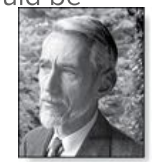
# Boolean Algebra
# พีชคณิตบูลีน

## Boolean Algebra

- The circuits in computers and other electronic devices have inputs, either 0 or 1, and produce outputs that are also 0s and 1s.
- Circuits can be constructed using any basic element that has 2 different states.
    - For example, switches that can be in either the on or off position.
- In 1938, Claude Shannon showed how the basic rules of logic, first given by George Boole in 1854 in his *The Laws of Thought,* could be used to design circuits.
    - These rules form the basis for Boolean Algebra.



## Boolean Algebra

The operation of a circuit is defined by a Boolean function that specifies the value of an output for each set of inputs.

The first step in constructing a circuit is to represent its Boolean function by an expression built up using the basic operations of Boolean algebra.

However, the expression may contain more operations than are necessary to represent the function.

- There are methods for finding an expression with the minimum number of sums and products that represents a Boolean function.
- The procedures, Karnaugh maps and the Quine–McCluskey method, are important in the design of efficient circuits.

## Chapter Summary

1. Boolean Functions
2. Representing Boolean Functions
3. Logic Gates
4. Minimization of Circuits

# 1. Boolean Functions

**Boolean Algebra**

- Boolean algebra provides the operations and the rules for working with elements from the set {0, 1}
- Three operators are defined by:
  - + (Boolean sum): 1 + 1 = 1, 1 + 0 = 1, 0 + 1 = 1, 0 + 0 = 0
  - · (Boolean product), 1 · 1 = 1, 1 · 0 = 0, 0 · 1 = 0, 0 · 0 = 0
  - - (Complement): $\overline{0} = 1$ and $\overline{1} = 0$
- Example: Find the value of $1 \cdot 0 + \overline{(0 + 1)}$

# Boolean Operations

- The complement, Boolean sum, and Boolean product correspond to the logical operators, ¬, ∨, and ∧, respectively, where 0 corresponds to **F** (false) and 1 corresponds to **T** (true).
  - AND (conjunction), denoted x ∧ y.
  - OR (disjunction), denoted x ∨ y.
  - NOT (negation), denoted ¬x.

# Boolean Expressions and Boolean Functions

**Definition 1**: Let $B = \{0, 1\}$. Then $B^n = \{(x_1, x_2, ..., x_n) \mid x_i \in B \text{ for } 1 \le i \le n\}$ is the set of all possible *n*-tuples of 0s and 1s.

The variable *x* is called a **Boolean variable** if it assumes values only from B, that is, if its only possible values are 0 and 1.

A function from $B^n$ to *B* is called a **Boolean function of degree n**.

Example 1: The function $F(x, y) = x\overline{y}$ from the set of ordered pairs of Boolean variables to the set {0, 1} is a Boolean function of degree 2 with $F(1, 1) = 0$, $F(1, 0) = 1$, $F(0, 1) = 0$, and $F(0, 0) = 0$.

# Boolean Expressions and Boolean Functions

Boolean functions can be represented using **Boolean expressions** made up from variables and Boolean operations.

Example 2: Find the values of the Boolean function represented by.

$$F(x, y, z) = xy + \overline{z}$$

## Boolean Expressions and Boolean Functions

**Definition 2**: Boolean functions *F* and *G* of *n* variables are equal if and only if $F(b_1, b_2, \ldots, b_n) = G(b_1, b_2, \ldots, b_n)$ whenever $b_1, b_2, \ldots, b_n$ belong to *B*.

Two different Boolean expressions that represent the same function are called **equivalent**.

- For instance, the Boolean expressions *xy* and *xy* + 0 and *xy* · 1 are equivalent.

## Boolean Expressions and Boolean Functions

**Definition 3**: The **complement** of the Boolean function F is the function $\overline{F}$, where $\overline{F}(x_1, \ldots, x_n) = \overline{F(x_1, \ldots, x_n)}$.

**Definition 4**: Let *F* and *G* be Boolean functions of degree *n*.

The **Boolean sum** *F* + *G* and the **Boolean product** *FG* are defined by

$$(F + G)(x_1, \ldots, x_n) = F(x_1, \ldots, x_n) + G(x_1, \ldots, x_n),$$
$$(FG)(x_1, \ldots, x_n) = F(x_1, \ldots, x_n)G(x_1, \ldots, x_n).$$

## Boolean Functions

Example 3: How many different Boolean functions of degree *n* are there?

By the product rule for counting, there are $2^n$ different *n*-tuples of 0s and 1s.

Because a Boolean function is an assignment of 0 or 1 to each of these $2^n$ different *n*-tuples, by the product rule there are $2^{(2^n)}$ different Boolean functions of degree *n*.

TABLE 3 The 16 Boolean Functions of Degree Two.

| $x$ | $y$ | $F_1$ | $F_2$ | $F_3$ | $F_4$ | $F_5$ | $F_6$ | $F_7$ | $F_8$ | $F_9$ | $F_{10}$ | $F_{11}$ | $F_{12}$ | $F_{13}$ | $F_{14}$ | $F_{15}$ | $F_{16}$ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 | 1 | 1 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |

## Boolean Functions

Example 3: How many different Boolean functions of degree *n* are there?

TABLE 4 The Number of Boolean Functions of Degree *n*.

| *Degree* | *Number* |
|---|---|
| 1 | 4 |
| 2 | 16 |
| 3 | 256 |
| 4 | 65,536 |
| 5 | 4,294,967,296 |
| 6 | 18,446,744,073,709,551,616 |

## Identities of Boolean Algebra

TABLE 5 **Boolean Identities.**

| *Identity* | *Name* |
|---|---|
| $\bar{\bar{x}} = x$ | Law of the double complement |
| $x + x = x$<br>$x \cdot x = x$ | Idempotent laws |
| $x + 0 = x$<br>$x \cdot 1 = x$ | Identity laws |
| $x + 1 = 1$<br>$x \cdot 0 = 0$ | Domination laws |
| $x + y = y + x$<br>$xy = yx$ | Commutative laws |
| $x + (y + z) = (x + y) + z$<br>$x(yz) = (xy)z$ | Associative laws |
| $x + yz = (x + y)(x + z)$<br>$x(y + z) = xy + xz$ | Distributive laws |
| $\overline{(xy)} = \bar{x} + \bar{y}$<br>$\overline{(x + y)} = \bar{x}\,\bar{y}$ | De Morgan's laws |
| $x + xy = x$<br>$x(x + y) = x$ | Absorption laws |
| $x + \bar{x} = 1$ | Unit property |
| $x\bar{x} = 0$ | Zero property |

Example 4: Show that the distributive law $x(y + z) = xy + xz$ is valid.

TABLE 6 **Verifying One of the Distributive Laws.**

| $x$ | $y$ | $z$ | $y+z$ | $xy$ | $xz$ | $x(y+z)$ | $xy+xz$ |
|---|---|---|---|---|---|---|---|
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 0 | 1 | 1 |
| 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

## Identities of Boolean Algebra

Example 6: Translate the distributive law $x + yz = (x + y)(x + z)$ into a logical equivalence using $p$, $q$, $r$ variables.

## Identities of Boolean Algebra

Example 6: Translate the distributive law $x + yz = (x + y)(x + z)$ into a logical equivalence using $p$, $q$, $r$ variables.

## Identities of Boolean Algebra

Example 7: Prove the absorption law $x(x + y) = x$ using the other identities of Boolean algebra.

# The Abstract Definition of a Boolean Algebra

**Definition 5**:

A *Boolean algebra* is a set $B$ with two binary operations $\vee$ and $\wedge$, elements 0 and 1, and a unary operation $^-$ such that these properties hold for all $x$, $y$, and $z$ in $B$:

| | |
|---|---|
| $x \vee 0 = x$, $x \wedge 1 = x$ | Identity laws |
| $x \vee \bar{x} = 1$, $x \wedge \bar{x} = 0$ | Complement laws |
| $(x \vee y) \vee z = x \vee (y \vee z)$, $(x \wedge y) \wedge z = x \wedge (y \wedge z)$ | Associative laws |
| $x \vee y = y \vee x$, $x \wedge y = y \wedge x$ | Commutative laws |
| $x \vee (y \wedge z) = (x \vee y) \wedge (x \vee z)$, $x \wedge (y \vee z) = (x \wedge y) \vee (x \wedge z)$ | Distributive laws |

# 2. Representing Boolean Functions

**2 Important problems of Boolean Algebra**

- Given the values of a Boolean function, how can a Boolean expression that represents this function can be found?
  - Any Boolean function can be represented by a Boolean sum of Boolean products of the variables and their complements.
- Is there a smaller set of operators that can be used to represent all Boolean functions?
  - All Boolean functions can be represented using only one operator.
- Both of these problems have practical importance in circuit design.

# Sum-of-Products Expansions

Example 1: Find Boolean expressions that represent the functions $F(x, y, z)$ and $G(x, y, z)$, which are given in Table 1.

TABLE 1

| $x$ | $y$ | $z$ | $F$ | $G$ |
|---|---|---|---|---|
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 |

- Each combination of values of the variables for which the function has the value 1 leads to a Boolean product of the variables or their complements.

# Sum-of-Products Expansions

**Definition 1**: A ***literal*** is a Boolean variable or its complement.

A ***minterm*** of the Boolean variables $x_1, x_2, \ldots, x_n$ is a Boolean product $y_1 y_2 \cdots y_n$, where $y_i = x_i$ or $y_i = \bar{x}_i$.

- Hence, a minterm is a product of $n$ literals, with one literal for each variable.

A minterm has the value 1 for one and only one combination of values of its variables.

- More precisely, the minterm $y_1 y_2 \cdots y_n$ is 1 if and only if each $y_i$ is 1, and this occurs if and only if $x_i = 1$ when $y_i = x_i$ and $x_i = 0$ when $y_i = \bar{x}_i$.

## Sum-of-Products Expansions

Example 2: Find a minterm that equals 1, if $x_1 = x_3 = 0$ and $x_2 = x_4 = x_5 = 1$, and equals 0 otherwise.

## Sum-of-Products Expansions

By taking Boolean sums of distinct minterms we can build up a Boolean expression with a specified set of values.

- In particular, a Boolean sum of minterms has the value 1 when exactly one of the minterms in the sum has the value 1.
- It has the value 0 for all other combinations of values of the variables.

Consequently, given a Boolean function, a Boolean sum of minterms can be formed that has the value 1 when this Boolean function has the value 1, and has the value 0 when the function has the value 0.

The minterms in this Boolean sum correspond to those combinations of values for which the function has the value 1.

## Sum-of-Products Expansions

**Definition 2**: The sum of minterms that represents the function is called the **sum-of-products expansion** or the **disjunctive normal form** of the Boolean function.

## Sum-of-Products Expansions

Example 3: Find the sum-of-products expansion for the function $F(x, y, z) = (x + y)\,\overline{z}$.

- There are two ways to find the sum-of-products expansion.
- First, Boolean Identities to expand the product and simplify.

## Sum-of-Products Expansions

Example 3: Find the sum-of-products expansion for the function $F(x, y, z) = (x + y)\bar{z}$.

- Second, form the Boolean sum of the minterms corresponding to each row of the table that has the value 1.

## Functional Completeness

**Definition 3**: Because every Boolean function can be represented using the Boolean operators $\cdot$ , +, and $\bar{}$, we say that the set $\{\cdot, +, \bar{}\}$ is ***functionally complete***.

Can we find a smaller set of functionally complete operators?

- We can do so if one of the 3 operators of this set can be expressed in terms of the other 2. This can be done using one of De Morgan's laws.

## Functional Completeness

To eliminate all Boolean sums, the set $\{\cdot, \bar{}\}$ is functionally complete since

$$x + y = \overline{\bar{x}\,\bar{y}}$$

To eliminate all Boolean products, the set $\{+, \bar{}\}$ is functionally complete since

$$xy = \overline{\bar{x} + \bar{y}}$$

The ***NAND*** operator, denoted by |, is defined by
1 | 1 = 0, and 1 | 0 = 0 | 1 = 0 | 0 = 1.

The **NOR** operator, denoted by ↓, is defined by
0 ↓ 0 = 1, and 1 ↓ 0 = 0 ↓ 1 = 1 ↓ 1 = 0.

Both of the sets {|} and {↓} are functionally complete.

## 3. Logic Gates

- Boolean algebra is used to model the circuitry of electronic devices.
- Each input and each output Links of such a device can be thought of as a member of the set {0, 1}.
- A computer, or other electronic device, is made up of a number of circuits.
- Each circuit can be designed using the rules of Boolean algebra to perform a variety of tasks.
- The basic elements of circuits are called **gates** where each type of gate implements a Boolean operation.

# Logic Gates

- The circuits in this chapter give output that depends only on the input, and not on the current state of the circuit.
  - In other words, these circuits have no memory capabilities.
  - Such circuits are called **combinational circuits** or **gating networks**.

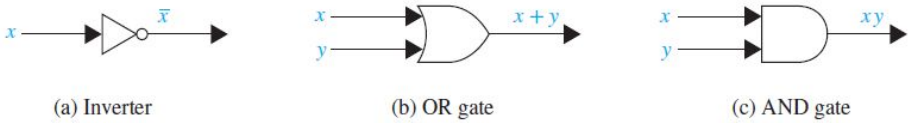
# Logic Gates

- The circuits in this chapter give output that depends only on the input, and not on the current state of the circuit.
  - In other words, these circuits have no memory capabilities.
  - Such circuits are called **combinational circuits** or **gating networks**.
- Combinational circuits are constructed using 3 types of elements
  - **Inverter**
  - **OR gate**
  - **AND gate**

# Logic Gates

Combinational circuits are constructed using 3 types of elements

- **Inverter** accepts the value of one Boolean variable as input and produces the complement of this value as its output.
- **OR gate** accepts 2+ inputs and output the Boolean sum of values.
- **AND gate** accepts 2+ inputs and output the Boolean product of values.

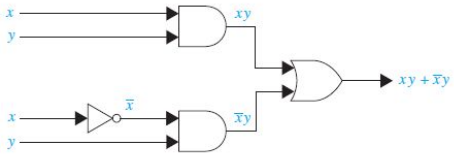



(a) Inverter (b) OR gate (c) AND gate

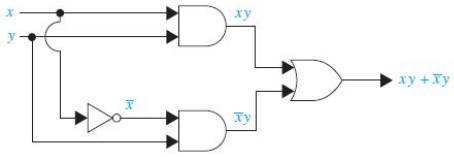
# Combinations of Gates

Combinational circuits can be constructed using a combination of inverters, OR gates, and AND gates.

- Some gates may share inputs.
- Output from a gate may be used as input by other gates.




Example 1: Show 2 ways of constructing a circuit that produces the output $xy + x\bar{y}$



# Combinations of Gates

Example 2: Construct circuits that produce these outputs

(a) $(x + y)\bar{x}$

(b) $\bar{x}\,\overline{(y + \bar{z})}$

(c) $(x + y + z)(\bar{x}\,\bar{y}\,\bar{z})$

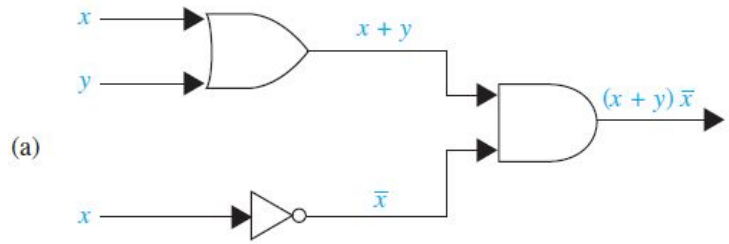
# Combinations of Gates

Example 2: Construct circuits that produce these outputs

(a) $(x + y)\bar{x}$

(b) $\bar{x}\,\overline{(y + \bar{z})}$

(c) $(x + y + z)(\bar{x}\,\bar{y}\,\bar{z})$



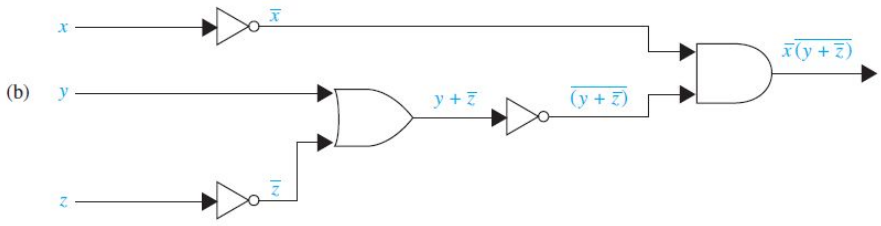

# Combinations of Gates

Example 2: Construct circuits that produce these outputs

(a) $(x + y)\bar{x}$

(b) $\bar{x}\,\overline{(y + \bar{z})}$

(c) $(x + y + z)(\bar{x}\,\bar{y}\,\bar{z})$



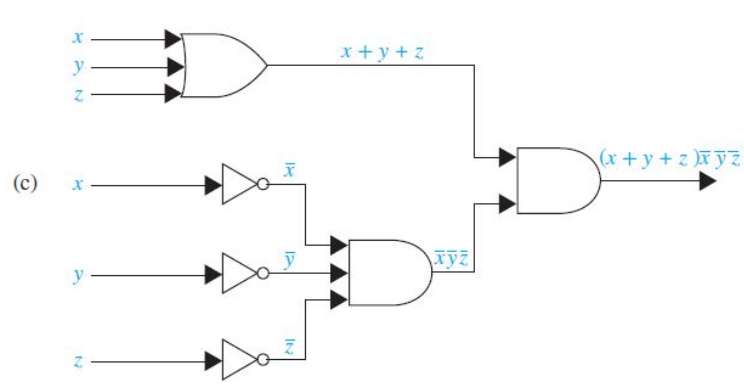

# Combinations of Gates

Example 2: Construct circuits that produce these outputs

(a) $(x + y)\bar{x}$

(b) $\bar{x}\,\overline{(y + \bar{z})}$

(c) $(x + y + z)(\bar{x}\,\bar{y}\,\bar{z})$



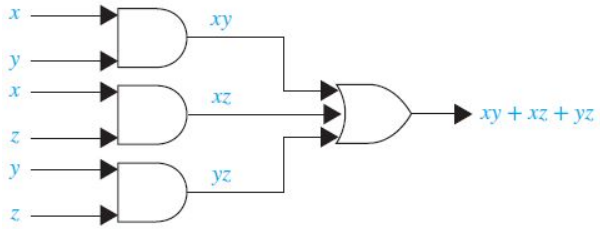
# Examples of Circuits

Example 3: A Committee of 3 individuals decides issues for an organisation.

Each individual votes either yes/no for each proposal that arises.

A proposal is passed if it receives at least 2 yes votes.

Design a circuit that determines whether a proposal passes.




# Examples of Circuits

Example 4: Sometimes light fixtures are controlled by more than one switch.

Circuits need to be designed so that flipping any one of the switches for the fixture turns the light ON when it is OFF and turns the light OFF when it is ON.

Design circuits that accomplish this when there are 2 switches.

# Examples of Circuits

Example 5: Design circuits that accomplish this when there are 3 switches.

# Adders

Logic circuits can be used to carry out addition of 2 positive integers from their binary expansions.

The first step is to build a ***half adder*** that adds 2 bits (x + y), but which does not accept a carry from a previous addition.

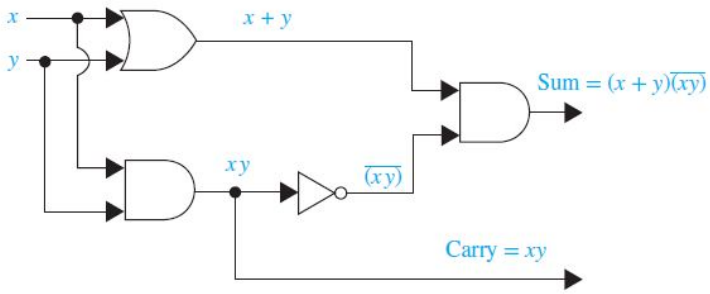
- The output will consist of 2 bits, *s* and *c*.
    - *s* is the sum bit
    - *c* is the carry bit

Since the circuit has more than one output, it is a ***multiple output circuit***.

## Half Adder

TABLE 3
**Input and Output for the Half Adder.**

| *Input* | | *Output* | |
|---|---|---|---|
| $x$ | $y$ | $s$ | $c$ |
| 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 |

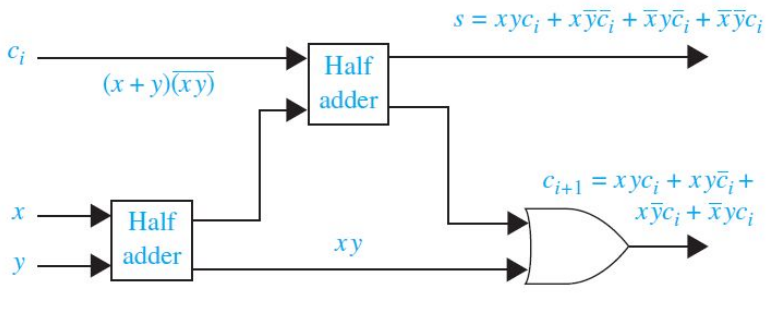



## Full Adder

A **full adder** is used to compute the sum bit and the carry bit when 2 bits and a carry are added.

TABLE 4
**Input and Output for the Full Adder.**

| *Input* | | | *Output* | |
|---|---|---|---|---|
| $x$ | $y$ | $c_i$ | $s$ | $c_{i+1}$ |
| 1 | 1 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 |




## 4. Minimization of Circuits

- The sum-of-products expansion may contain many more terms than are necessary.
- For instance, consider the circuit that has output 1 if and only if x = y = z = 1 or x = z = 1 and y = 0.
- The sum-of-products expansion of this circuit is xyz + x $\bar{y}$z. They can be combined to a simpler expression as

$$\begin{aligned} xyz + x\bar{y}z &= (y + \bar{y})(xz) \\ &= 1 \cdot (xz) \\ &= xz. \end{aligned}$$

- Finding such a simpler expression is called **minimization of the Boolean function**.

## Minimization of Circuits

- Reducing the number of gates on a chip can lead to a more reliable circuit and can reduce the cost to produce the chip.
- Also, minimization makes it possible to fit more circuits on the same chip.
- Furthermore, minimization reduces the number of inputs to gates in a circuit. This reduces the time used by a circuit to compute its output.
- One of the well-known minimization procedures is **Karnaugh maps** (or **K-maps**).
  - Useful in minimising circuits with up to 6 variables, although become complex for 5 variables.

# Karnaugh Maps in 2 variables

- To reduce the number of terms in a Boolean expression representing a circuit, it is necessary to find terms to combine.
- There are 4 possible minterms in the sum-of-products expansion of a Boolean function in the 2 variables $x$ and $y$.
- A K-map for a Boolean function in these 2 variables consists of 4 cells, where a 1 is placed in the cell representing a minterm if this minterm is present in the expansion.

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | $xy$ | $x\bar{y}$ |
| $\bar{x}$ | $\bar{x}y$ | $\bar{x}\bar{y}$ |

# Karnaugh Maps in 2 variables

Example 1: Find the K-maps for (a) $xy + \bar{x}y$, (b) $x\bar{y} + \bar{x}y$, and (c) $x\bar{y} + \bar{x}y + \bar{x}\bar{y}$.

Solution: Include a 1 in a cell when the minterm represented by this cell is present in the sum-of-products expansion.

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | 1 | |
| $\bar{x}$ | 1 | |

(a)

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | | 1 |
| $\bar{x}$ | 1 | |

(b)

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | | 1 |
| $\bar{x}$ | 1 | 1 |

(c)

# Simplifying the sum-of-products expansions

Whenever there are 1s in 2 adjacent cells in the K-map, the minterms represented by these cells can be combined into a product involving just 1 of the variables.

- Moreover, if 1s are in all 4 cells, the 4 minterms can be combined into one term, namely, the Boolean expression 1 that involves none of the variables.

We circle blocks of cells in the K-map that represent minterms that can be combined and then find the corresponding sum of products.

The goal is to identify the largest possible blocks, and to cover all the 1s with the fewest blocks using the largest blocks first and always using the largest possible blocks.

# Karnaugh Maps in 2 variables

Example 2: Simplify the sum-of-products expansions.

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | 1 | |
| $\bar{x}$ | 1 | |

(a)

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | | 1 |
| $\bar{x}$ | 1 | |

(b)

| | $y$ | $\bar{y}$ |
|---|---|---|
| $x$ | | 1 |
| $\bar{x}$ | 1 | 1 |

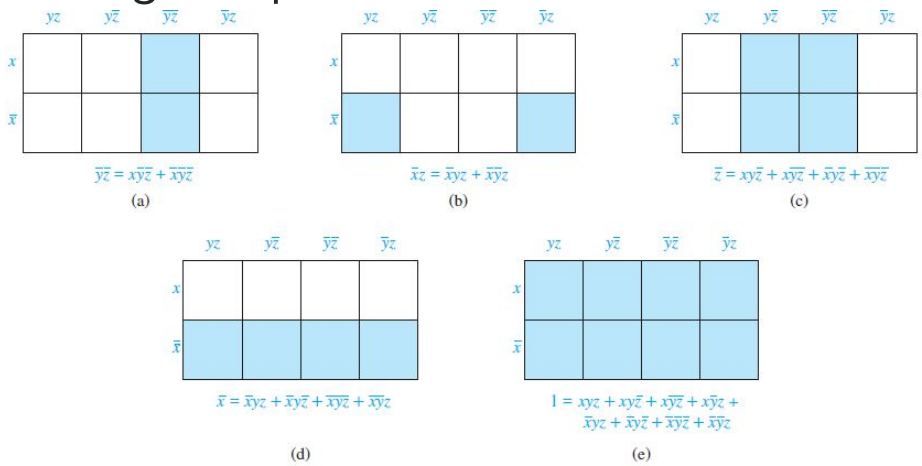
(c)

# Karnaugh Maps in 3 variables

There are 8 possible minterms in 3 variables.

| | $yz$ | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|---|---|---|---|---|
| $x$ | $xyz$ | $xy\bar{z}$ | $x\bar{y}\bar{z}$ | $x\bar{y}z$ |
| $\bar{x}$ | $\bar{x}yz$ | $\bar{x}y\bar{z}$ | $\bar{x}\bar{y}\bar{z}$ | $\bar{x}\bar{y}z$ |

# Karnaugh Maps in 3 variables




# Karnaugh Maps in 3 variables

Example 3: Use K-maps to minimize these sum-of-products expansions.

(a) $xy\bar{z} + x\bar{y}\,\bar{z} + \bar{x}yz + \bar{x}\,\bar{y}\,\bar{z}$

(b) $x\bar{y}z + x\bar{y}\,\bar{z} + \bar{x}yz + \bar{x}\,\bar{y}z + \bar{x}\,\bar{y}\,\bar{z}$

(c) $xyz + xy\bar{z} + x\bar{y}z + x\bar{y}\,\bar{z} + \bar{x}yz + \bar{x}\,\bar{y}z + \bar{x}\,\bar{y}\,\bar{z}$

(d) $xy\bar{z} + x\bar{y}\,\bar{z} + \bar{x}\,\bar{y}z + \bar{x}\,\bar{y}\,\bar{z}$

# Karnaugh Maps in 4 variables

| | $yz$ | $y\bar{z}$ | $\bar{y}\bar{z}$ | $\bar{y}z$ |
|---|---|---|---|---|
| $wx$ | $wxyz$ | $wxy\bar{z}$ | $wx\bar{y}\bar{z}$ | $wx\bar{y}z$ |
| $w\bar{x}$ | $w\bar{x}yz$ | $w\bar{x}y\bar{z}$ | $w\bar{x}\bar{y}\bar{z}$ | $w\bar{x}\bar{y}z$ |
| $\bar{w}\bar{x}$ | $\bar{w}\bar{x}yz$ | $\bar{w}\bar{x}y\bar{z}$ | $\bar{w}\bar{x}\bar{y}\bar{z}$ | $\bar{w}\bar{x}\bar{y}z$ |
| $\bar{w}x$ | $\bar{w}xyz$ | $\bar{w}xy\bar{z}$ | $\bar{w}x\bar{y}\bar{z}$ | $\bar{w}x\bar{y}z$ |