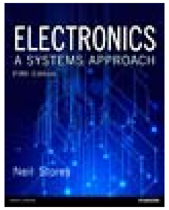
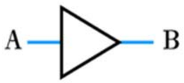
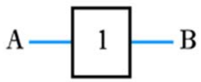
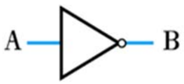
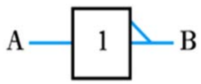
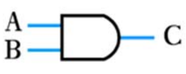
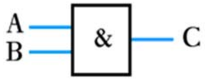
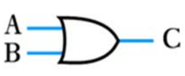
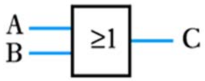
- Reference group report is on It’s Learning
- The reference group will be available at the end of lecture today to take any additional comments

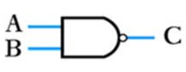
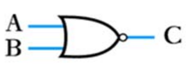
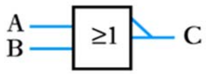
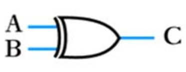
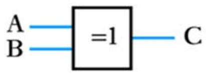
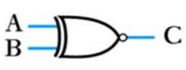
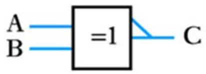


Last time: Digital systems

- 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

Last time: Summary of combinational logic

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

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

Note: Boolean expression are not unique

- Can express an exclusive-OR as:

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

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

- Can also express an exclusive-OR as:

  “*The output is true if A is true* AND *B is* NOT *true*,
  OR *if A is* NOT *true* AND *B is true*.”

$$X = A\bar{B} + \bar{A}B$$

- Both give:

$$X = A \oplus B$$

▪ 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**

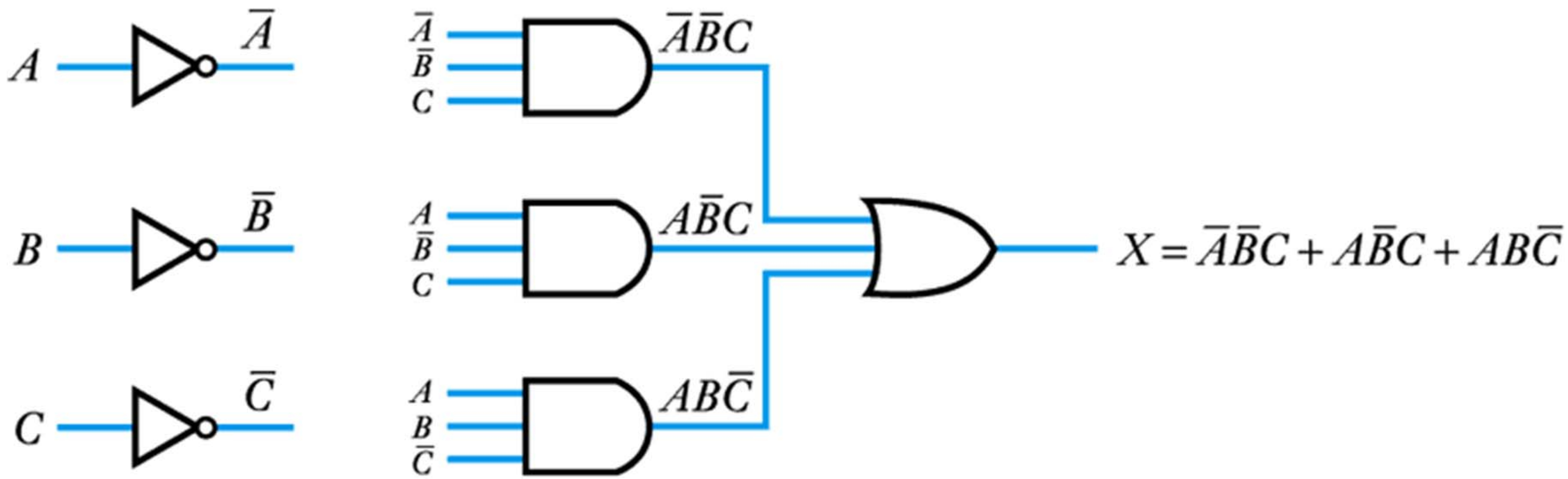
– then implement as before

– in this case

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

Example (continued)

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

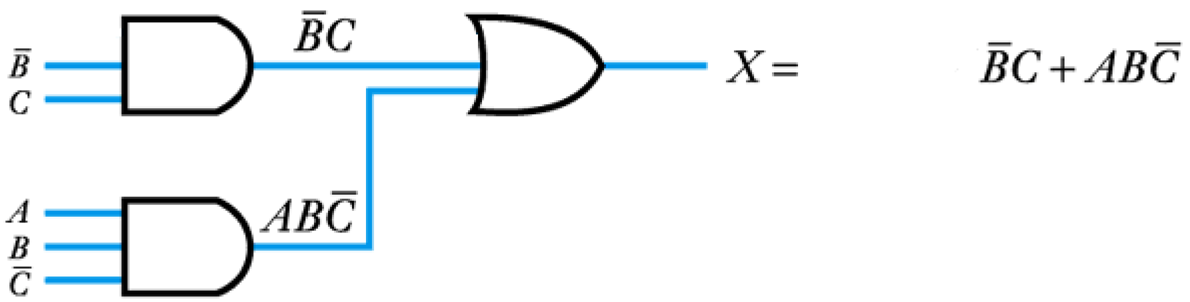


Simplify the Boolean equation, simplify the circuit

- 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:



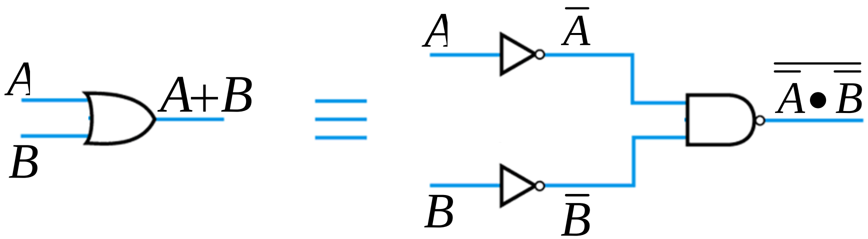
- Questions?

Can use Boolean relationships to re-format so that only 1 type of gate (NAND) is used

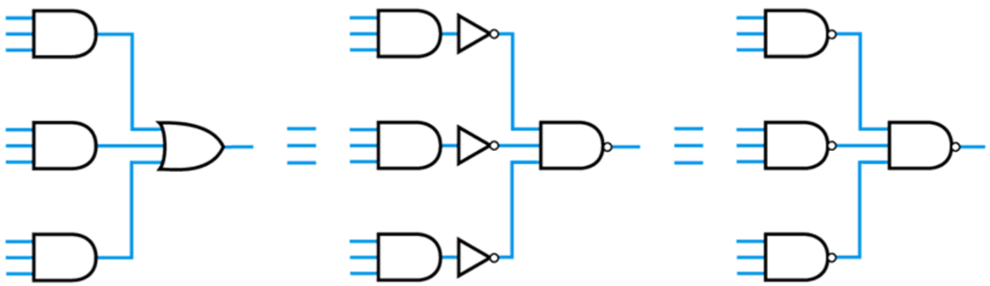
- Using De Morgan's 1st Law: $\overline{A+B}=\overline{A}\bullet\overline{B}$ *can be written*

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

- Hence:



- So any multi-term



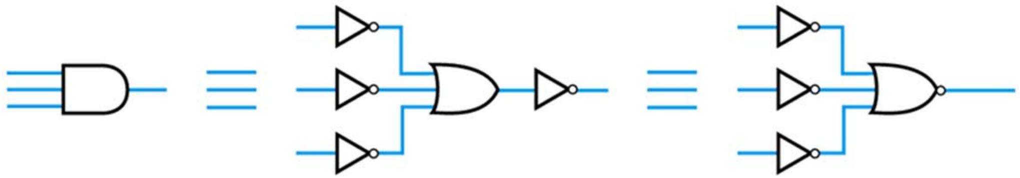
Why?
Many logic arrays use only one type of gate, or <u>multiple NANDs or NORs on one chip</u>
For logic arrays user generally does not need to know as this is auto-implemented!

Similarly with NOR gates

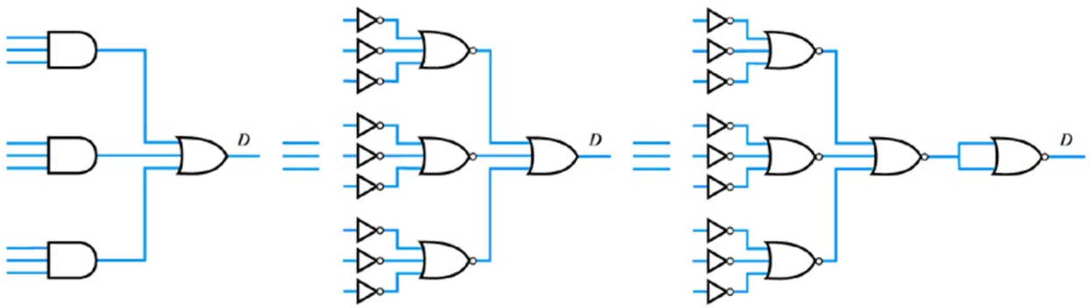
- Using De Morgan's second: $\overline{A \bullet B} = \overline{A} + \overline{B}$ *can be written*

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

- Hence:



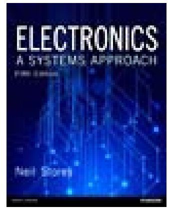
- So:



How do we know what to simplify?

Karnaugh maps

Video 24E

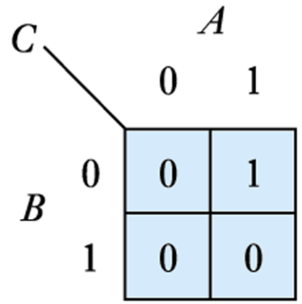


24.8

- With algebraic simplification it is not obvious whether an optimum form has been obtained
- Karnaugh maps are a graphical approach
- They represent the information of a truth table within a two dimensional grid

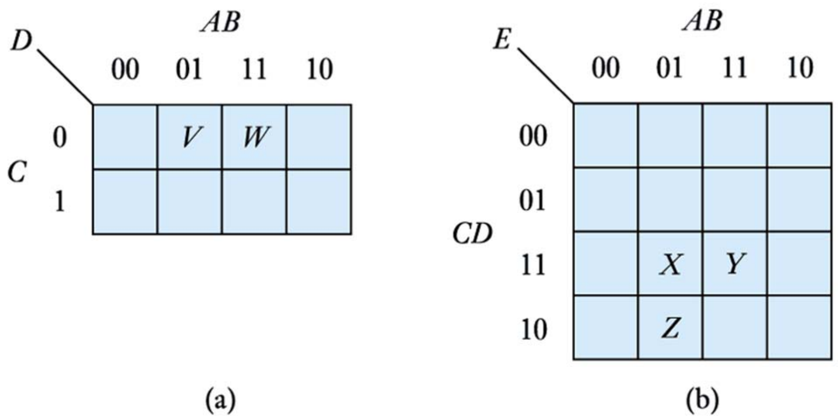
A	*B*	*C*
0	0	0
0	1	0
1	0	1
1	1	0

(a) Truth table



(b) Karnaugh map

- The diagram here shows maps for systems with three and four inputs
- Note the numbering of the grid – **Gray Code**



- Any adjacent cells (vertically or horizontally) differ by only one term
- For example

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

$$W = AB\overline{C}$$

- **Forming Gray code**
  - first write down the first two numbers (0 and 1)
  - then repeat in reverse order with a 1 in front (add 0’s to top half)
  - then keep repeating in reverse order with a 1 in front

0		00
1	➡	01
		11
		10

- **Forming Gray code**
  - first write down the first two numbers (0 and 1)
  - then repeat in reverse order with a 1 in front (add 0's to top half)
  - then keep repeating in reverse order with a 1 in front

0	00	000
1	01	001
	11	011
	10	010
		110
		111
		101
		100

Forming Gray code

- first write down the first two numbers (0 and 1)
- then repeat in reverse order with a 1 in front (add 0’s to top half)
- then keep repeating in reverse order with a 1 in front

0	00	000	
1	01	001	
	11	011	
	10	010	
		110	
		111	etc.
		101	
		100	

Consider the two truth tables

Can they be simplified?

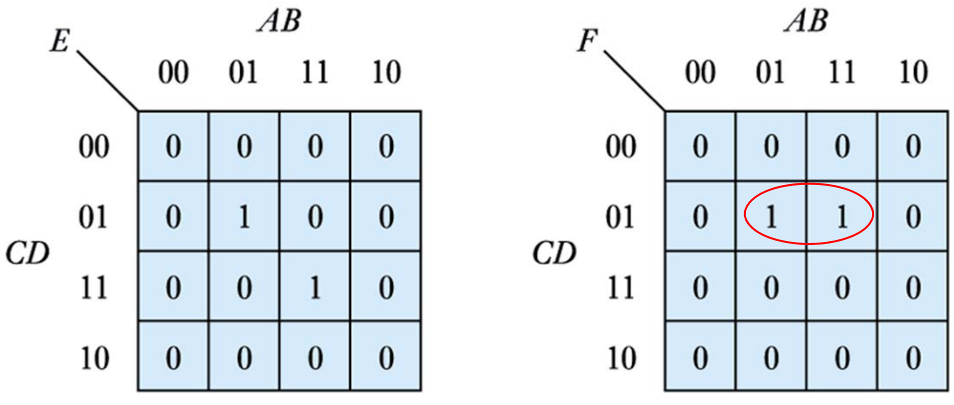
ABCD	E
0000	0
0001	0
0010	0
0011	0
0100	0
0101	1
0110	0
0111	0
1000	0
1001	0
1010	0
1011	0
1100	0
1101	0
1110	0
1111	1

$$E = \overline{A}B\overline{C}D + ABCD$$

ABCD	F
0000	0
0001	0
0010	0
0011	0
0100	0
0101	1
0110	0
0111	0
1000	0
1001	0
1010	0
1011	0
1100	0
1101	1
1110	0
1111	0

$$F = \overline{A}B\overline{C}D + AB\overline{C}D$$

- Consider these maps
- We can extract the algebraic expressions, as in a truth table



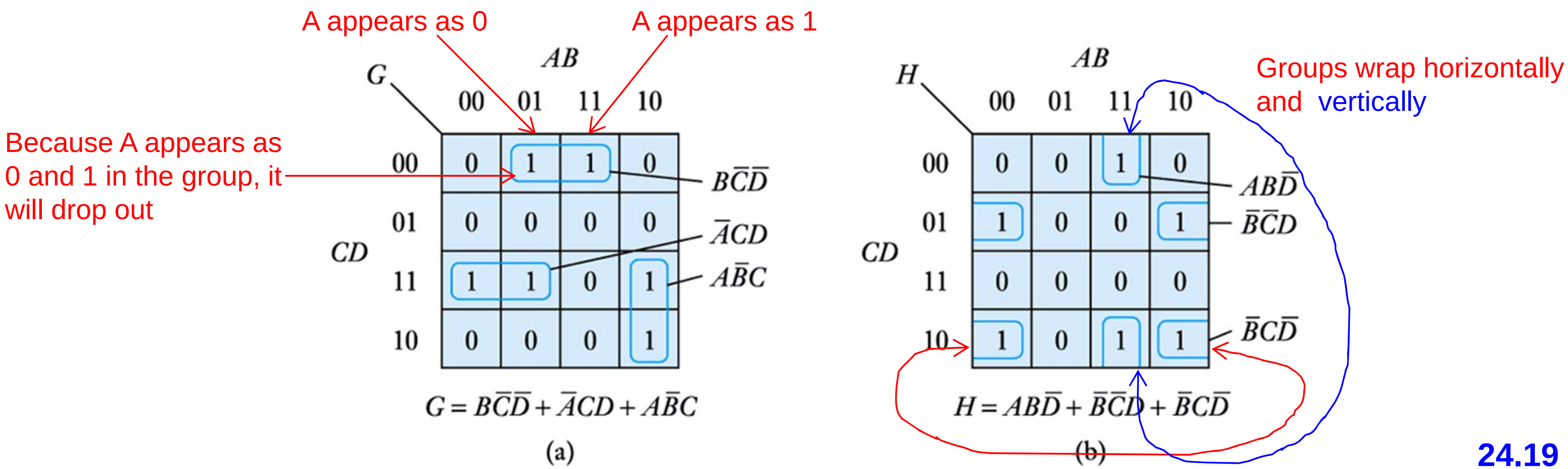
(a) (b)

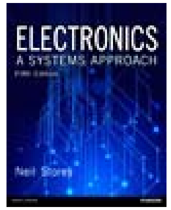
$$E = \overline{A}B\overline{C}D + ABCD$$

$$\begin{aligned} F &= \overline{A}B\overline{C}D + AB\overline{C}D \\ &= B\overline{C}D(A + \overline{A}) \\ &= B\overline{C}D \end{aligned}$$

- When adjacent cells contain ‘1’s they can *always* be combined, thus simplifying the expression

- We group terms on the map by drawing a loop around them (size of 2^n, where n=number of input variables).
- The expression for this group is made up of the terms that are consistent for the cells within
- The expression for the function is the sum of the groups



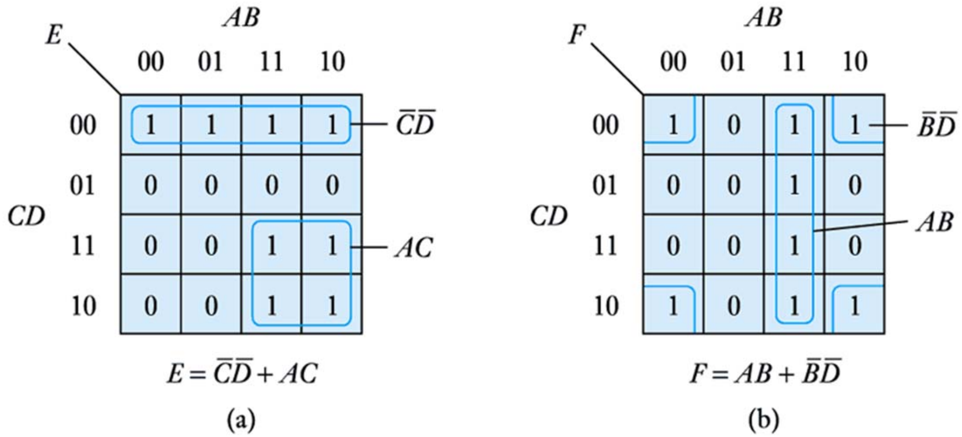


24.9

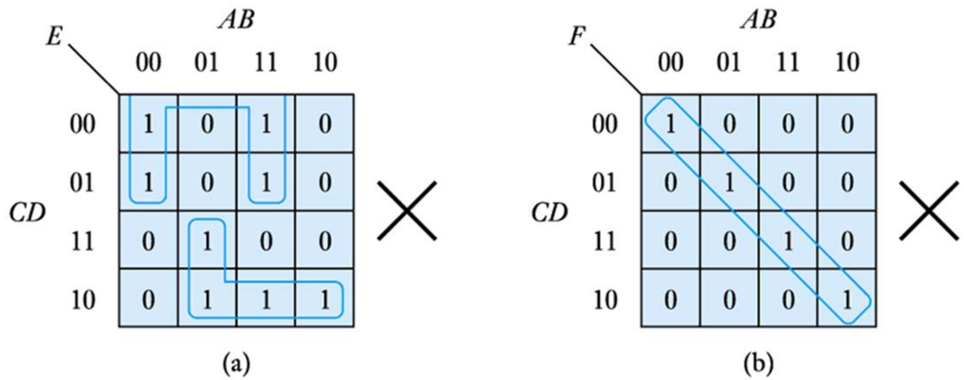
Method of minimisation

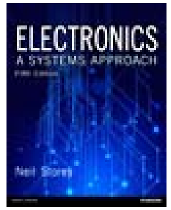
- The largest possible groups of cells should be constructed first, each group containing 2^n elements (n = number of input variables)
- Progressively smaller groups should be added until every cell containing a ‘1’ has been included at least once.
- Any redundant groups should then be removed, even if these are large groups, to avoid duplication.
- Karnaugh maps can be used fairly easily with up to six variables. Beyond that computer algorithms are generally used

- Allowable groups



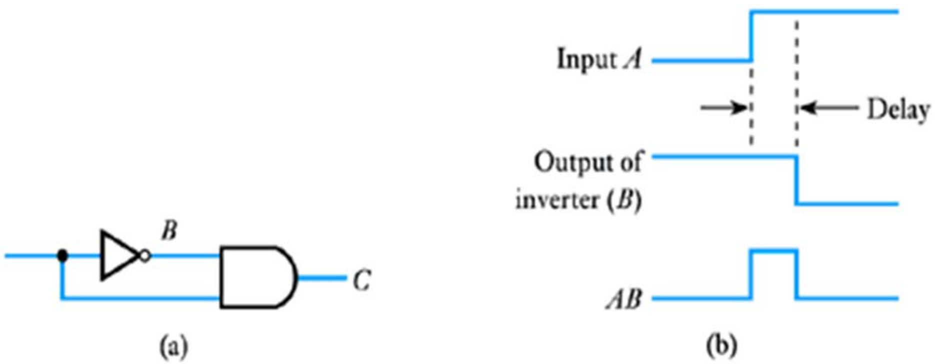
- Illegal groups



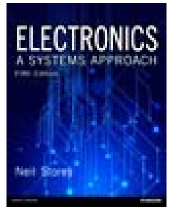


Propagation delay and hazards

- All physical logic gates take a finite time to respond to input signals and there is a delay between the time the input changes and when the output responds
- This is the **propagation time delay**
- This delay can cause problems. Consider this circuit



- The delay causes the output to differ from what we might expect
- This is an example of a **hazard** (can be identified and eliminated using a Karnaugh map)



24.11

Number systems and binary arithmetic

- Most number systems are order dependent
- **Decimal**

$$1234_{10} = (1 \times 10^{3}) + (2 \times 10^{2}) + (3 \times 10^{1}) + (4 \times 10^{0})$$

- **Binary**

$$1101_{2} = (1 \times 2^{3}) + (1 \times 2^{2}) + (0 \times 2^{1}) + (1 \times 2^{0}) \qquad = 13_{10}$$

- **Octal**

$$123_{8} = (1 \times 8^{2}) + (2 \times 8^{1}) + (3 \times 8^{0}) \qquad = 83_{10}$$

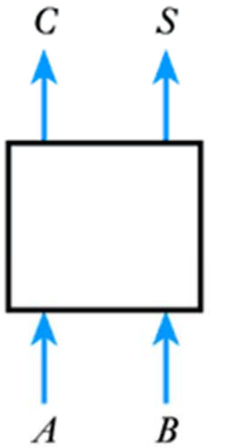
- **Hexadecimal**

$$123_{16} = (1 \times 16^{2}) + (2 \times 16^{1}) + (3 \times 16^{0}) \qquad = 291_{10}$$

here we need 16 characters – 0,1,2,3,4,5,6,7,8,9,A,B,C,D,E,F

Video 24F

- **Binary arithmetic**
  - much simpler than decimal arithmetic
  - can be performed by simple circuits, e.g. half adder



(a) Block diagram

Binary Addition

$0 + 0 = 0$

$0 + 1 = 1$

$1 + 0 = 1$

$1 + 1 = 10$

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

(b) Truth table

C:

B \ A	0	1
0	0	0
1	0	1

S:

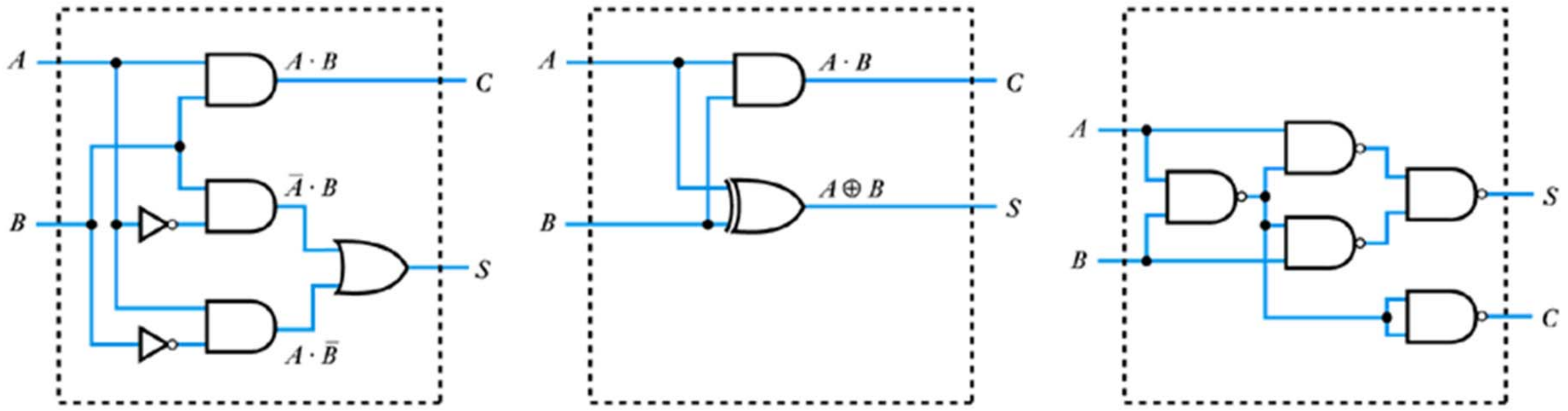
B \ A	0	1
0	0	1
1	1	0

(c) Karnaugh maps

$$C = A \cdot B$$

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

Implementation of a half adder



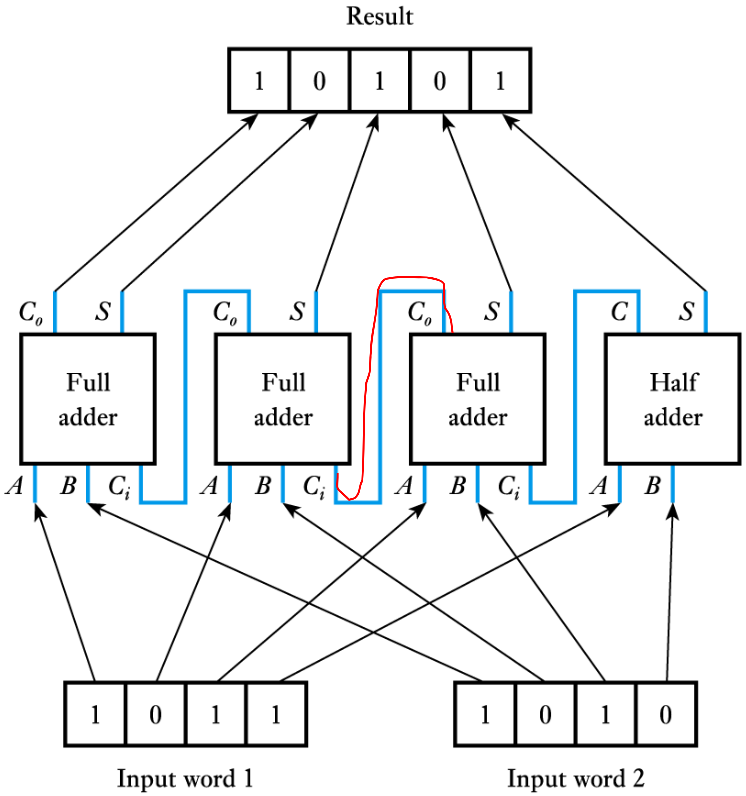
(a) (b) (c)

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

$$C = A \cdot B$$
$$S = A \oplus B$$

Or if you only have NAND gates

Adding multiple-bit words



- A half adder can add two single bits
- To add multiple-bit words we also need a component that can also cope with a carry from the previous stage
- This is a **full adder**

- **A full adder**

(a)

A	B	C_i	C_o	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

(b)

C_o (C_i \ AB)	00	01	11	10
0	0	0	1	0
1	0	1	1	1

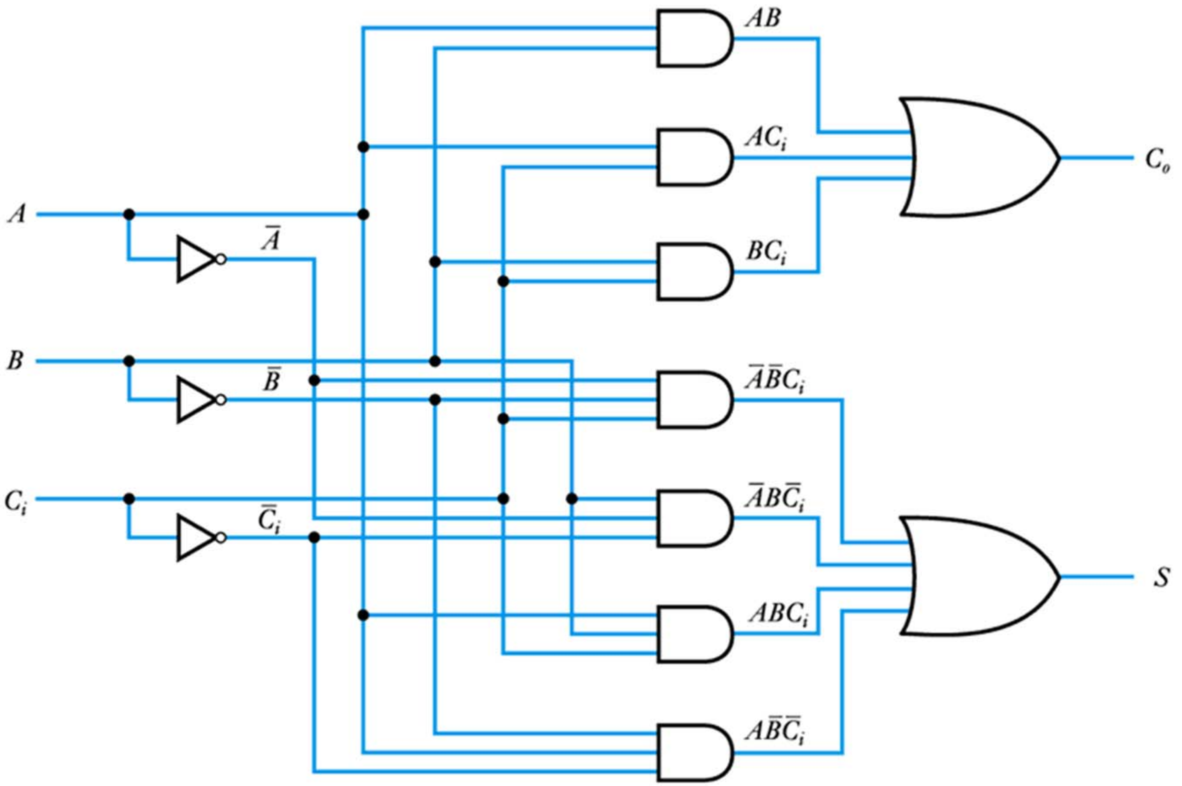
S (C_i \ AB)	00	01	11	10
0	0	1	0	1
1	1	0	1	0

$$C_o = \bar{A}BC_i + A\bar{B}C_i + ABC_i + AB\bar{C}_i$$

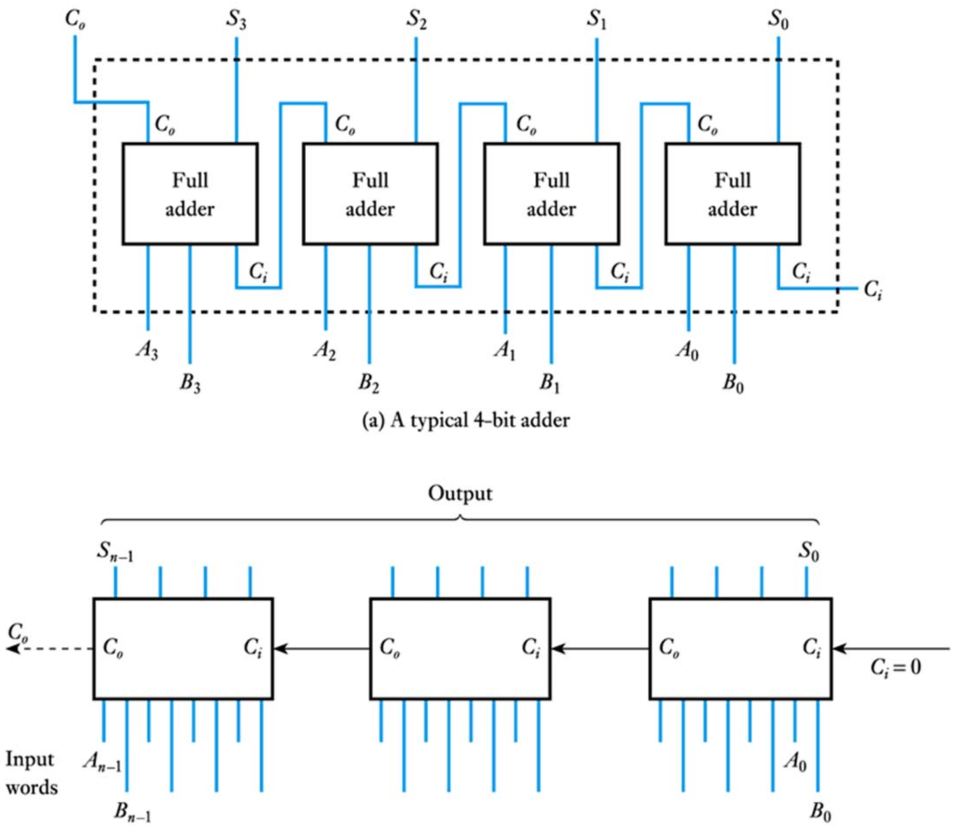
$$C_o = AB + AC_i + BC_i$$

$$S = \bar{A}\bar{B}C_i + \bar{A}B\bar{C}_i + ABC_i + A\bar{B}\bar{C}_i$$

- **Implementation of a full adder**



- **A cascadable 4-bit adder**



(a) A typical 4-bit adder

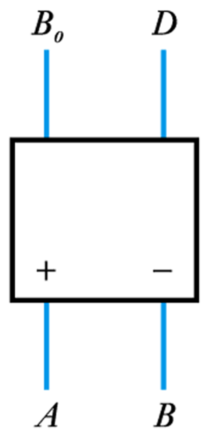
(b) Cascaded circuits

- **Binary subtraction**
  - A similar approach can be applied to subtraction
  - A half subtractor has two outputs called:

    D difference

    B_o borrow output

    The inputs are also labelled + and – to show which is subtracted from which

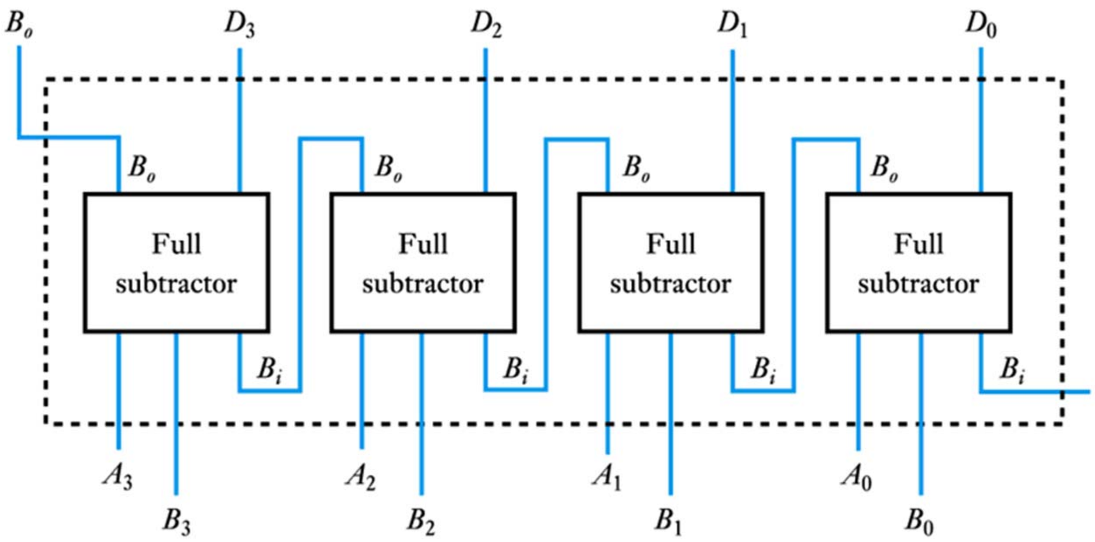


(a) Block diagram

A	*B*	B_o	*D*
0	0	0	0
0	1	1	1
1	0	0	1
1	1	0	0

(b) Truth table

- **A 4-bit subtractor**

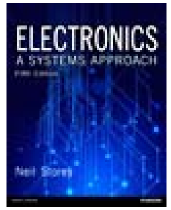


– These can be cascaded as with the adder circuits

- **Binary multiplication and division**
  - Although it is possible to construct circuits to perform multiplication and division using simple logic gates, it is fairly unusual as the complexity of the circuits makes them impractical
  - It is more usual to use dedicated circuits containing large numbers of gates or to use a microprocessor
  - We will look at such techniques in later lectures

Numeric and alphabetic codes

Video 24G



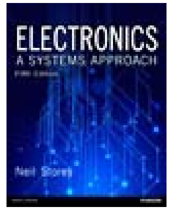
24.12

- **Binary code**
  - by far the most common way of representing numeric information
  - has advantages of simplicity and efficiency of storage

Decimal	Binary
0	0
1	1
2	10
3	11
4	100
5	101
6	110
7	111
8	1000
9	1001
10	1010
11	1011
12	1100
etc.	etc.

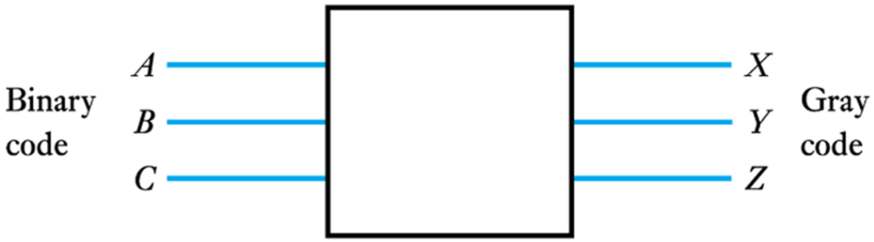
- **Gray code**
  - used in Karnaugh maps
  - also used in encoders and high-speed counters
  - only one bit changes state between adjacent values
  - allows counters/encoders to be read unambiguously

Decimal	Gray code
0	0000
1	0001
2	0011
3	0010
4	0110
5	0111
6	0101
7	0100
8	1100
9	1101
10	1111
11	1110
12	1010
13	1011
14	1001
15	1000



Examples of combinational logic design

- **Example** – see **Example 24.25** in the course text
- Design a circuit to convert 3-bit binary numbers into Gray code



  – First we produce a Truth Table

	binary			Gray		
dec	A	B	C	X	Y	Z
0	0	0	0	0	0	0
1	0	0	1	0	0	1
2	0	1	0	0	1	1
3	0	1	1	0	1	0
4	1	0	0	1	1	0
5	1	0	1	1	1	1
6	1	1	0	1	0	1
7	1	1	1	1	0	0

– From the truth table we produce Karnaugh maps

X \ C / AB	00	01	11	10
0	0	0	1	1
1	0	0	1	1

$$X = A$$

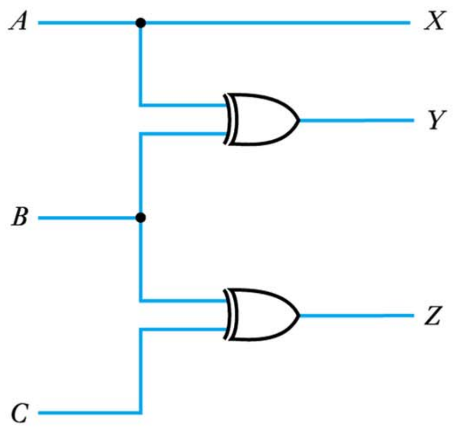
Y \ C / AB	00	01	11	10
0	0	1	0	1
1	0	1	0	1

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

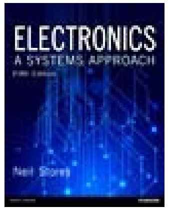
Z \ C / AB	00	01	11	10
0	0	1	1	0
1	1	0	0	1

$$Z = B\overline{C} + \overline{B}C = B \oplus C$$

– and implement the circuit



- **Further design examples**
  - The text contains further combinational logic design examples:
  - **Example 24.28:** A 4-input multiplexer



Video 24H Further Study

Further Study

- The Further Study section at the end of Chapter 24 is concerned with the design of fault tolerant arrangements, such as those used within critical systems within aircraft.

- Your task is to design a simple ‘voting’ arrangement, that allows a system to continue working correctly even in the event of a fault.
- Try the design and then look at the video.

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