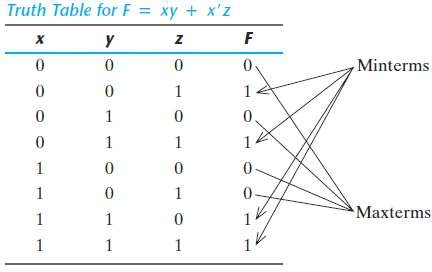
1



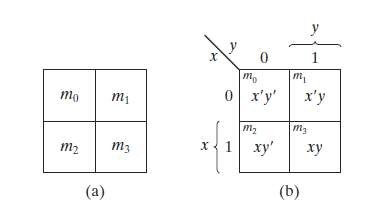
اسم التدريسي: م.م علياء محمد جواد

اسم المادة : تقنيات رقمية

المرحلة :الثانية

السنة الدراسية :2023\_2024

Karnaugh mapاسم المحاظرة :



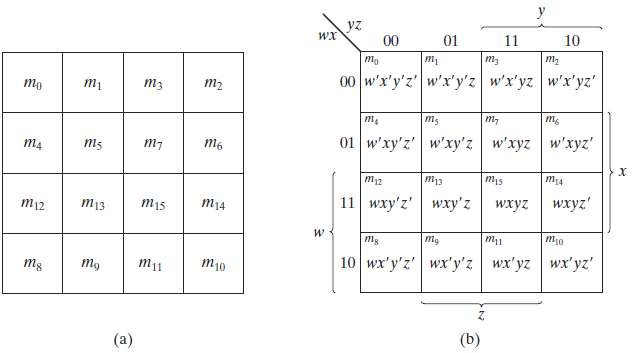
Karnaugh map (Recap the maps)

**Two-Variable Map**

F=xy+x’y

**Three-Variable Map**

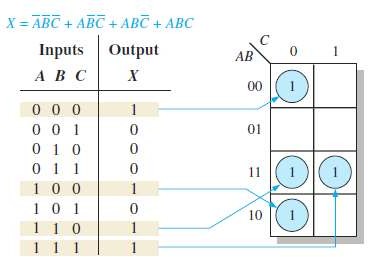
2



Karnaugh map (Recap the maps)

Four Variable Map

3



Karnaugh map

Mapping Directly from a Truth Table

* Truth table gives the output of a Boolean expression for all possible input variable combinations.
* In the figure below, you can see that the Boolean expression, the truth table, and the Karnaugh map are simply different ways to represent a logic function.

4

Karnaugh map

“Don’t Care” Conditions

* Sometimes a situation arises in which some input variable combinations are not allowed. For example, in the BCD code there are six invalid combinations: 1010, 1011, 1100, 1101, 1110, and 1111.
* Since these unallowed states will never occur in an application involving the BCD code, they can be treated as **“don’t care”** terms with respect to their effect on the output.
* That is, for these “don’t care” terms either a 1 or a 0 may be assigned to the output; it really does not matter since they will never occur.

5



Karnaugh map

“Don’t Care” Conditions

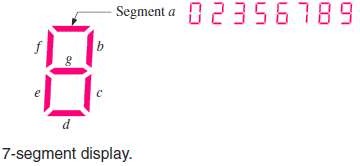
* The figure below shows that for each “don’t care” term, an X is placed in the cell. When grouping the 1s, the Xs can be treated as 1s to get the simplest expression.

The truth table describes a logic function that has a 1 output only when the BCD code for 7, 8, or 9 is present on the inputs.

If the “don’t cares” are used as 1s, the resulting expression for the function is *A* + *BCD*, as indicated in part (b).

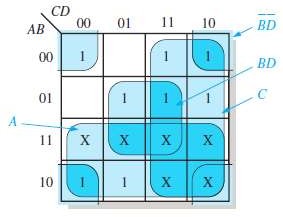
If the “don’t cares” are not used as 1s, the resulting expression is

6



Karnaugh map

* Example: In a 7-segment display, each of the seven segments is activated for various digits. For example, segment *a* is activated for the digits 0, 2, 3, 5, 6, 7, 8, and 9, as illustrated in the figure below. Since each digit can be represented by a BCD code, derive an SOP expression for segment *a* using the variables *ABCD* and then minimize the expression using a Karnaugh map.

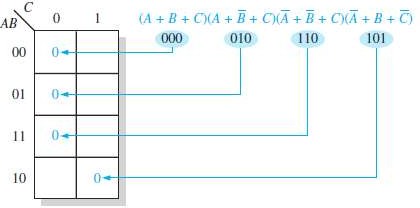
7



Karnaugh map

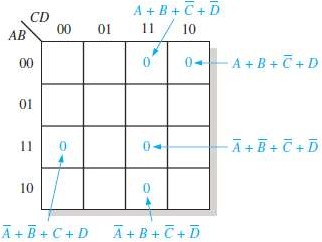
* Solution: The expression for segment *a* is
* Each term in the expression represents one of the digits in which segment *a* is used. X’s (don’t cares) are entered for those states that do not occur in the BCD code.

8



Karnaugh Map POS Minimization

* In the previous slides, you studied the minimization of an SOP expression using a Karnaugh map.
* For the next slides, we focus on POS expressions. The approaches are much the same except that with POS expressions, 0s representing the standard sum terms are placed on the Karnaugh map instead of 1s.
* Mapping a Standard POS Expression

9



Karnaugh Map POS Minimization

* Example: Map the following standard POS expression on a Karnaugh map:
* Solution:

10



Karnaugh Map Simplification of POS Expressions

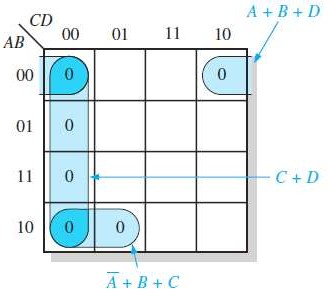
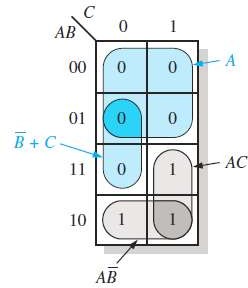
* The process for minimizing a POS expression is that you group 0s to produce minimum sum terms.
* Example: Use a Karnaugh map to minimize the following standard POS expression:  Also, derive the equivalent SOP expression.

Solution: The combinations of binary values of the expression are: (0 + 0 + 0)(0 + 0 + 1)(0 + 1 + 0)(0 + 1 + 1)(1 + 1 + 0)

-The sum term for each blue group is shown in the figure, and the resulting POS expression is

-Grouping the 1s as shown by the gray areas yields an SOP expression

note: 

11



Karnaugh Map Simplification of POS Expressions

Example: Use a Karnaugh map to minimize the following POS expression: 

* Solution: The first term must be expanded into *A* + *B* + *C* + *D* and *A* + *B*

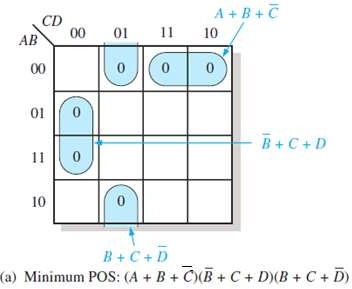
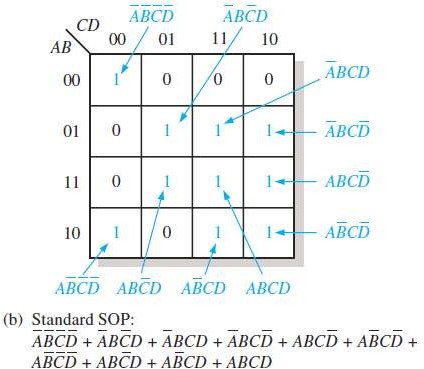
+ *C* + *D* to get a standard POS expression, which is then mapped; and the cells are grouped as shown in the figure below

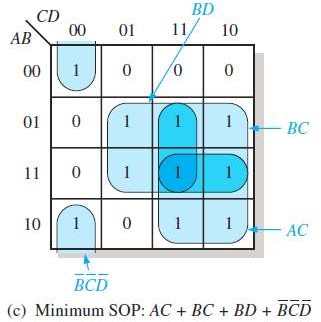
* + Therefore the resulting minimum POS expression is

12

Converting Between POS and SOP Using the Karnaugh Map

* For a POS expression, all the cells in the Karnaugh map that do not contain 0s contain 1s, from which the SOP expression is derived.
* Likewise, for an SOP expression, all the cells that do not contain 1s contain 0s, from which the POS expression is derived.

13



Converting Between POS and SOP Using the Karnaugh Map

* Example: Using a Karnaugh map, convert the following standard POS expression into a minimum POS expression, a standard SOP expression, and a minimum SOP expression.



* Solution:

14

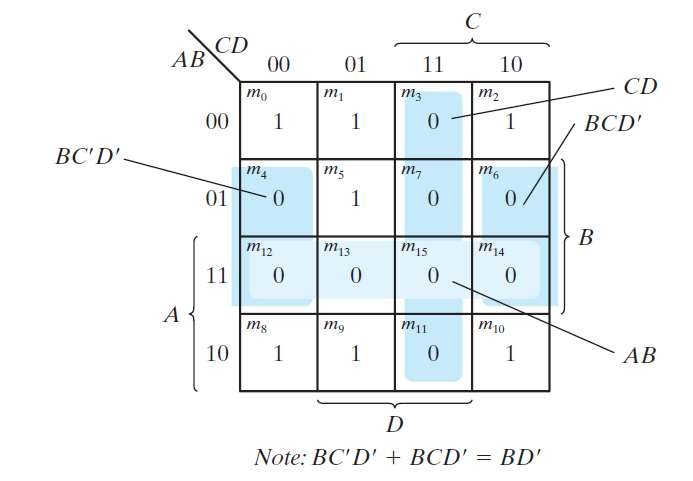


Example: Simplifying the Boolean Function

* EX: Simplify the following Boolean function into product-of-sums form
* Sol:

1. First, we combine the squares marked with 0’s, as shown in the diagram below, to obtain the simplified complemented function:
2. Then, we apply DeMorgan’s theorem to obtain the simplified function in product form:

of-sums

15

**A *Prime Implicant and Essential Prime Implicant***

* **A *prime implicant*** is a product term obtained by combining the maximum possible number of adjacent squares in the map.
* If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential.*
* a single 1 on a map represents a prime implicant if it is not adjacent to any other 1’s.
* Two adjacent 1’s form a prime implicant, provided that they are not within a group of four adjacent squares.
* Four adjacent 1’s form a prime implicant if they are not within a group of eight adjacent squares, and so on.
* The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it.
* The prime implicant is essential if it is the only prime implicant that covers the minterm.

16



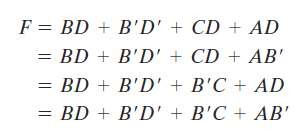
Example 1

Find all the prime implicants for the following Boolean function, and determine which are essential:

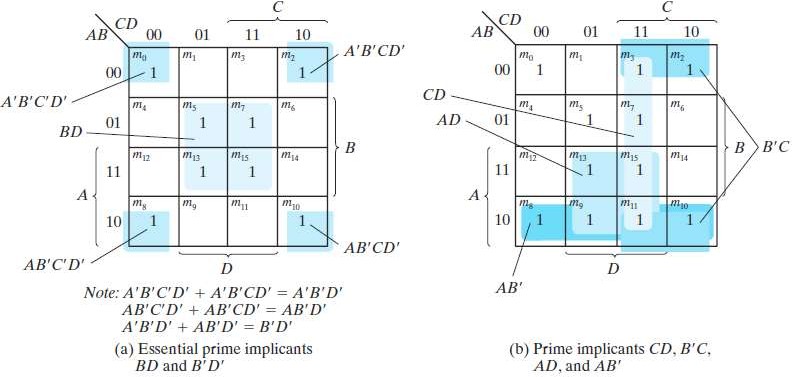
Note: The procedure for finding the simplified expression from the map requires that we first determine all the essential prime implicants.

The simplified expression is obtained from the logical sum of all the essential prime implicants, plus other prime implicants that may be needed to cover any remaining minterms not covered by the essential prime implicants.

17



Example 1: solution



* Note: There are four possible ways that the function can be expressed with four product terms of two literals each:

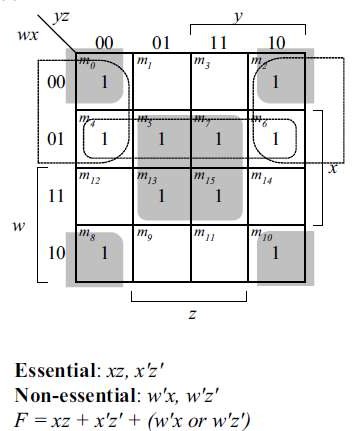
18



Example 2

* Find all the prime implicants for the following Boolean function, and determine which are essential:

19



Example2: Solution

Solution

20

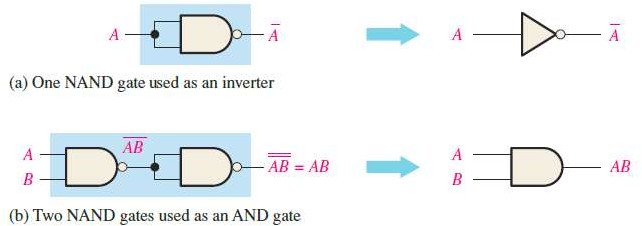
Exercise

* Find all the prime implicants for the following Boolean function, and determine which are essential (note the d function represents the don’t-care conditions):

*F*(W, X, Y, Z) = Σ(0, 3, 13, 15),

d(W, X, Y, Z)= Σ(4, 6, 8, 10)

21



The NAND Gate as a Universal Logic Element

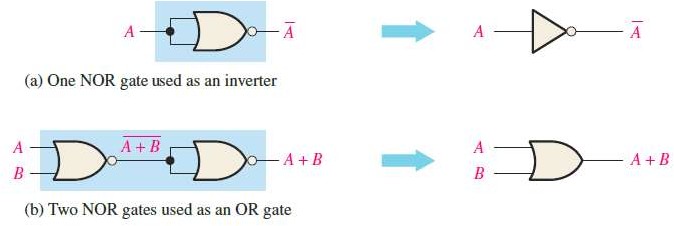
* The NAND gate is a **universal gate** because it can be used to produce the NOT, the AND, the OR, and the NOR functions.

22



The NAND Gate as a Universal Logic Element

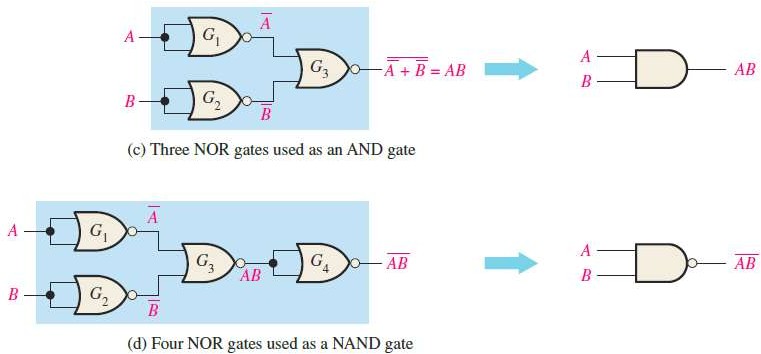
23



The NOR Gate as a Universal Logic Element

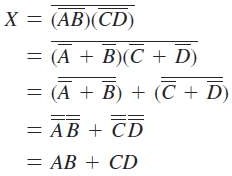
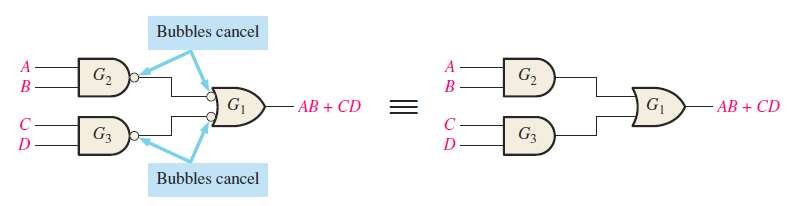
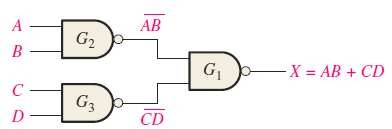
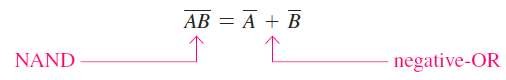
* Like the NAND gate, the NOR gate can be used to produce the NOT, AND, OR, and NAND functions.

24



The NOR Gate as a Universal Logic Element

25



Combinational Logic Using NAND Gates

A NAND gate can function as either a NAND or a negative-OR

Example: Develop the output expression for the figure (a) below:

(a)

(b)

(c)

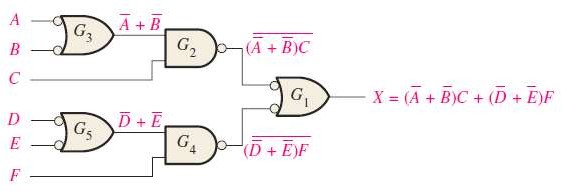
Solution:

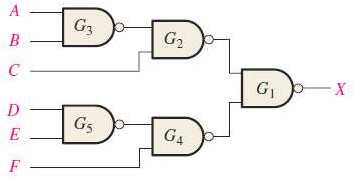
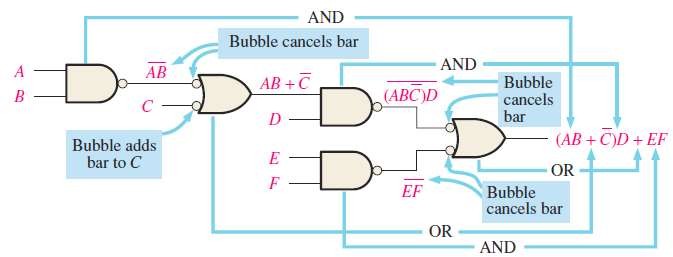
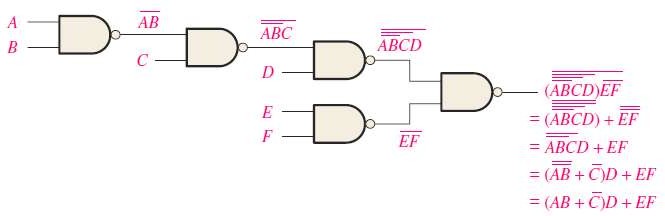
26



**NAND Logic Diagrams Using Dual Symbols**

* The NAND symbol and the **negative-OR** symbol are called *dual symbols.*
* All logic diagrams using NAND gates should be drawn with each gate represented by either a NAND symbol or the equivalent negative-OR symbol.
* Although using all NAND symbols as in Figure (a) is correct, the diagram in part (b) is much easier to “read” and is the preferred method.

27



**NAND Logic Diagrams Using Dual Symbols**

Example: Redraw the logic diagram and develop the output expression for the circuit below using the appropriate dual symbols.

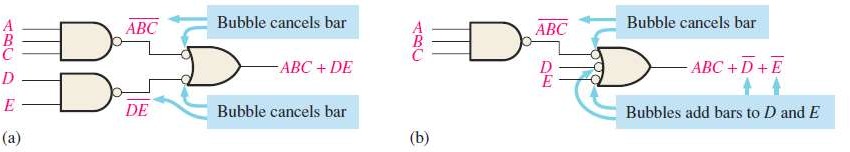
Solution:

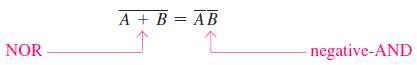
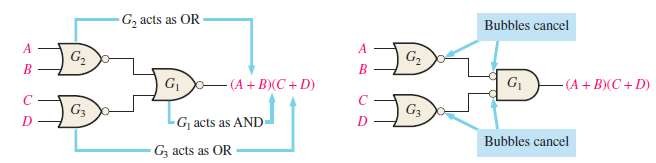
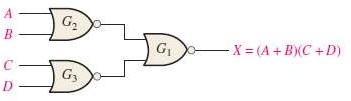
28



Combinational Logic Using NAND Gates

* Example: Implement each expression with NAND logic using appropriate dual symbols:
* Solution:

29



Combinational Logic Using NOR Gates

* A NOR gate can function as either a NOR or a **negative-AND**

Example: Develop the output expression for the figure (a) below

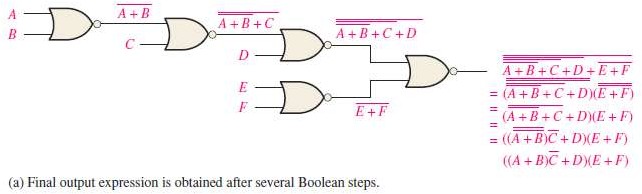
(a)

(b)

(c)

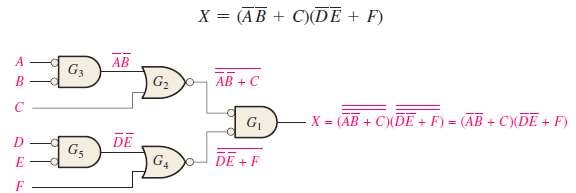
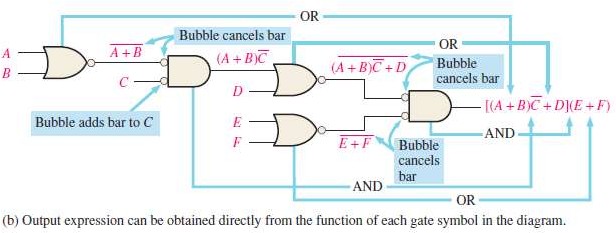
Solution:

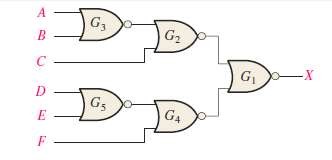
30



**NOR Logic Diagram Using Dual Symbols**

* As with NAND logic, the purpose for using the dual symbols is to make the logic diagram easier to read and analyze.
* Example: The circuit in part (a) is redrawn with dual symbols in part (b), notice that all output-to-input connections between gates are bubble-to-bubble or nonbubble-to-nonbubble.
* Solution:

31



Combinational Logic Using NOR Gates

* Example: Using appropriate dual symbols, redraw the logic diagram and develop the output expression for the circuit in the figure below:
* Solution:

32

