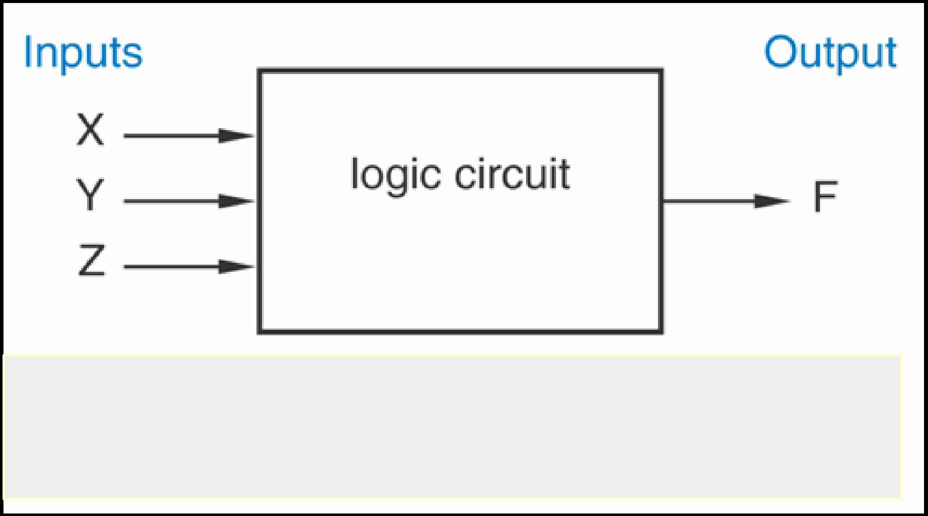
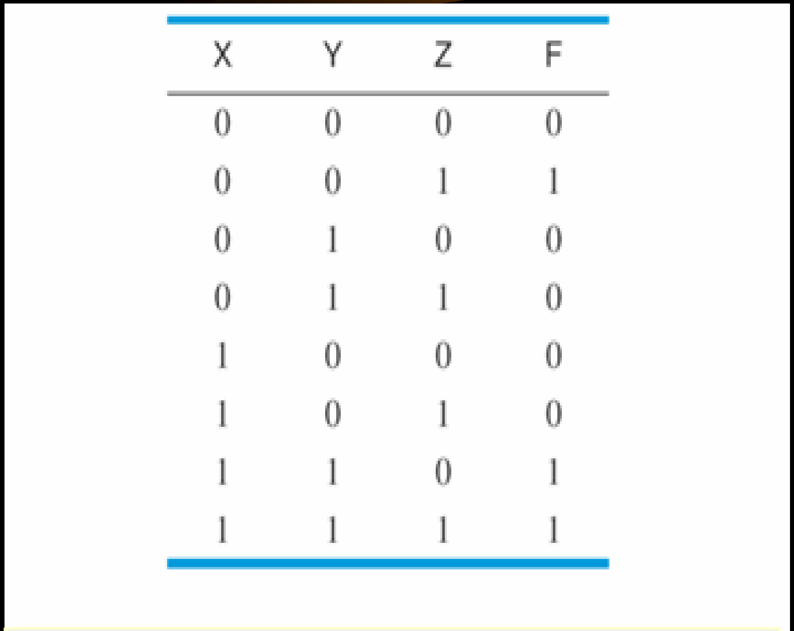
Computer Organization

Arithmetic Logic Unit

Background

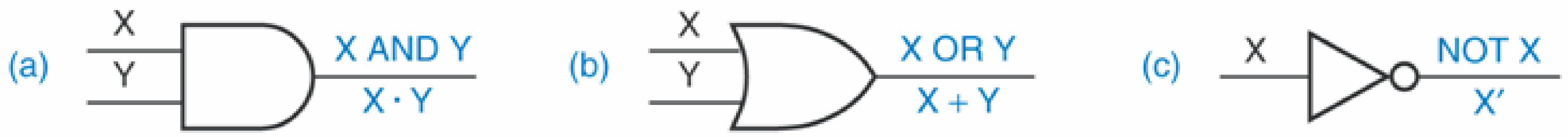
Department of Computer Science
Missouri University of Science & Technology
hurson@mst.edu





X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Truth Table for a combinational logic

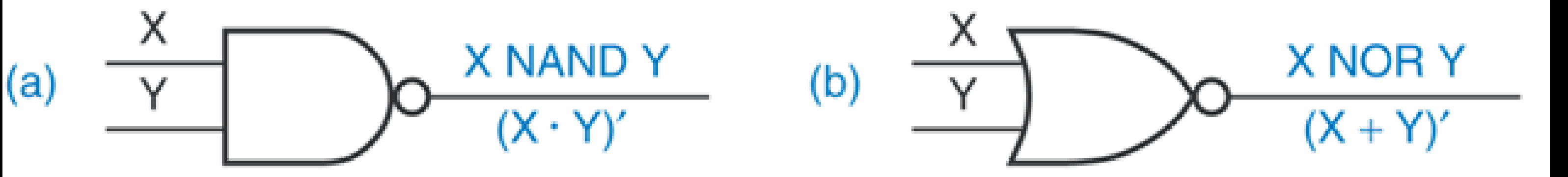


X	Y	X AND Y
0	0	0
0	1	0
1	0	0
1	1	1

X	Y	X OR Y
0	0	0
0	1	1
1	0	1
1	1	1

X	NOT X
0	1
1	0

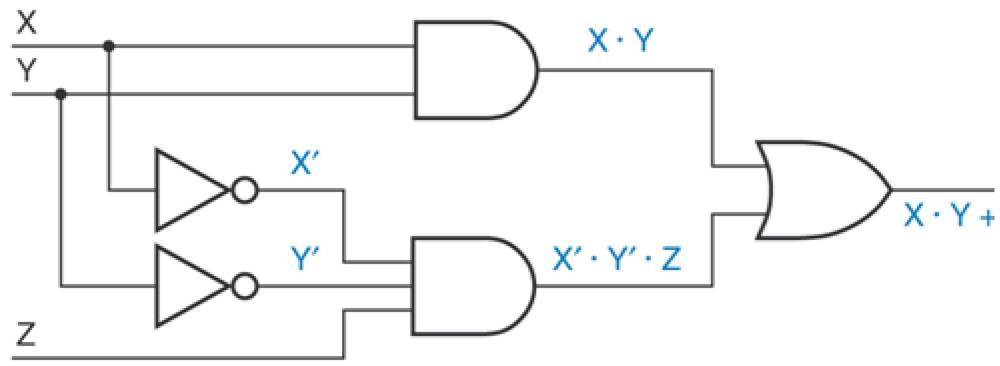
Basic logic elements: (a) AND; (b) OR; (c) NOT (inverter).



X	Y	X NAND Y
0	0	1
0	1	1
1	0	1
1	1	0

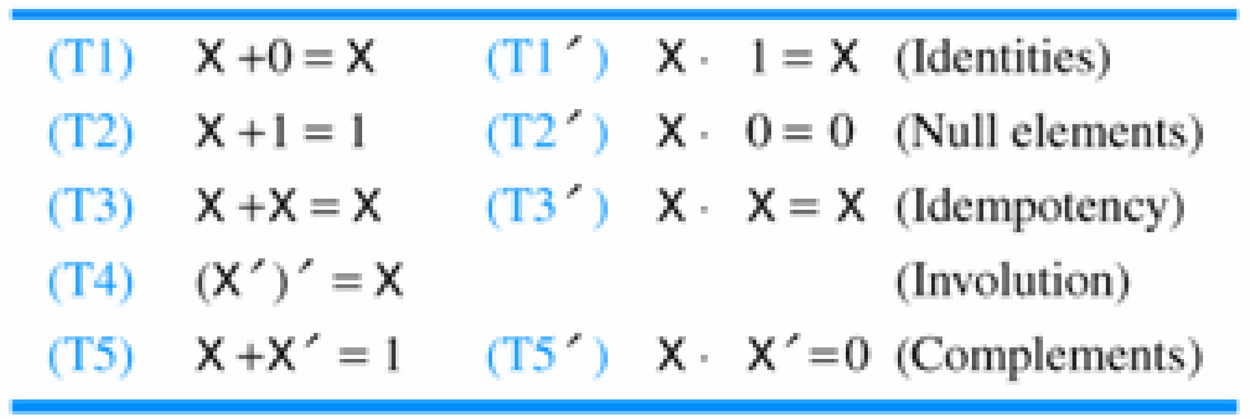
X	Y	X NOR Y
0	0	1
0	1	0
1	0	0
1	1	0

Inverting gates: a) NAND, b) NOR



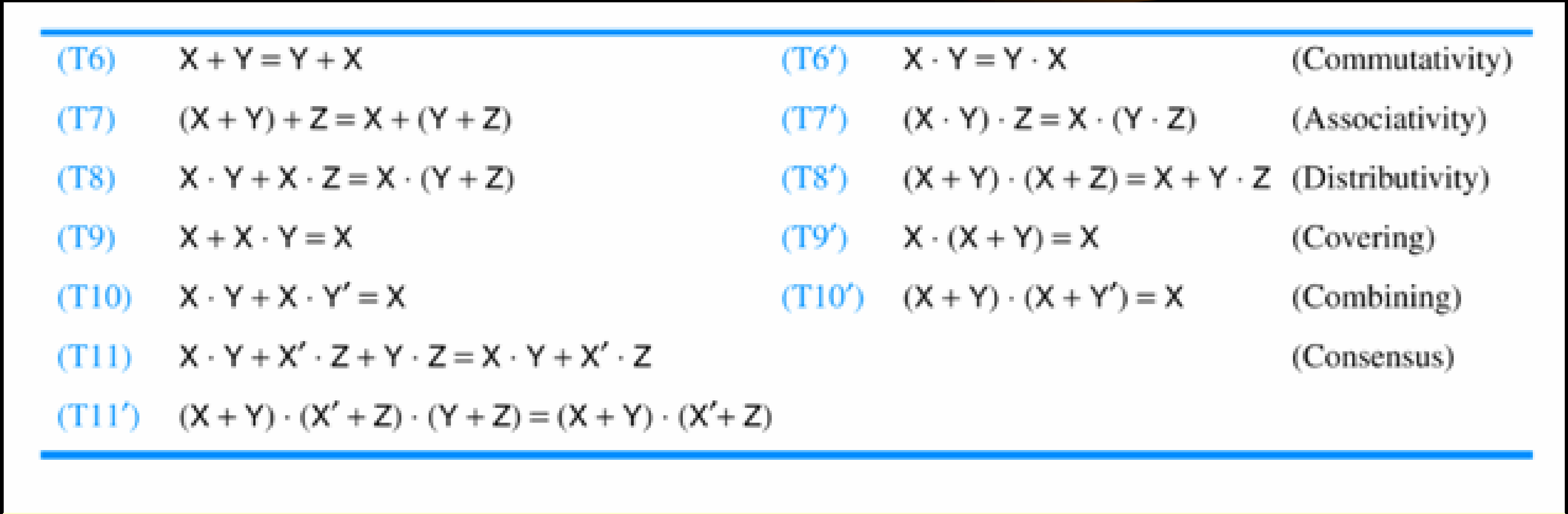
X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

A combinational circuit with its truth table



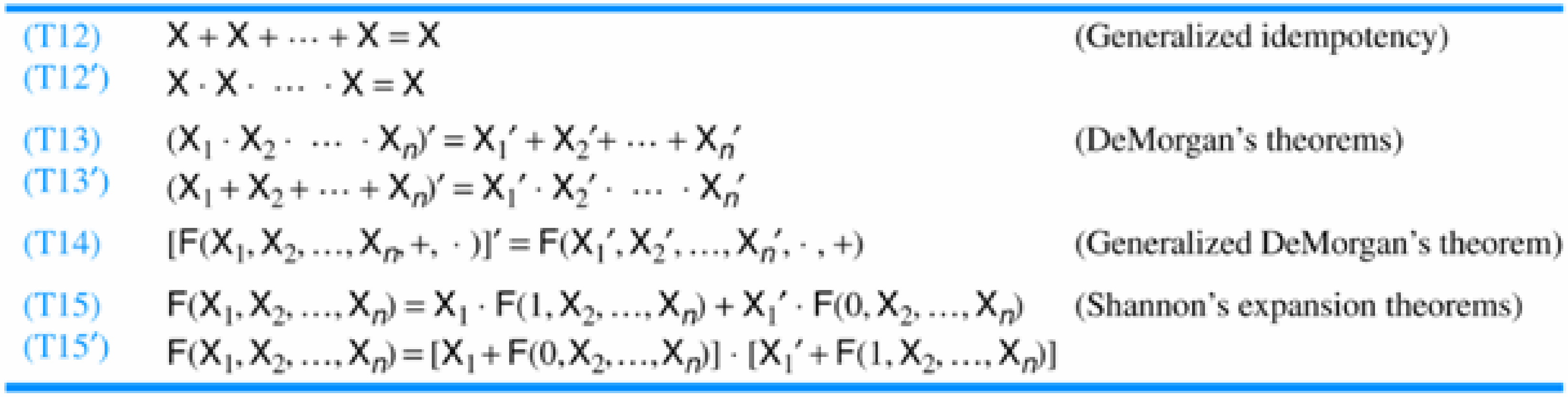
(T1)	$X + 0 = X$	(T1′)	$X \cdot 1 = X$	(Identities)
(T2)	$X + 1 = 1$	(T2′)	$X \cdot 0 = 0$	(Null elements)
(T3)	$X + X = X$	(T3′)	$X \cdot X = X$	(Idempotency)
(T4)	$(X')' = X$			(Involution)
(T5)	$X + X' = 1$	(T5′)	$X \cdot X' = 0$	(Complements)

Switching algebra theorems with one variable



(T6)	$X + Y = Y + X$	(T6′)	$X \cdot Y = Y \cdot X$	(Commutativity)
(T7)	$(X + Y) + Z = X + (Y + Z)$	(T7′)	$(X \cdot Y) \cdot Z = X \cdot (Y \cdot Z)$	(Associativity)
(T8)	$X \cdot Y + X \cdot Z = X \cdot (Y + Z)$	(T8′)	$(X + Y) \cdot (X + Z) = X + Y \cdot Z$	(Distributivity)
(T9)	$X + X \cdot Y = X$	(T9′)	$X \cdot (X + Y) = X$	(Covering)
(T10)	$X \cdot Y + X \cdot Y' = X$	(T10′)	$(X + Y) \cdot (X + Y') = X$	(Combining)
(T11)	$X \cdot Y + X' \cdot Z + Y \cdot Z = X \cdot Y + X' \cdot Z$			(Consensus)
(T11′)	$(X + Y) \cdot (X' + Z) \cdot (Y + Z) = (X + Y) \cdot (X' + Z)$			

Switching algebra theorems with two or three variables variable

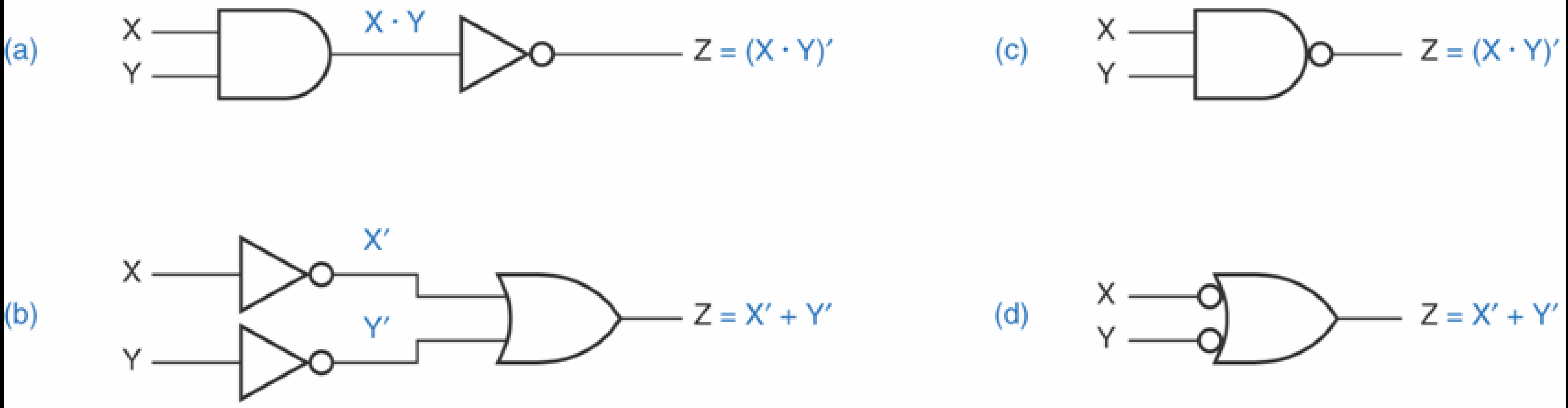


(T12)	$X + X + \cdots + X = X$	(Generalized idempotency)
(T12′)	$X \cdot X \cdot \ \cdots \ \cdot X = X$	
(T13)	$(X_1 \cdot X_2 \cdot \ \cdots \ \cdot X_n)' = X_1' + X_2' + \cdots + X_n'$	(DeMorgan's theorems)
(T13′)	$(X_1 + X_2 + \cdots + X_n)' = X_1' \cdot X_2' \cdot \ \cdots \ \cdot X_n'$	
(T14)	$[F(X_1, X_2, \ldots, X_n, +, \cdot)]' = F(X_1', X_2', \ldots, X_n', \cdot, +)$	(Generalized DeMorgan's theorem)
(T15)	$F(X_1, X_2, \ldots, X_n) = X_1 \cdot F(1, X_2, \ldots, X_n) + X_1' \cdot F(0, X_2, \ldots, X_n)$	(Shannon's expansion theorems)
(T15′)	$F(X_1, X_2, \ldots, X_n) = [X_1 + F(0, X_2, \ldots, X_n)] \cdot [X_1' + F(1, X_2, \ldots, X_n)]$	

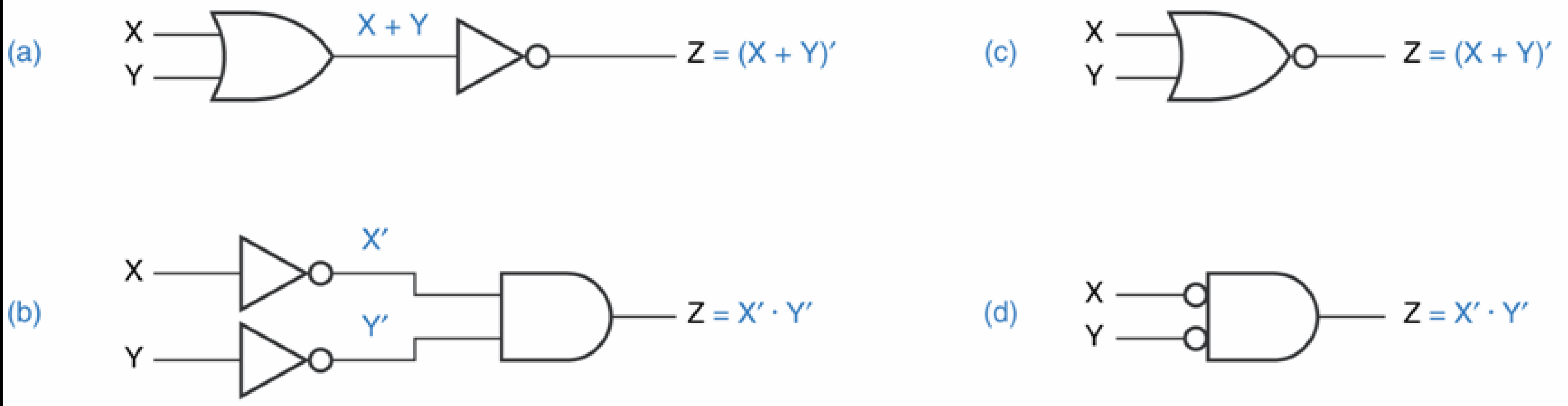
Switching-algebra theorems with n variables.

Generalized DeMorgan's theorem

- The complement of a logic expression F, which is denoted as (F)', is an expression whose value is the opposite of F's for every possible input combination.
- T14 states that given any *n*-variable logic expression, its complement can be obtained by swapping + and . and complementing all variables.
- Example:
  - F(W,X,Y,Z) = (W'.X) + (X.Y) + (W.(X'+Z'))
    = ((W)'.X) + (X.Y) + (W.((X)'+(Z)'))
  - [F(W,X,Y,Z)]' = (((W)')'+X') . (X'+Y') . (W'+(((X)')'.((Z)')'))
    = (W+X') . (X'+Y') . (W'+(X.Z))



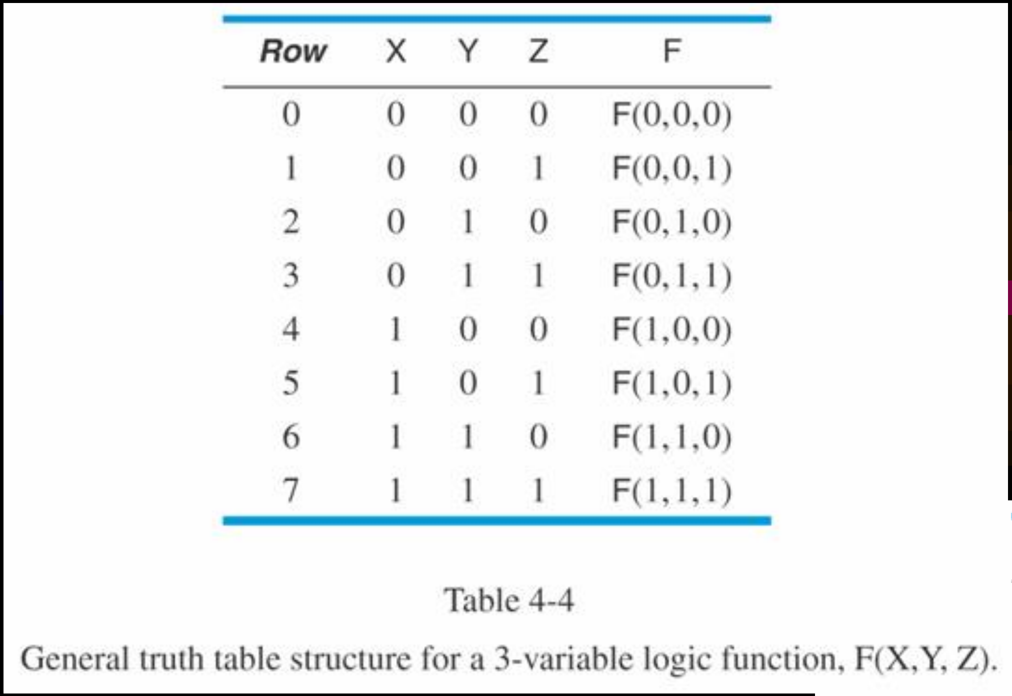
Equivalent circuits according to DeMorgan's theorem T13: (a) AND-NOT; (b) NOT-OR; (c) logic symbol for a NAND gate; (d) equivalent symbol for a NAND gate.



Equivalent circuits according to DeMorgan's theorem T13' :(a) OR-NOT; (b) NOT-AND; (c) logic symbol for a NOR gate; (d) equivalent symbol for a NOR gate.

Principle of duality

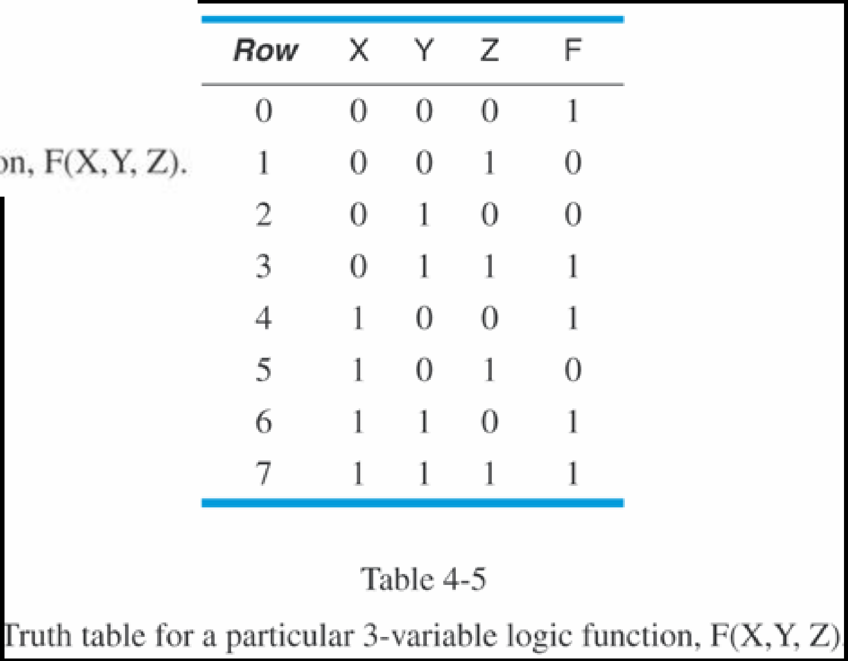
- Any theorem or identity in switching algebra remains true if 0 and 1 are swapped and . and + are swapped throughout.
- If $F(X_1, X_2,\ldots, X_n)$ is a fully parenthesized logic expression involving the variables $X_1, X_2,\ldots, X_n$ and the operators ., +, and ’, then the dual of F, denoted as F^D, is the same expression with + and . swapped.
- $F^D(X_1, X_2,\ldots, X_n, +, \cdot, ') = F(X_1, X_2,\ldots, X_n, \cdot, +, ')$
- Generalized DeMorgan’s theorem can be restated as:
  - $[F(X_1, X_2,\ldots, X_n)]' = F^D(X'_1, X'_2,\ldots, X'_n)$



Row	X	Y	Z	F
0	0	0	0	F(0,0,0)
1	0	0	1	F(0,0,1)
2	0	1	0	F(0,1,0)
3	0	1	1	F(0,1,1)
4	1	0	0	F(1,0,0)
5	1	0	1	F(1,0,1)
6	1	1	0	F(1,1,0)
7	1	1	1	F(1,1,1)

Table 4-4

General truth table structure for a 3-variable logic function, F(X,Y, Z).



Row	X	Y	Z	F
0	0	0	0	1
1	0	0	1	0
2	0	1	0	0
3	0	1	1	1
4	1	0	0	1
5	1	0	1	0
6	1	1	0	1
7	1	1	1	1

Table 4-5

Truth table for a particular 3-variable logic function, F(X,Y, Z).

$$F = \sum X,Y,Z(0,3,4,6,7)$$

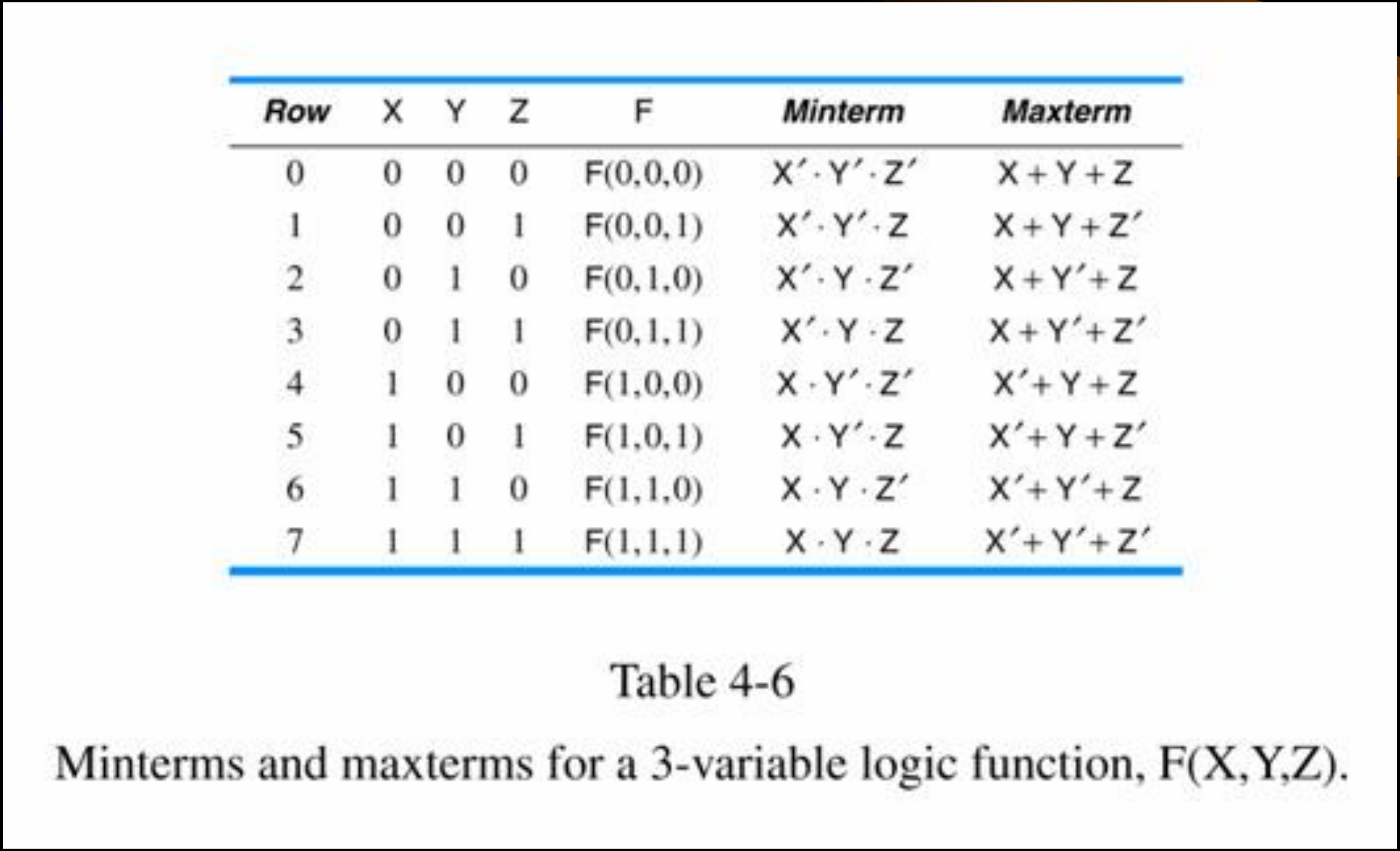
$$F = \Pi_{X,Y,Z}(1,2,5)$$

Definitions

- A *literal* is a variable or the complement of a variable.
  - Examples: X, Y, X’, Y’
- A *product term* is a single literal or a logical product of two or more literals.
  - Examples: Z’, W.X.Y, X.Y’.Z
- A *sum-of-products (SOP) expression* is a logical sum of product terms.
  - Example: Z’ + W.X.Y + X.Y’.Z
- A *sum term* is a single literal or a logical sum of two or more literals.
  - Examples: Z’, W+X+Y, X+Y’+Z
- A *product-of-sums (POS) expression* is a logical product of sum terms.
  - Example: Z’.(W+X+Y).(X+Y’+Z)

More definitions

- A *normal term* is a product or sum term in which no variable appears more than once.
  - A nonnormal term can always be simplified to a constant or a normal term using one of theorems T3, T3’, T5, or T5’.
  - Examples of nonnormal terms:
    W.X.X’.Y, W+W+X’+Y
  - Examples of normal terms: W.X.Y’, W+X’+Y
- An *n*-variable minterm is a normal product term with *n* literals. There are 2^n such product terms.
  - Some examples of 4-variable minterms:
    W’.X’.Y’.Z’, W.X.Y’.Z
  - A minterm can be defined as a product term that is 1 in exactly one row of the truth table.
- An *n*-variable maxterm is a normal sum term with *n* literals. There are 2^n such sum terms.
  - Some examples of 4-variable maxterms:
    W’+X’+Y’+Z’, W+X+Y’+Z
  - A maxterm can be defined as a sum term that is 0 in exactly one row of the truth table.



Row	X	Y	Z	F	*Minterm*	*Maxterm*
0	0	0	0	F(0,0,0)	$X' \cdot Y' \cdot Z'$	$X + Y + Z$
1	0	0	1	F(0,0,1)	$X' \cdot Y' \cdot Z$	$X + Y + Z'$
2	0	1	0	F(0,1,0)	$X' \cdot Y \cdot Z'$	$X + Y' + Z$
3	0	1	1	F(0,1,1)	$X' \cdot Y \cdot Z$	$X + Y' + Z'$
4	1	0	0	F(1,0,0)	$X \cdot Y' \cdot Z'$	$X' + Y + Z$
5	1	0	1	F(1,0,1)	$X \cdot Y' \cdot Z$	$X' + Y + Z'$
6	1	1	0	F(1,1,0)	$X \cdot Y \cdot Z'$	$X' + Y' + Z$
7	1	1	1	F(1,1,1)	$X \cdot Y \cdot Z$	$X' + Y' + Z'$

Table 4-6

Minterms and maxterms for a 3-variable logic function, F(X,Y,Z).

Integer representation

- A minterm can be represented by an *n*-bit integer, the *minterm number*.
- Minterm *i* denotes the minterm corresponding to row *i* of the truth table.
  - Row 5: 101 ; Minterm 5 = X.Y’.Z
- Opposite is true for maxterms
  - Maxterm 5 = X’+Y+Z’

Canonical sum and product

- The *canonical sum* of a logic function is a sum of the minterms corresponding to truth table rows (input combinations) for which the function produces a 1 output.
  - $F = \sum_{X,Y,Z}(0,3,4,6,7)$

    $= X'.Y'.Z' + X'.Y.Z + X.Y'.Z' + X.Y.Z' + X.Y.Z$
- The minterm list is also known as the *on-set* for the logic function. Each minterm "turns on" the output for exactly one input combination.
- Any logic function can be written as a canonical sum.

- The *canonical product* of a logic function is a product of the maxterms corresponding to truth table rows (input combinations) for which the function produces a 0 output.
  - $F = \Pi_{X,Y,Z}(1,2,5)$

    $= (X+Y+Z') . (X+Y'+Z) . (X'+Y+Z')$
- The maxterm list is also known as the *off-set* for the logic function. Each minterm "turns off" the output for exactly one input combination.
- Any logic function can be written as a canonical product.

Converting between minterm and maxterm lists

- For a function of *n* variables, the possible minterm and maxterm numbers are between 0 and 2^n-1.
- A minterm or maxterm list contains a subset of these numbers.
- To switch between list types, take the set compliment.
- Examples:

$$\sum_{X,Y,Z}(0,1,2,3) = \Pi_{X,Y,Z}(4,5,6,7)$$
$$\sum_{X,Y}(1) = \Pi_{X,Y}(0,2,3)$$
$$\sum_{W,X,Y,Z}(0,1,2,3,5,7,11,13)=\Pi_{W,X,Y,Z}(4,6,8,9,10,12,14,15)$$

Equivalent representations for a combinational logic function

- Truth table
- Algebraic sum of minterms, the canonical sum
- A minterm using ∑ notation
- Algebraic product of maxterms, the canonical product
- A maxterm using Π notation
- All of these representations provide exactly the same information as the others
- Given any one representation, the other four can be derived simply

Combinational circuit analysis

- In analyzing a logic circuit, you find a formal description of its logic function.
- Once you have that, you can represent it in all the different ways you have learned, and play with equivalent representations
- Can also feed it to a program that will generate the standard form circuit for you
  - This will not necessarily be the most efficient circuit that carries out the function
  - Programs are dumb

Exhaustive search

- Given an n-input circuit, construct the truth table. This involves finding the output generated for each of the 2^n possible combinations of input variables
- This will blow up for anything other than very small n
- Need a better way
- Already have it: algebraic representation
  - Use variable names rather than explicit values

Exhaustive search

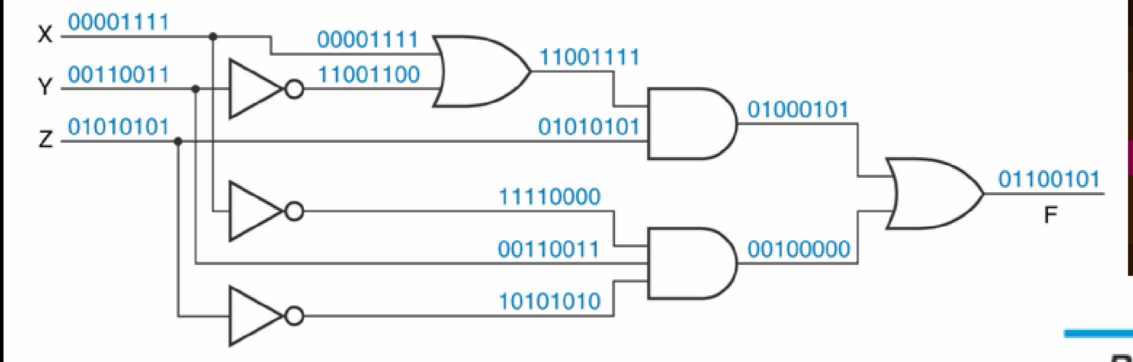


Figure 4-10
Gate outputs created by all input combinations.

Row	X	Y	Z	F
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

Table 4-7
Truth table for the logic circuit of Figure 4-9.

Algebraic representation

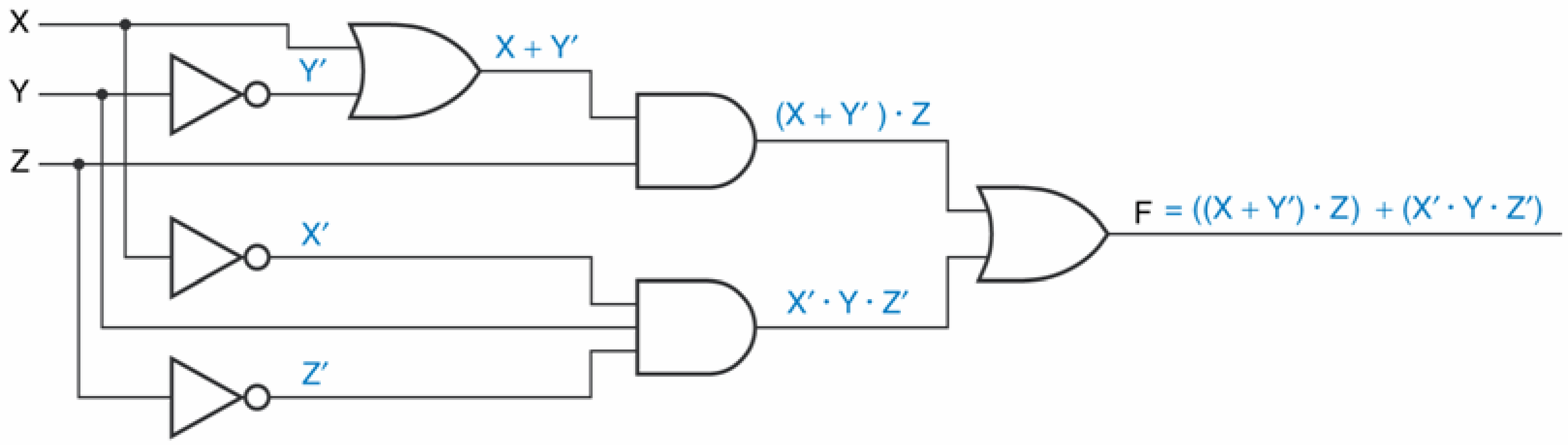


Figure 4-11
Logic expressions for signal lines

Equivalent circuit

- F = ((X+Y’) . Z) + (X’ . Y . Z’)
  = X . Z + Y’ . Z + X’ . Y . Z’

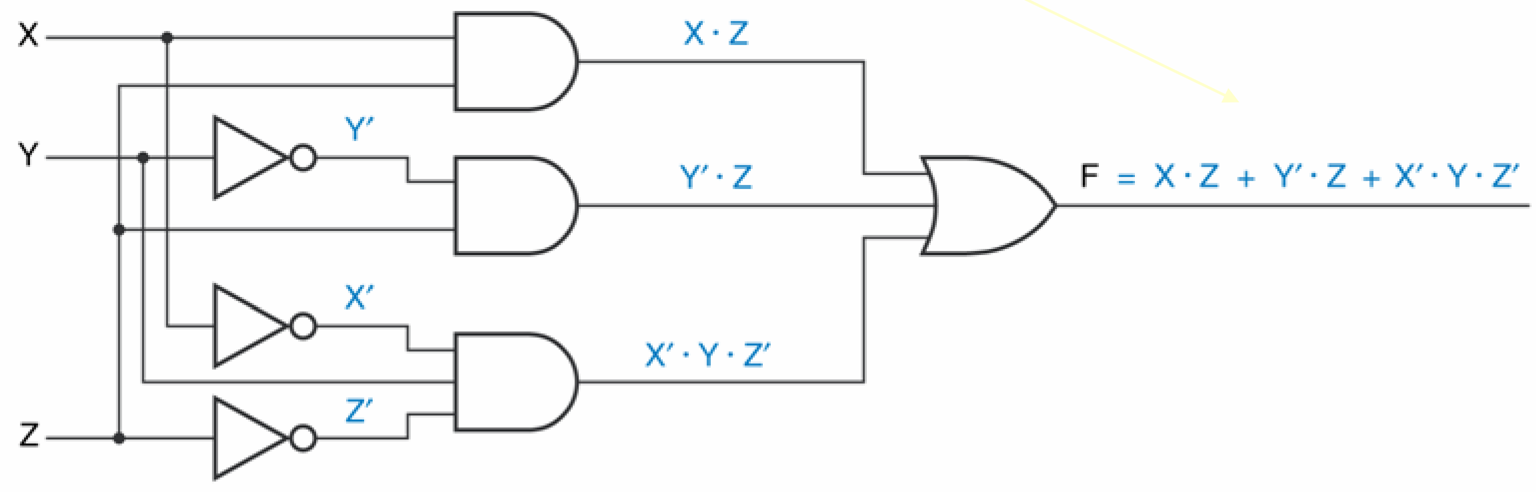


Figure 4-12
Two-level AND-OR circuit.

Another equivalent

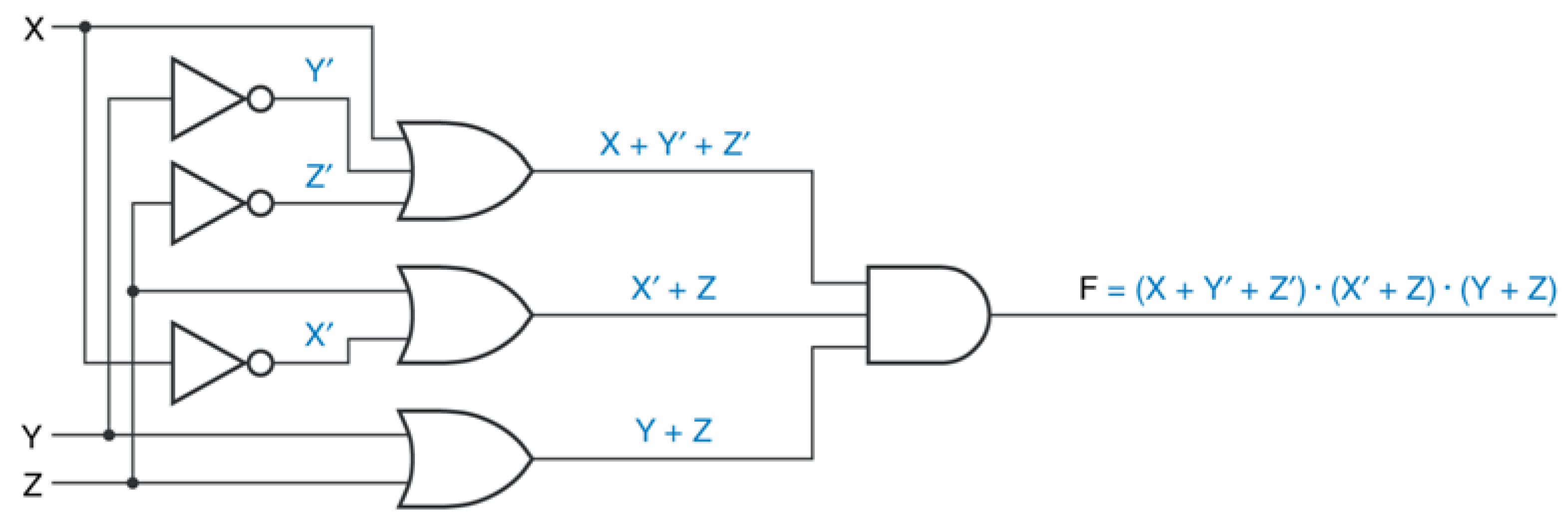


Figure 4-13

Two-level OR-AND circuit.

Circuit synthesis from "verbal" description

The ALARM output is 1 if the PANIC input is 1, or if the ENABLE input is 1, the EXITING input is 0, and the house is not secure; the house is secure if the WINDOW, DOOR, and GARAGE inputs are all 1.

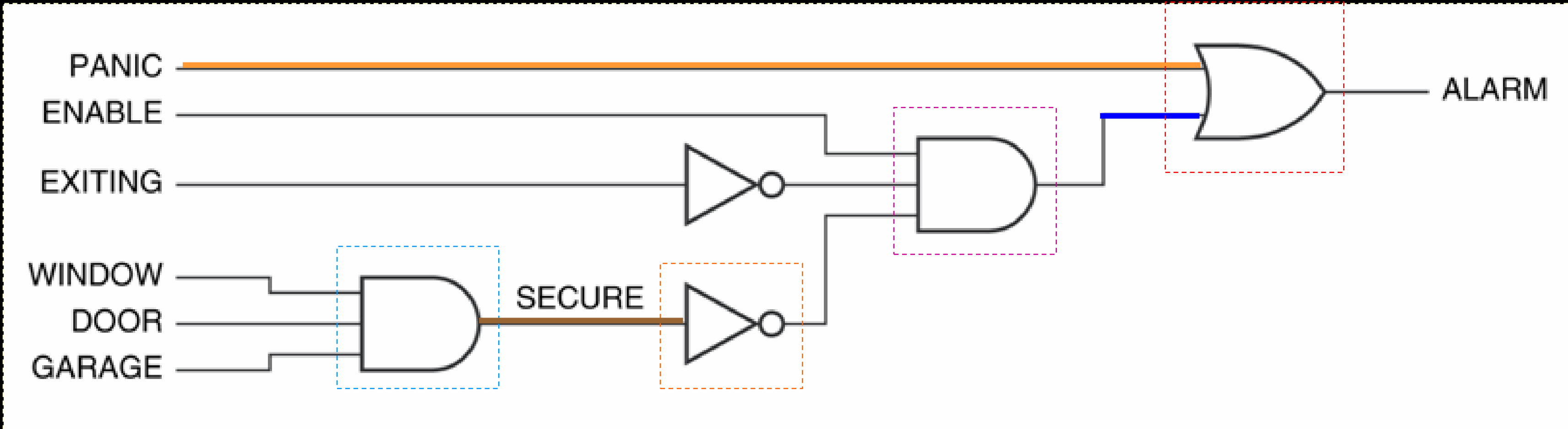


Figure 4-19

Alarm circuit derived from logic expression.

Sum-of-products equivalent

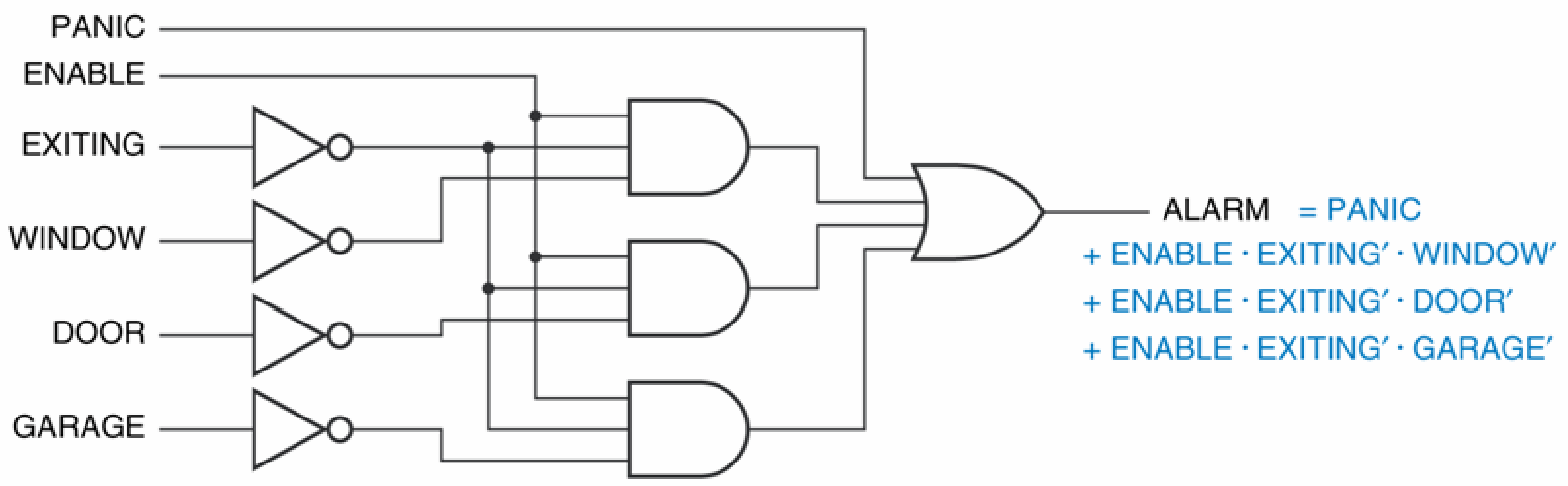

PANIC
ENABLE
EXITING
WINDOW
DOOR
GARAGE
ALARM
= PANIC
+ ENABLE · EXITING′ · WINDOW′
+ ENABLE · EXITING′ · DOOR′
+ ENABLE · EXITING′ · GARAGE′

Figure 4-20

Sum-of-products version of alarm circuit.

Need for minimization

- A logic expression can be synthesized into a circuit in many different forms.
- Some of these forms are “better” than others
  - Fewer gates
  - Smaller gates (fewer inputs)
- Canonical SOP and POS realizations blow up quickly. Too many gates!
- We need to “minimize” our combinational circuits.

Typical minimization technique

- Generalization of the combining theorems, T10 and T10’
  - given product term . Y + given product term . Y’ = given product term
  - (given sum term + Y) . (given sum term + Y’) = given sum term
  - Each expression has been converted to a single term with one less variable.
    - One gate has been eliminated, one fewer input

Prime number detector

- Given a 4-bit input combination $N = N_3N_2N_1N_0$, produce a 1 output for N=1, 2, 3, 5, 7, 11, 13, and 0 otherwise.
- $F = \sum_{N_3,N_2,N_1,N_0}(1, 3, 5, 7, 11, 13)$

  $= N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0$

  $+ N_3 \cdot N_2' \cdot N_1 \cdot N_0' + \ldots$

Minimization

- $F = \sum_{N_3,N_2,N_1,N_0}(1, 3, 5, 7, 11, 13)$

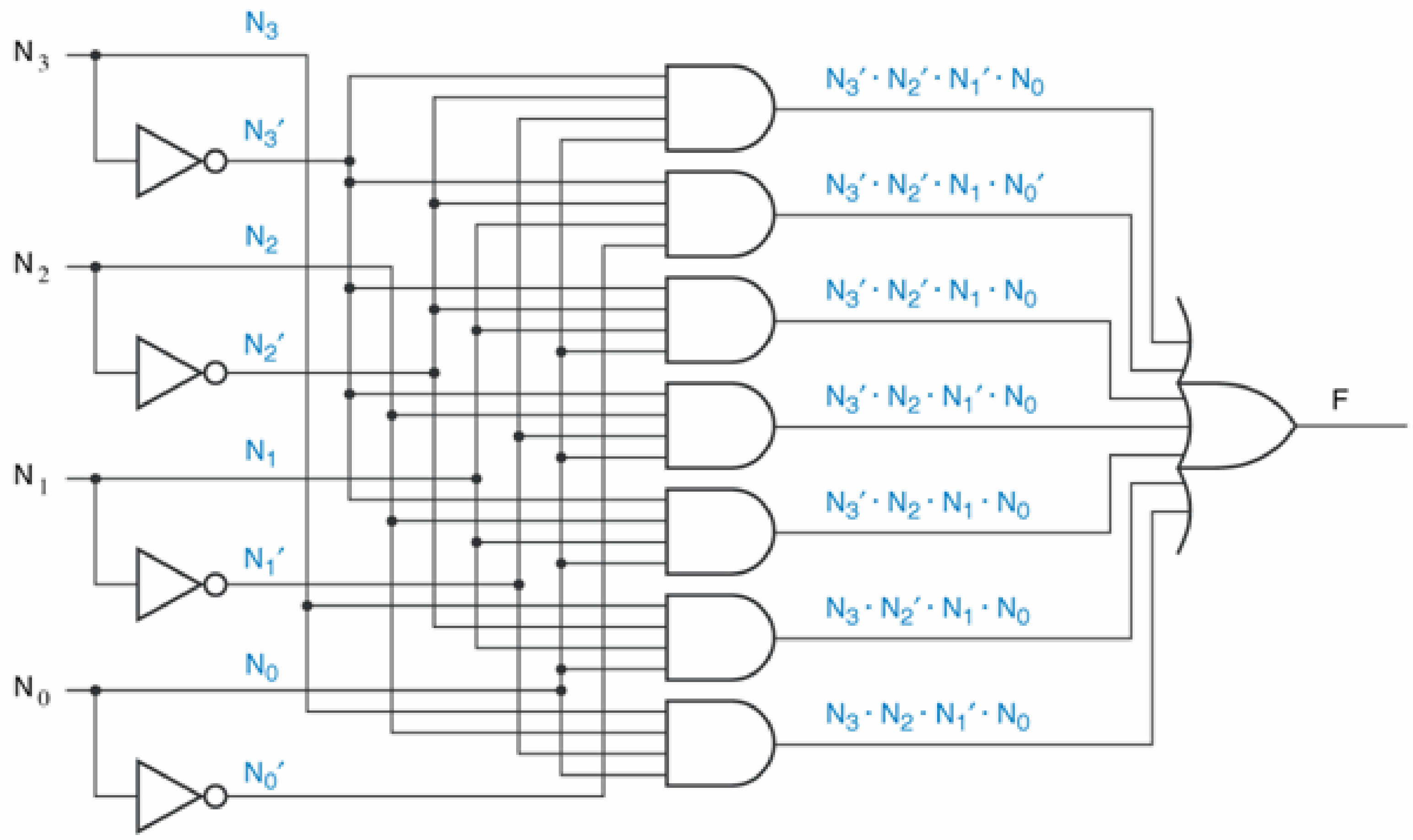
  $= (N_3' \,.\, N_2' \,.\, N_1' \,.\, N_0 + N_3' \,.\, N_2' \,.\, N_1 \,.\, N_0)$

  $+ (N_3' \,.\, N_2 \,.\, N_1' \,.\, N_0 + N_3' \,.\, N_2 \,.\, N_1 \,.\, N_0) + \ldots$

  $= N_3' \,.\, N_2' \,.\, N_0 + N_3' \,.\, N_2 \,.\, N_0 + \ldots$

  $= N_3' \,.\, N_0 + \ldots$

Canonical sum design



N_3
N_3
N_3'
N_2
N_2
N_2'
N_1
N_1
N_1'
N_0
N_0
N_0'
$N_3' \cdot N_2' \cdot N_1' \cdot N_0$
$N_3' \cdot N_2' \cdot N_1 \cdot N_0'$
$N_3' \cdot N_2' \cdot N_1 \cdot N_0$
$N_3' \cdot N_2 \cdot N_1' \cdot N_0$
$N_3' \cdot N_2 \cdot N_1 \cdot N_0$
$N_3 \cdot N_2' \cdot N_1 \cdot N_0$
$N_3 \cdot N_2 \cdot N_1' \cdot N_0$
F

Figure 4-18

Canonical-sum design for 4-bit prime-number detector.

Minimized circuit

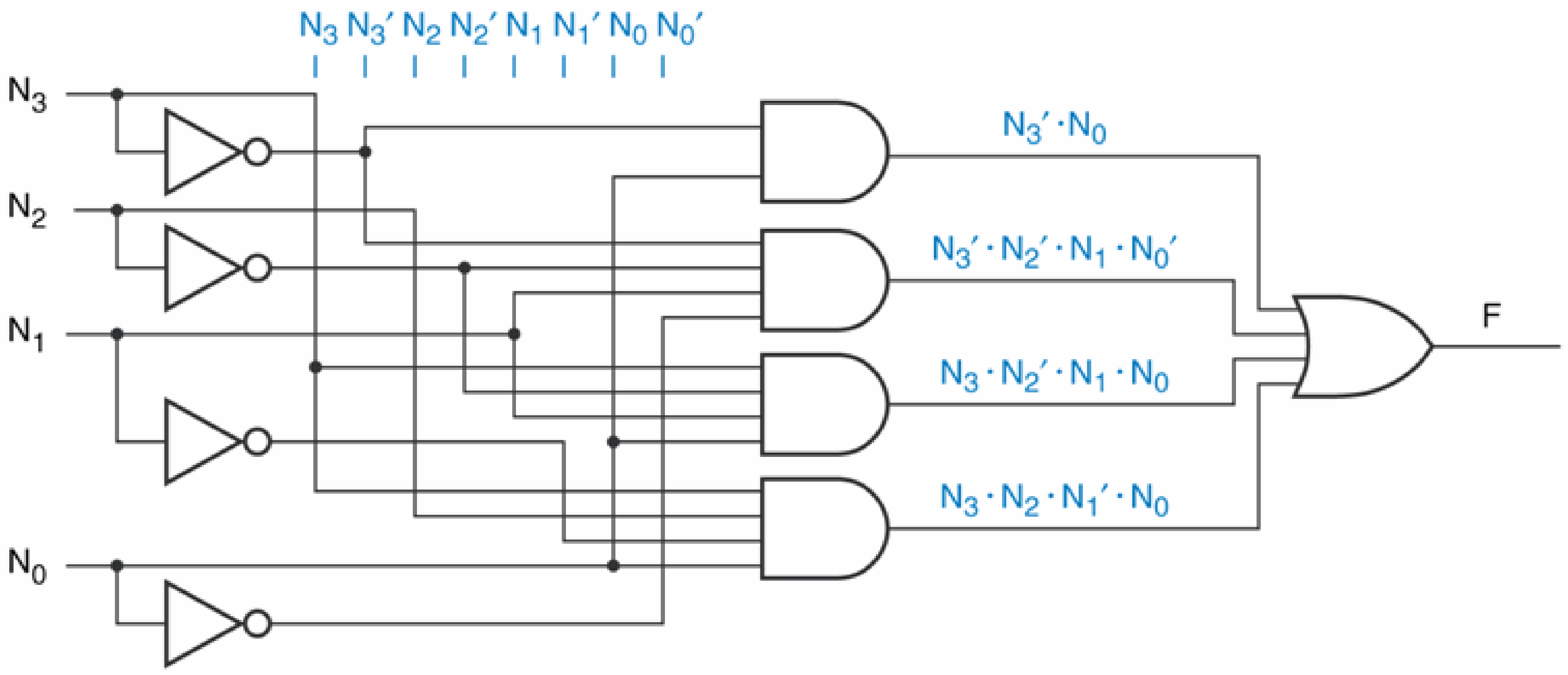

N_3 N_3' N_2 N_2' N_1 N_1' N_0 N_0'
N_3
N_2
N_1
N_0
$N_3' \cdot N_0$
$N_3' \cdot N_2' \cdot N_1 \cdot N_0'$
$N_3 \cdot N_2' \cdot N_1 \cdot N_0$
$N_3 \cdot N_2 \cdot N_1' \cdot N_0$
F

Figure 4-25

Simplified sum-of-products realization for 4-bit prime-number detector.

Karnaugh maps

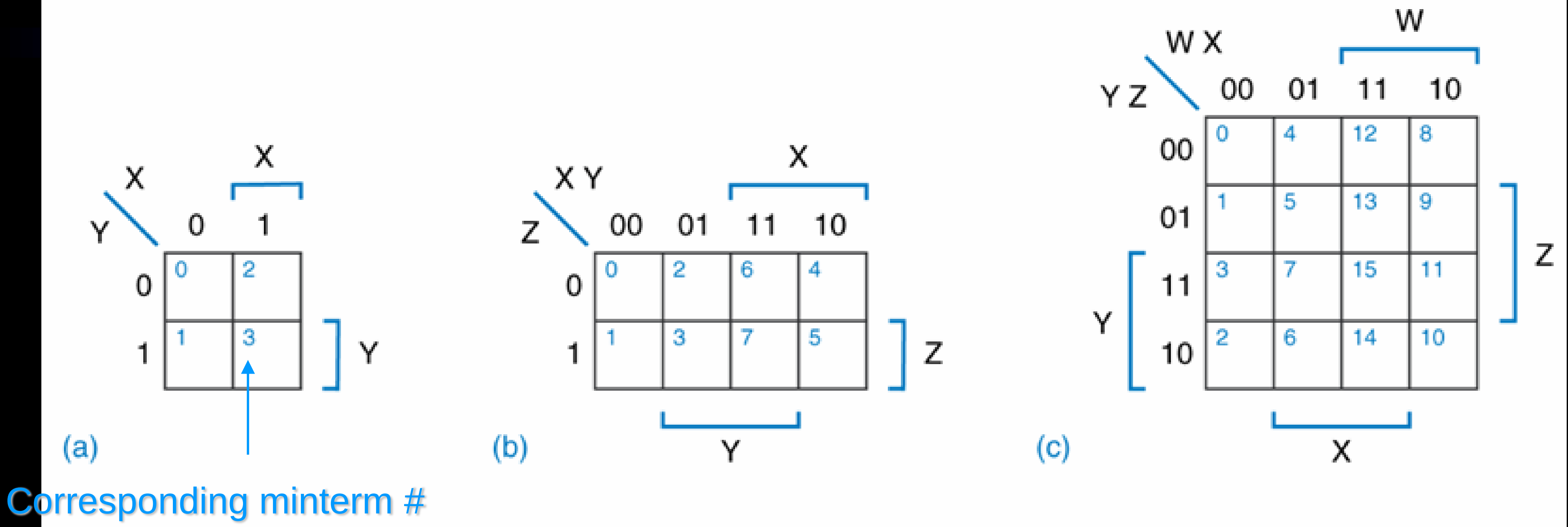


Figure 4-26

Karnaugh maps: (a) 2-variable; (b) 3-variable; (c) 4-variable.

Examples

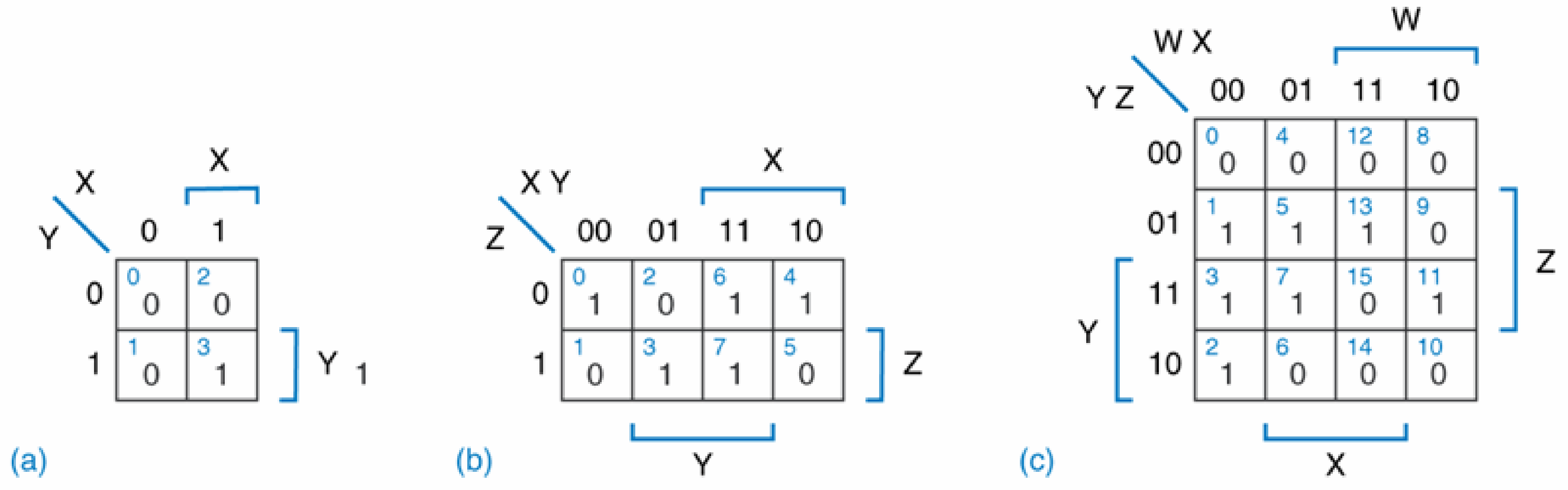
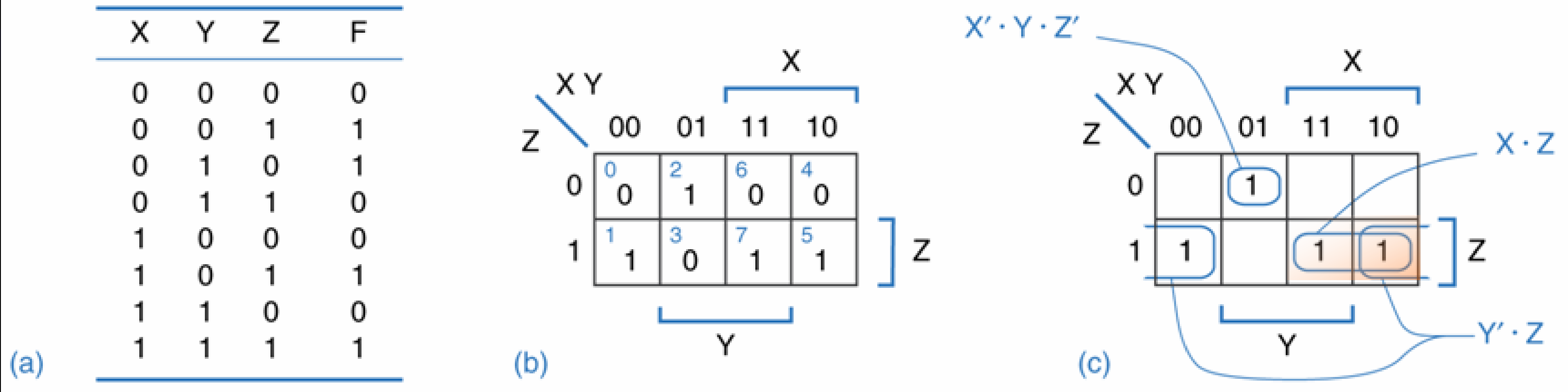


Figure 4-27

Karnaugh map for logic functions: (a) $F = \Sigma_{X,Y}(3)$;
(b) $F = \Sigma_{X,Y,Z}(0,3,4,6,7)$; (c) $F = \Sigma_{W,X,Y,Z}(1,2,3,5,7,11,13)$.

Minimization



X	Y	Z	F
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 4-28

F = $\Sigma_{X,Y,Z}(1,2,5,7)$: (a) truth table; (b) Karnaugh map; (c) combining adjacent 1-cells.

For cells 5 and 7:

F = … + X . Y' . Z + X . Y . Z

= … + X . Z

Logic expression from K-map

- Mark rectangular sets of 2^i (wraparound allowed) 1 cells
- Literals of corresponding product terms can be determined directly from the map:
  - If variable is 0 in every cell of marked area, then the variable is complemented in the product term
  - If variable is 1 in every cell of marked area, then the variable is uncomplemented in the product term
  - If variable is 0 in some cells of marked area and 1 in others, it does not appear in the product term
  - A sum-of-products expression for a function must contain product terms (marked sets of 1-cells) that cover all of the 1's and none of the 0's on the map.

Product-of-sums expression

- By duality, carry out the same process for the K-map, this time grouping the 0’s instead of the 1’s.
- Each 0 on the map corresponds to a maxterm in the canonical product of the logic function.
- Entire process can be repeated.