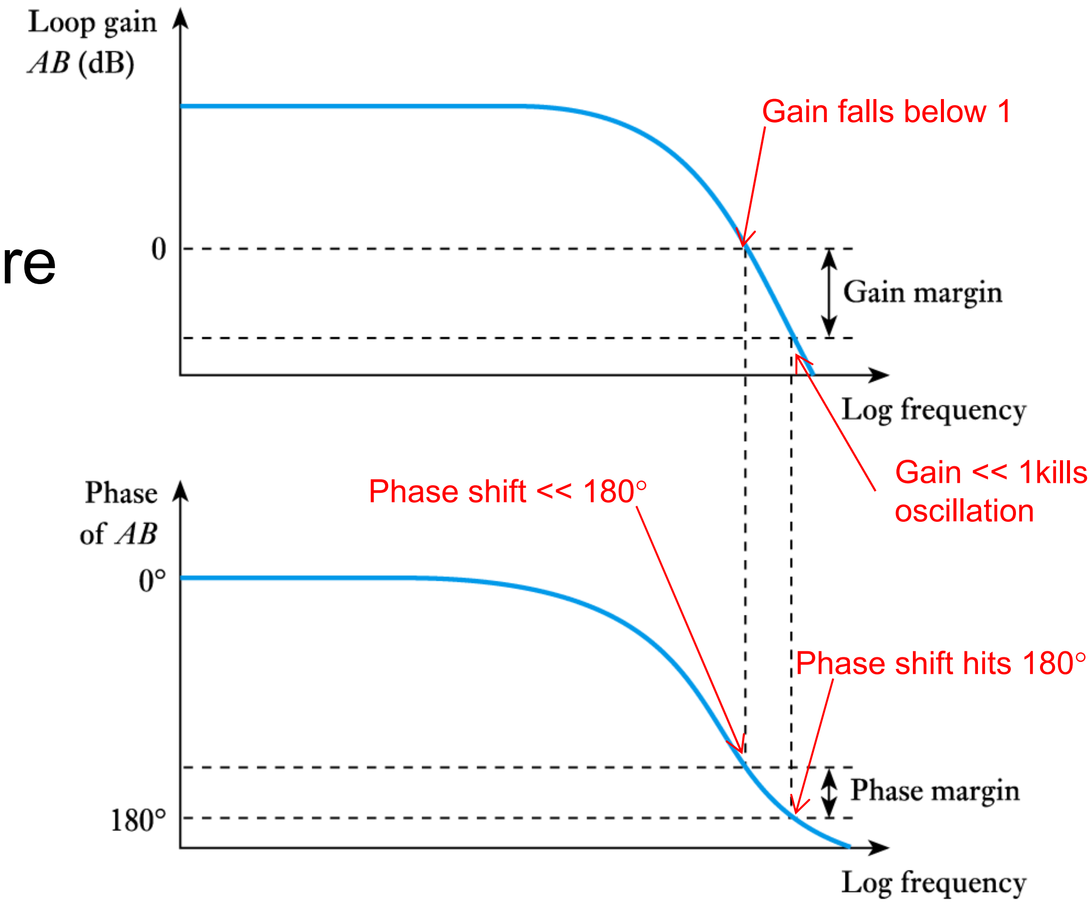


23.3

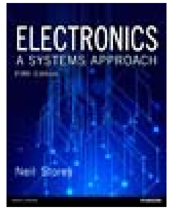
Stability

- The gain of all real amplifiers falls at high frequencies and this produces a phase shift of the feedback
- Unintended feedback within a circuit or **stray capacitance** or **stray inductance** can produce feedback phase shifts
- All multi-stage amplifiers will produce 180° of phase shift at *some* frequency
- To ensure stability we must ensure that the Baukhausen conditions for oscillation are *not* met
  - That is, |AB|=1 *and* 180° phase shift of B
  - to guarantee this we must ensure that the gain falls below unity before the phase shift reaches 180°

- **Gain and phase margins**
  - these are a measure of the stability of a circuit



- QUESTIONS?



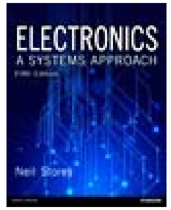
Chapter 24

Digital systems

- Introduction
- Binary quantities and variables
- Logic gates
- Boolean algebra
- Combinational logic
- Boolean algebraic manipulation
- Algebraic simplification
- Karnaugh maps
- Automated methods of minimisation
- Propagation delay and hazards
- Number systems and binary arithmetic
- Numeric and alphabetic codes
- Examples of combinational logic design

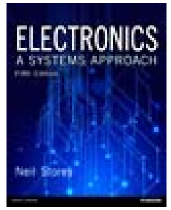
Introduction

Video 24A



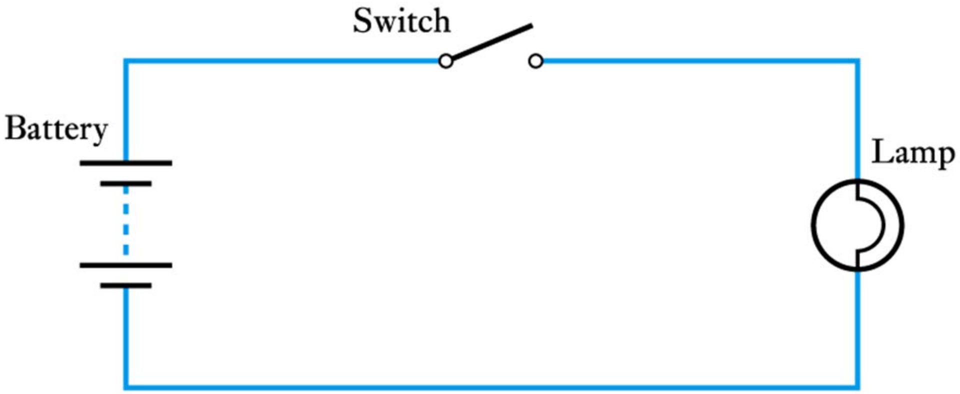
24.1

- 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



Binary quantities and variables

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



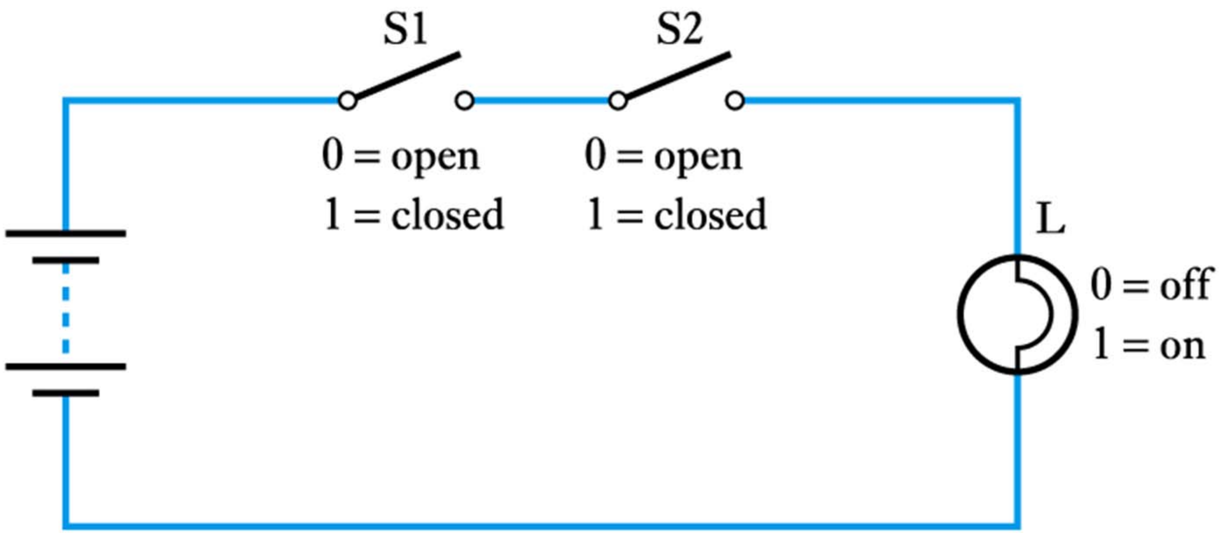
A simple binary arrangement

S	*L*
OPEN	OFF
CLOSED	ON

S	*L*
0	0
1	1

A truth table

- A binary arrangement with two switches in series



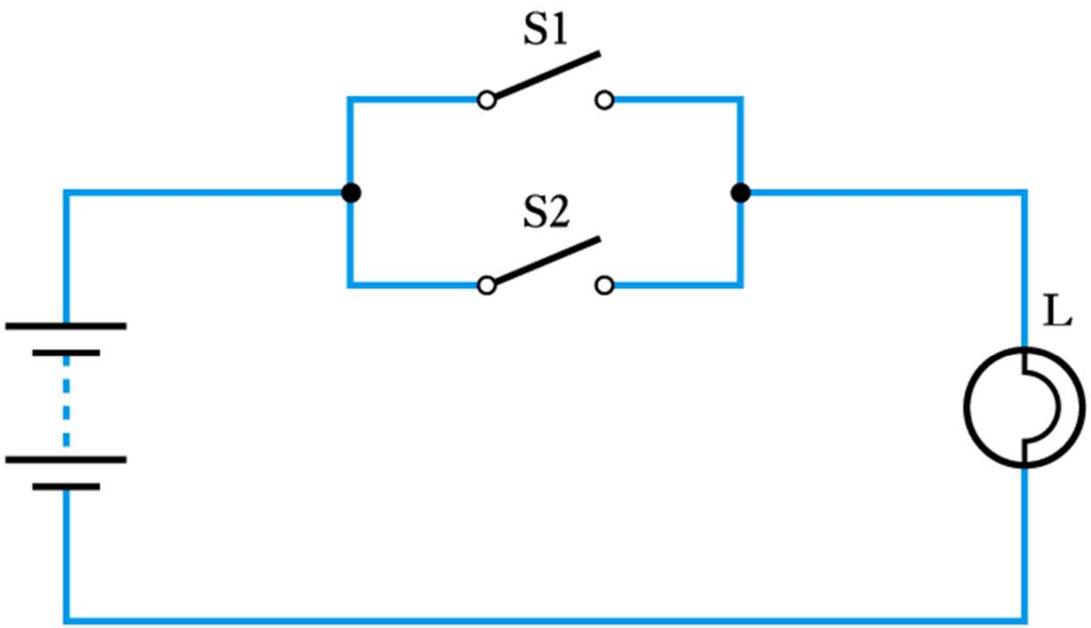
(a) Circuit

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



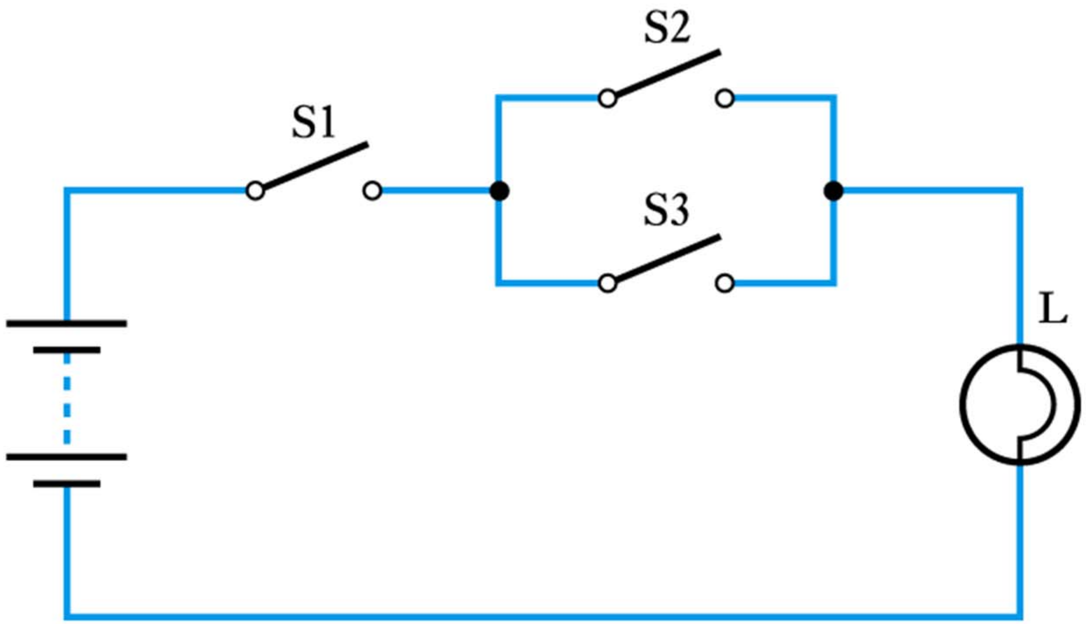
(a) Circuit

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

(b) Truth table

L = $S1$ OR $S2$

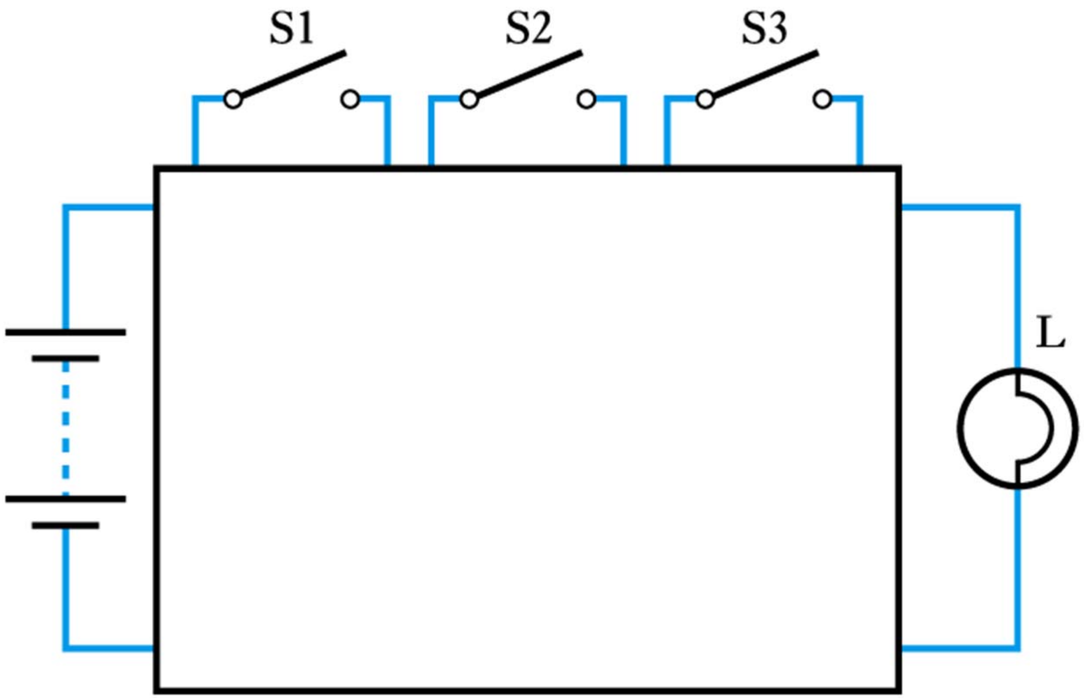
- A series/parallel arrangement

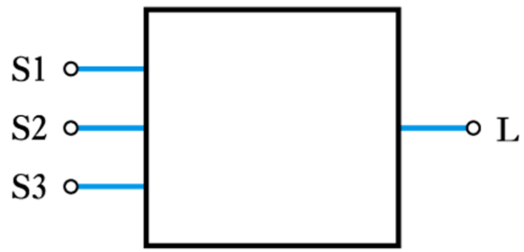


S1	S2	S3	L
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

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

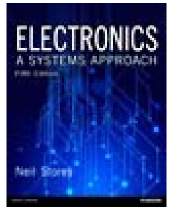
- Representing an unknown network with a truth table





Use this to look at Logic gates

Video 24B

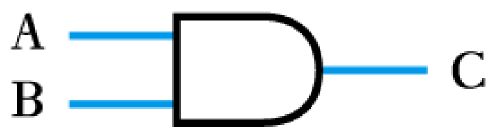


24.3

- 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**

Count up the inputs in binary to ensure all possibilities covered



(a) Circuit symbol

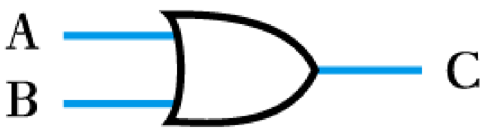
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

(b) Truth table

$$C = A \cdot B$$

(c) Boolean expression

- ## The OR gate



(a) Circuit symbol

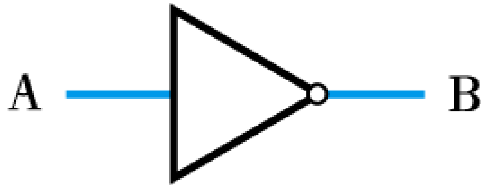
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

(b) Truth table

$$C = A + B$$

(c) Boolean expression

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



(a) Circuit symbol

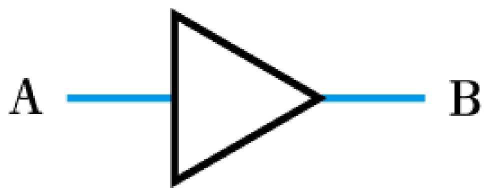
A	B
0	1
1	0

(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

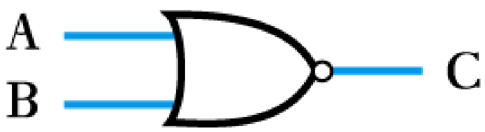
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

(b) Truth table

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

(c) Boolean expression

▪ The NOR gate



(a) Circuit symbol

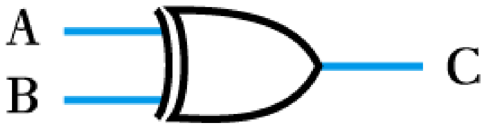
A	B	C
0	0	1
0	1	0
1	0	0
1	1	0

(b) Truth table

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

(c) Boolean expression

The Exclusive OR gate



(a) Circuit symbol

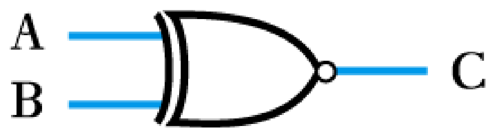
A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

(b) Truth table

$$C = A \oplus B$$

(c) Boolean expression

The Exclusive NOR gate



(a) Circuit symbol

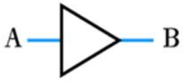
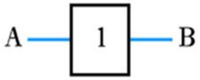
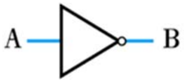
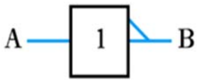
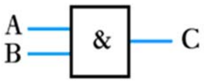
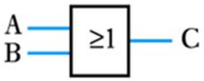
A	B	C
0	0	1
0	1	0
1	0	0
1	1	1

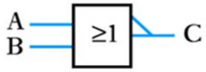
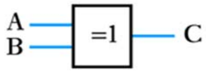
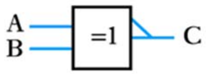
(b) Truth table

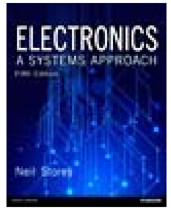
$$C = \overline{A \oplus B}$$

(c) Boolean expression

SUMMARY

Function	Symbol	Alternative symbol	Boolean expression	Truth table
Buffer			$B = A$	A | B 0 | 0 1 | 1
NOT			$B = \overline{A}$	A | B 0 | 1 1 | 0
AND			$C = A \cdot B$	A B | C 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR			$C = A + B$	A B | C 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1

Function	Symbol	Alternative symbol	Boolean expression	Truth table
NAND			$C = \overline{A \cdot B}$	A B | C 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR			$C = \overline{A + B}$	A B | C 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive OR			$C = A \oplus B$	A B | C 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive NOR			$C = \overline{A \oplus B}$	A B | C 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1



24.4

Boolean algebra

- **Boolean constants**
  - these are ‘0’ (false) and ‘1’ (true)
- **Boolean variables**
  - variables that can only take the values ‘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 Table 24.2 for details

Boolean theorems

Boolean identities

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

Boolean laws

Commutative law
$AB = BA$
$A + B = B + A$

Distributive law
$A(B + C) = AB + BC$
$A + BC = (A + B)(A + C)$

Associative law
$A(BC) = (AB)C$
$A + (B + C) = (A + B) + C$

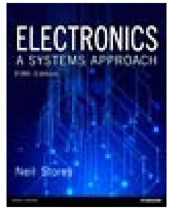
Absorption law
$A + AB = A$
$A(A + B) = A$

De Morgan's law
$\overline{A + B} = \overline{A} \cdot \overline{B}$
$\overline{A \cdot B} = \overline{A} + \overline{B}$

Note also
$A + \overline{A}\,B = A + B$
$A(\overline{A} + B) = AB$

Combinational logic

Video 24C



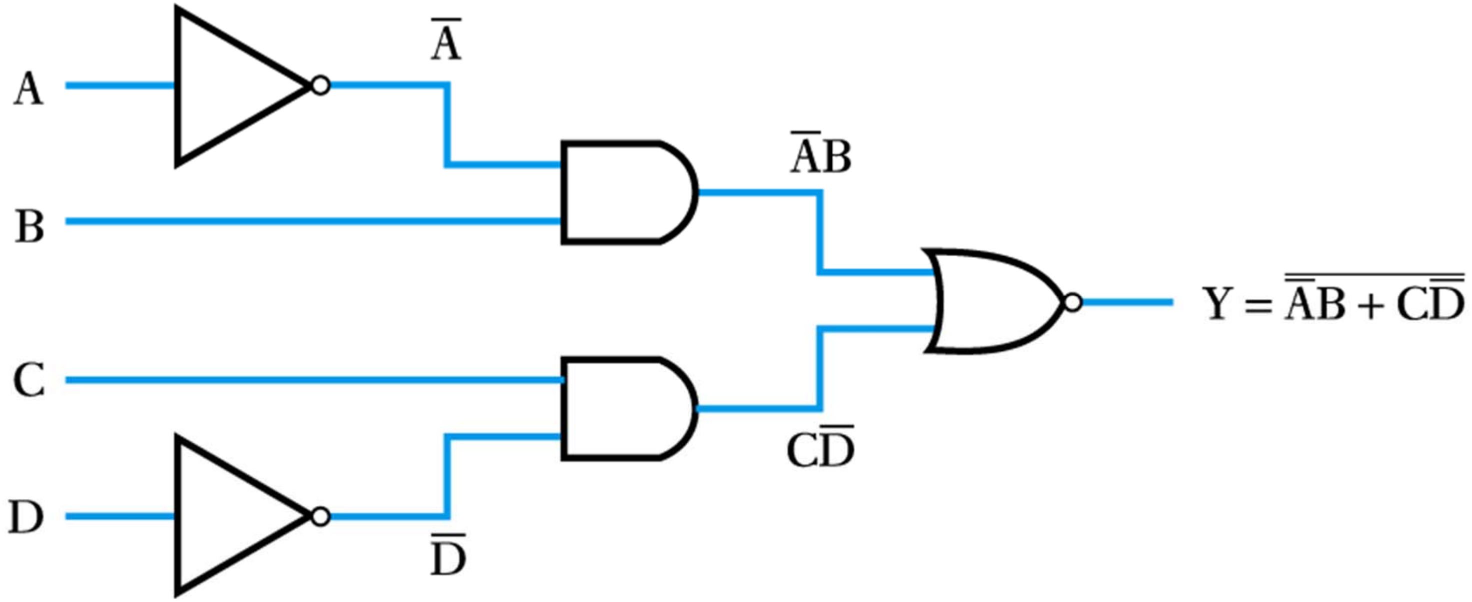
24.5

- 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***

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

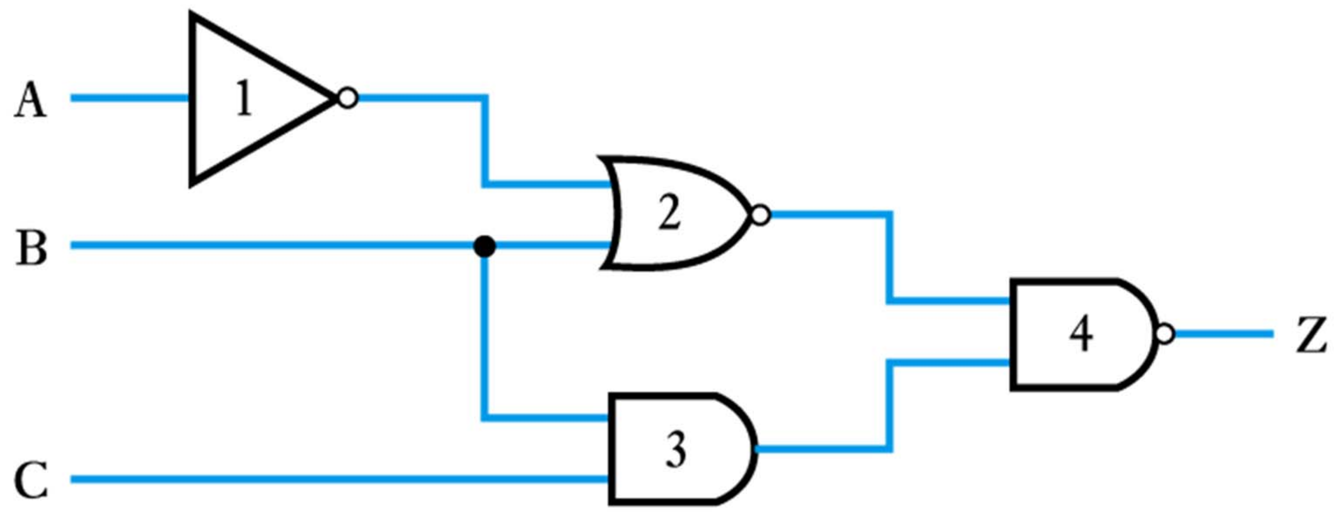
Example – see **Example 24.2** in the course text

Implement the function $Y = \overline{\overline{A}B + C\overline{D}}$



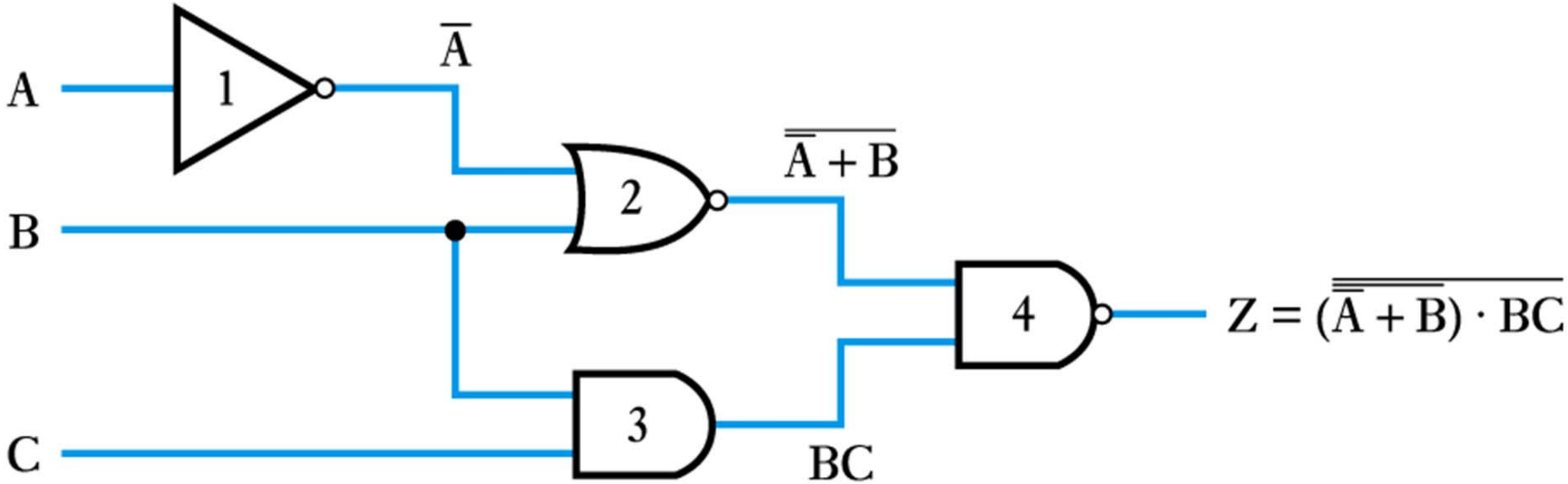
- **Generating a Boolean expression from a logic diagram**

Example – see **Example 24.3** in the course text



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 – see **Example 24.4** in the course text

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

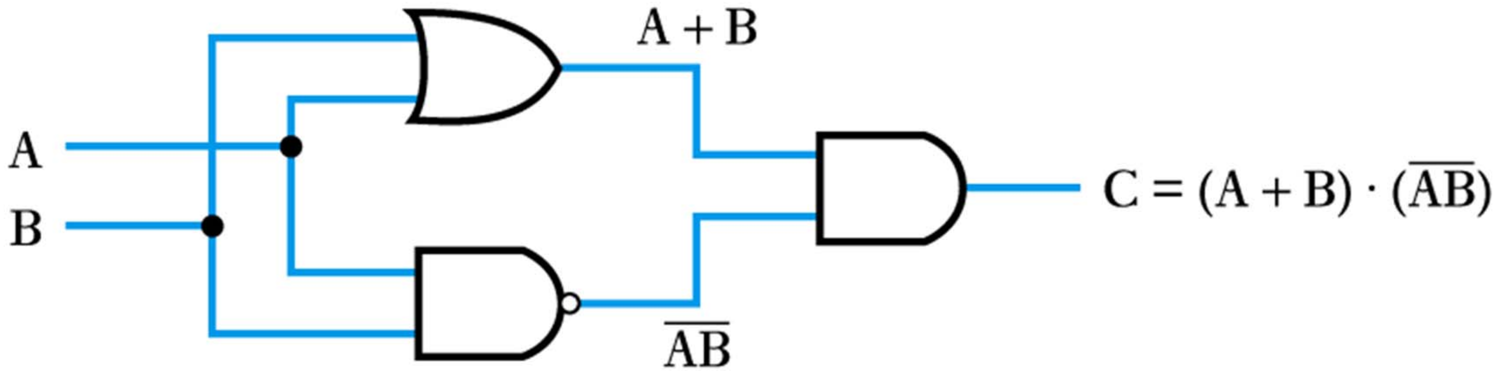
$$X = (A + B) \bullet \overline{(AB)}$$

Example (continued)

The logic function

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

can then be implemented as before



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

Example – see **Example 24.6** in the course text

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 as the sum of the **minterms**

– then implement as before

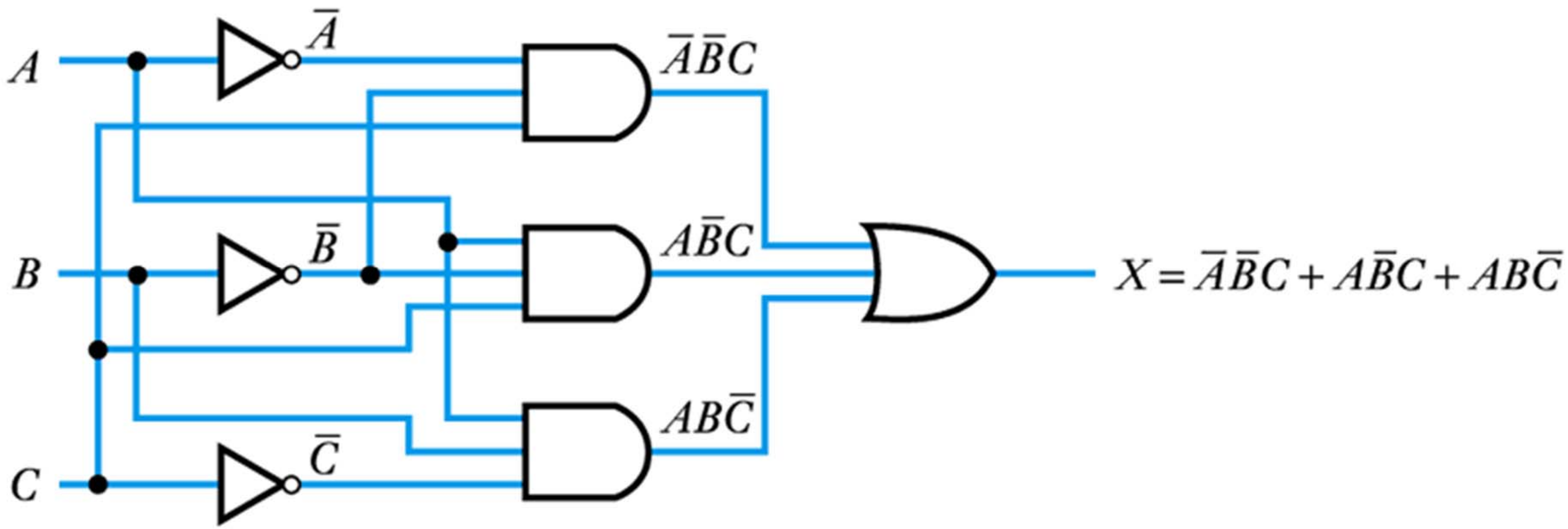
– in this case

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

Example (continued)

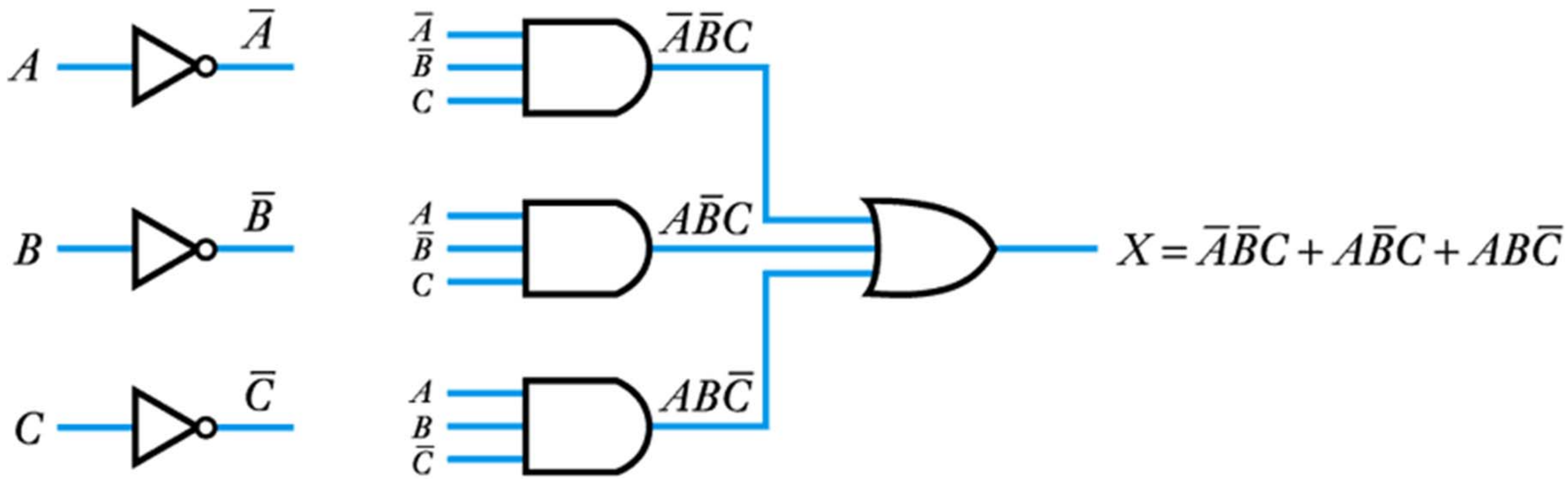
The logic function $X = \overline{A}\,\overline{B}C + A\overline{B}C + AB\overline{C}$

can then be implemented as before



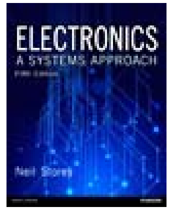
Example (continued)

– Complex logic diagrams are often made easier to understand by the use of labels, rather than showing complex interconnections – the earlier circuit becomes



Algebraic simplification

Video 24D



24.7

- All combination circuits can be described in a **sum-of-products** form
  - This consists of a number of terms (**minterms** or **products**) that are OR'ed together
  - Examples include

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

$$XYZ + \overline{X}Y\overline{Z} + X\overline{Y}Z$$

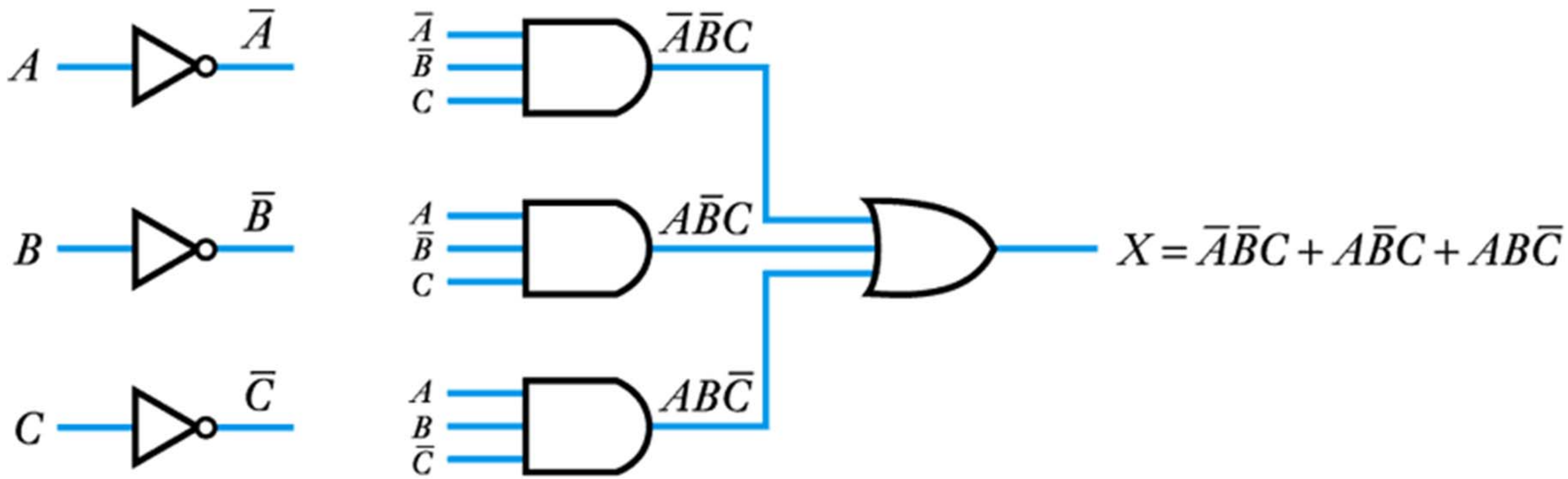
$$AB\overline{C}D + A\overline{B}C + \overline{A}BCD + ABC\overline{D}$$

  - The sum-of-products must *not* include inversions of a series of terms, as in

$$A\overline{BC}D + \overline{AB}CD$$

Example (continued)

– Complex logic diagrams are often made easier to understand by the use of labels, rather than showing complex interconnections – the earlier circuit becomes

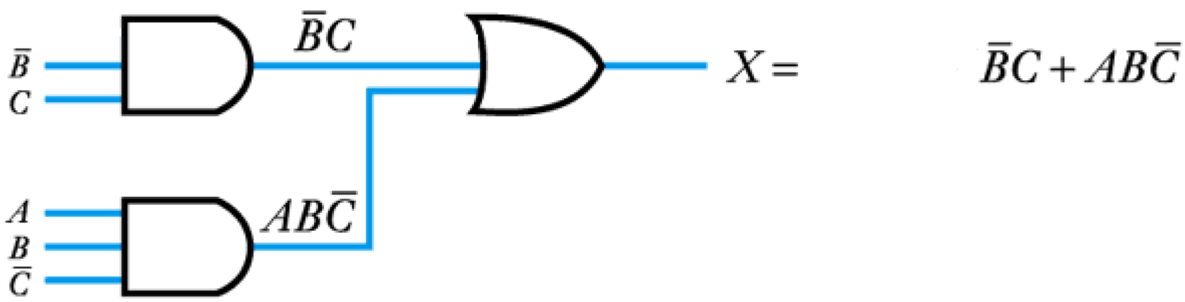


Allows us to simplify the circuit

- We can use Boolean laws to simplify expression and the circuit

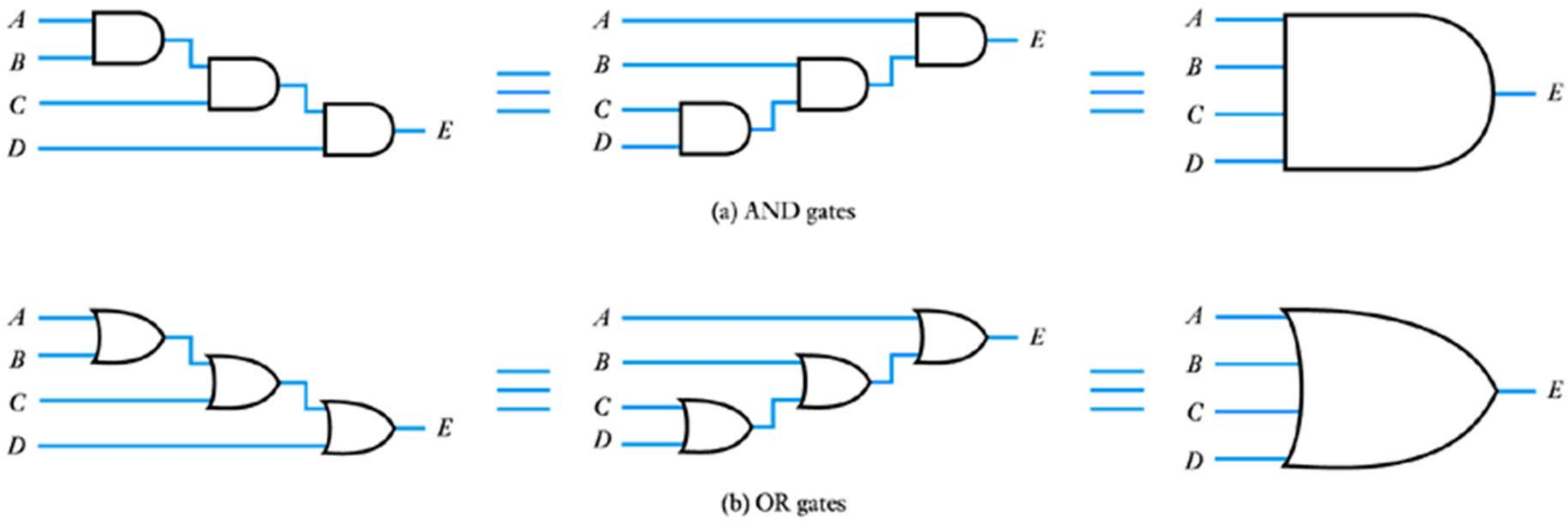
$$
\begin{aligned}
X &= \bar{A}\bar{B}C + A\bar{B}C + AB\bar{C} \\
&= \bar{B}C\,(A + \bar{A}) + AB\bar{C} \\
&= \bar{B}C + AB\bar{C}
\end{aligned}
$$

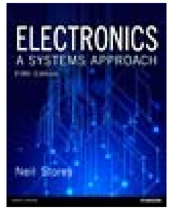
- So a circuit with identical logical properties is:



Boolean algebraic manipulation

- We can use the various laws of Boolean algebra to change the form of circuits to meet space available
  - the following diagram shows the implications of the associative laws





Chapter 24

Next time: Digital systems

- Introduction
- Binary quantities and variables
- Logic gates
- Boolean algebra
- Combinational logic
- Boolean algebraic manipulation
- Algebraic simplification
- Karnaugh maps
- Automated methods of minimisation
- Propagation delay and hazards
- Number systems and binary arithmetic
- Numeric and alphabetic codes
- Examples of combinational logic design

Key points

- Logic circuits are usually implemented using logic gates
- Circuits in which the output is determined solely by the current inputs are termed combinational logic circuits
- Logic functions can be described by truth tables or using Boolean algebraic notation
- Boolean expressions can often be simplified by algebraic manipulation, or using techniques such as Karnaugh maps
- Binary digits may be combined to form digital words that can be processed using binary arithmetic
- Several codes can be used to represent different forms of information