Chapter 2 Combinational Systems (I)

Chapter Outline (1/2)

- The Design Process for Combinational Systems
- Switching Algebra
- Implementation of Functions with AND, OR, and NOT Gates
- The Complement
- From the Truth Table to Algebraic Expressions
- NAND, NOR, and Exclusive-OR Gates
- Simplification of Algebraic Expressions
- Manipulation of Algebraic Functions and NAND Gate Implementations

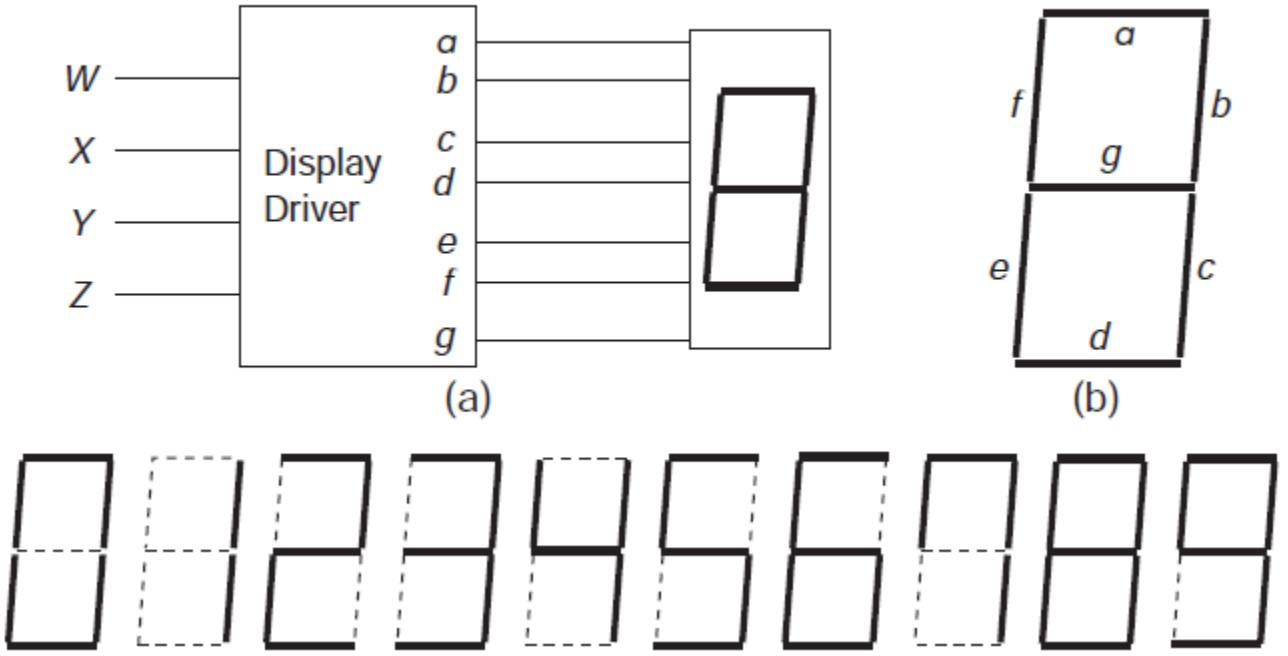
Chapter Outline (2/2)

- A More General Boolean Algebra
- Solved Problems

The Design Process for Combinational Systems (1/2)

- Consider designing a system that has as its input the code for a decimal digit and produces as its output the signals to drive a seven-segment display

Figure 2.3 A seven-segment display.



The Design Process for Combinational Systems (2/2)

- The design process involves the following steps (some may be omitted)
  1. Represent all the inputs and outputs in binary. If necessary, break the problem into smaller subproblems
  2. Formalize the design specification either by a truth table or by an algebraic expression
  3. Simplify the description
  4. Implement the system

Don't Care Conditions (1/1)

- In some systems, the value of the output is specified for only some of the input conditions. The remaining input combinations do not matter what the output is, that is, we don't care
- In a truth table, don't cares are indicated by an X, which can be either 1 or 0

Table 2.2 A truth table with a don't care.

a	*b*	*f*
0	0	0
0	1	1
1	0	1
1	1	X

The Development of Truth Table (1/4)

- Consider the design in Fig. 2.3. The display driver must provide the seven inputs to the display, a, b, c, d, e, f, and g.

1. The first thing that we must do is to select a code for the decimal digit. That will affect the truth table
2. The second thing is whether the display requires a 0 or a 1 on each segment input to light that segment
3. Finally, we must decide what to do about the inputs that do not correspond to a decimal digital (1010, 1011,…,1111)

The Development of Truth Table (2/4)

- We assume that the digits are stored in 8421 code. A 1 is to light a segment, the version of 6, 7, and 9 does not matter, and the inputs that do not represent a decimal digit never occur

Table 2.6 A truth table for the seven-segment display driver.

Digit	W	X	Y	Z	a	b	c	d	e	f	g
0	0	0	0	0	1	1	1	1	1	1	0
1	0	0	0	1	0	1	1	0	0	0	0
2	0	0	1	0	1	1	0	1	1	0	1
3	0	0	1	1	1	1	1	1	0	0	1
4	0	1	0	0	0	1	1	0	0	1	1
5	0	1	0	1	1	0	1	1	0	1	1
6	0	1	1	0	X	0	1	1	1	1	1
7	0	1	1	1	1	1	1	0	0	X	0
8	1	0	0	0	1	1	1	1	1	1	1
9	1	0	0	1	1	1	1	X	0	1	1
–	1	0	1	0	X	X	X	X	X	X	X
–	1	0	1	1	X	X	X	X	X	X	X
–	1	1	0	0	X	X	X	X	X	X	X
–	1	1	0	1	X	X	X	X	X	X	X
–	1	1	1	0	X	X	X	X	X	X	X
–	1	1	1	1	X	X	X	X	X	X	X

The Development of Truth Table (3/4)

- **Example 2.1.** We want to develop a truth table for a system with three inputs, a, b, and c, and four outputs, w, x, y, z. The output is a binary number equals to the largest integer that meets the input conditions:

a = 0: odd	a = 1: even
b = 0: prime	b = 1: not prime
c = 0: less than 8	c = 1: greater than or equal to 8

- Some inputs, *e.g.*, (0,1,0), (1,0,1), may never occur. For these inputs, the output is "don't care" denoted by (X,X,X,X)

The Development of Truth Table (4/4)

- **Example 2.1. (Cont'd)**

Input			Output			
a	*b*	*c*	*w*	*x*	*y*	*z*
0	0	0	0	1	1	1
0	0	1	1	1	0	1
0	1	0	X	X	X	X
0	1	1	1	1	1	1
1	0	0	0	0	1	0
1	0	1	X	X	X	X
1	1	0	0	1	1	0
1	1	1	1	1	1	0

Switching Algebra (1/2)

- Switching algebra is binary, that is, all variables and constants take on one of the two values: 0 and 1. Quantities that are not naturally binary must be coded into binary format
- Switching algebra is a special case of Boolean algebra, where each variable takes on one of k values ($k \geqq 2$)
- A number of properties of switching algebra:

1. OR (written as +); $a+b$ (read a OR b) is 1 iff $a=1$ or $b=1$ or both
2. AND (written as ・ or simply two variables catenated); $a \cdot b=ab$ is 1 iff $a=1$ and $b=1$
3. NOT (written as ’); a’ (read NOT a) is 1 iff $a=0$

Switching Algebra (2/2)

- The term *complement* is sometimes used instead of NOT. The operation is also referred to as inversion, and the device implementing it is called an inverter

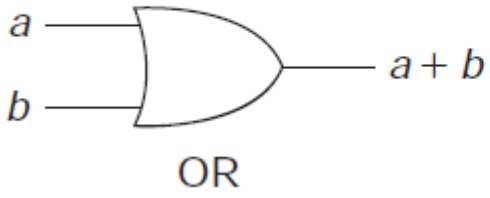
Table 2.7 Truth tables for OR, AND, and NOT.

a	b	$a + b$
0	0	0
0	1	1
1	0	1
1	1	1

a	b	ab
0	0	0
0	1	0
1	0	0
1	1	1

a	a'
0	1
1	0

Figure 2.4 Symbols for OR and AND gates.



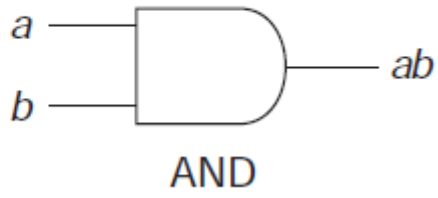
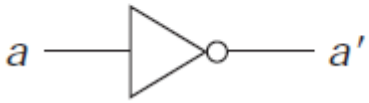


Figure 2.6 A NOT gate.



Basic Properties of Switching Algebra (1/5)

- Properties of switching algebra:

1. commutative

P1a. $a + b = b + a$ **P1b.** $ab = ba$

2. associative

P2a. $a + (b + c) = (a + b) + c$ **P2b.** $a(bc) = (ab)c$

- We can expand the definition of OR to

  $a+b+c+d+\cdots$ is 1 if any of the operands is 1 and is 0 only if all are 0's

- The definition of AND extends to

  $abcd\cdots$ is 1 if all of the operands are 1's and is 0 if any is 0

Basic Properties of Switching Algebra (2/5)

- Expressions inside the parentheses are evaluated first. When evaluating expressions without parentheses, the order of precedence is

1. NOT
2. AND
3. OR

- For example,

$ab'+c'd=[a(b')]+[(c')d]$

Basic Properties of Switching Algebra (3/5)

- Properties of switching algebra (Cont'd):

P3a.	$a + 0 = a$	**P3b.**	$a \cdot 1 = a$
P4a.	$a + 1 = 1$	**P4b.**	$a \cdot 0 = 0$
P5a.	$a + a' = 1$	**P5b.**	$a \cdot a' = 0$
P6a.	$a + a = a$	**P6b.**	$a \cdot a = a$

P7. $(a')' = a$

P8a. $a(b + c) = ab + ac$ **P8b.** $a + bc = (a + b)(a + c)$

- **Proof of P8b.** RHS$=aa+ac+ba+bc$

$=a+ac+ba+bc$

$=a(1+c)+ba+bc$

$=a+ba+bc$

$=(1+b)a+bc$

$=a+bc$

$=$LHS

Basic Properties of Switching Algebra (4/5)

- Another way to prove P8b is to produce a truth table for both sides of the equality and show that they are equal

Table 2.8 Truth table to prove Property 8b.

a	b	c	bc	LHS	$a + b$	$a + c$	RHS
0	0	0	0	0	0	0	0
0	0	1	0	0	0	1	0
0	1	0	0	0	1	0	0
0	1	1	1	1	1	1	1
1	0	0	0	1	1	1	1
1	0	1	0	1	1	1	1
1	1	0	0	1	1	1	1
1	1	1	1	1	1	1	1

Basic Properties of Switching Algebra (5/5)

- Properties of switching algebra (Cont'd):

P9a. $ab + ab' = a$ **P9b.** $(a + b)(a + b') = a$

P10a. $a + a'b = a + b$ **P10b.** $a(a' + b) = ab$

- The proof of P10a follows by using P8b

$$\begin{aligned} a + a'b &= (a + a')\,(a + b) \\ &= 1 \cdot (a + b) \\ &= a + b \end{aligned}$$

- P10b can be demonstrate by

$$a(a' + b) = aa' + ab = 0 + ab = ab$$

Manipulation of Algebraic Functions (1/3)

- A *literal* is the appearance of a variable or its complement. Examples are *a* and *b*. In determining the complexity of an expression, one of the measures is the number of literals. For example, the expression,

  $$ab'+bc'd+a'd+e'$$

  contains eight literals
- A product term is one or more literals connected by AND operators; *e.g.*, *ab'*, *bc'd*, *a'd*, and *e'*.
- A standard product term, also called minterm, is a product term that includes each variable of the problem

Manipulation of Algebraic Functions (2/3)

- A *sum of products* expression (abbreviated SOP) is one or more product terms connected by OR operators, *e.g.*,

$w'xyz' + wx'y'z' + wx'yz + wxyz$	(4 product terms)
$x + w'y + wxy'z$	(3 product terms)
$x' + y + z$	(3 product terms)
wy'	(1 product term)
z	(1 product term)

- A *canonical sum*, or sum of standard product terms, is just a sum of products expression where all of the terms are standard product terms

Manipulation of Algebraic Functions (3/3)

- Note that the first is a sum of standard product terms

(1)	$x'yz' + x'yz + xy'z' + xy'z + xyz$	5 terms, 15 literals
(2)	$x'y + xy' + xyz$	3 terms, 7 literals
(3)	$x'y + xy' + xz$	3 terms, 6 literals
(4)	$x'y + xy' + yz$	3 terms, 6 literals

- A *product of sums* expression (POS) is one or more sum terms connected by AND operators, *e.g.*,

$(w + x)(w + y)$	2 terms
$w(x + y)$	2 terms
w	1 term
$w + x$	1 term
$(w + x' + y' + z')(w' + x + y + z')$	2 terms

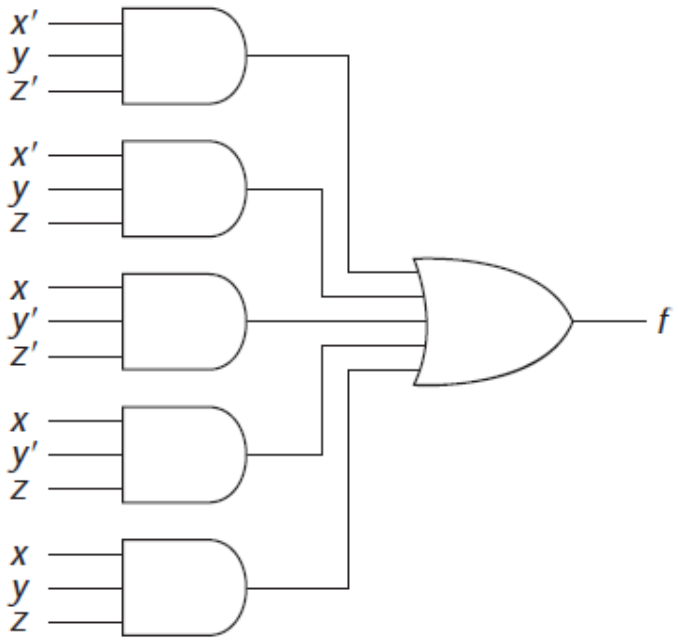
Implementation of Functions with AND, OR, and NOT Gates (1/9)

- Consider implementing the function

$$f=x'yz'+x'yz+xy'z'+xy'z+xyz.$$

A block diagram of a circuit to implement this is shown below.

Figure 2.7 Block diagram of f in sum of standard products form.

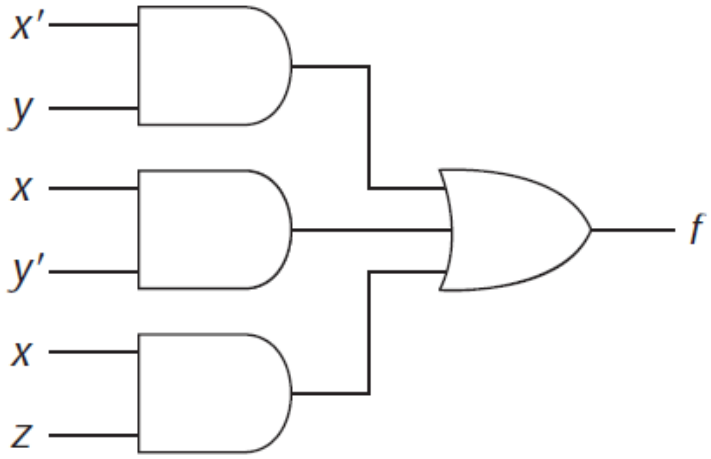


Implementation of Functions with AND, OR, and NOT Gates (2/9)

- This is an example of a *two-level* circuit. The number of levels is the maximum number of gates through which a signal must pass from the input to the output
- The same function can be manipulated to a minimum SOP expression, one version of which is

$$f = x'y + xy' + xz.$$

Figure 2.8 Minimum sum of product implementation of *f*.

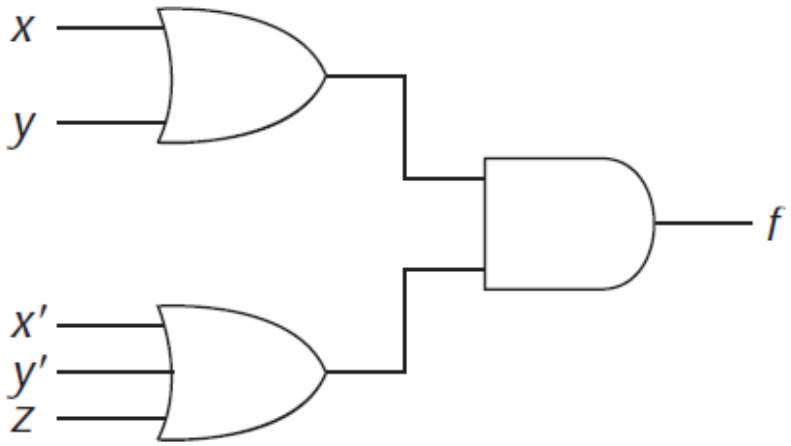


Implementation of Functions with AND, OR, and NOT Gates (3/9)

- The simplest definition of minimum for a gate network is minimum number of gates and, among those with the same number of gates, minimum number of gate inputs
- The minimum POS form of the same function is

$$f=(x+y)(x'+y'+z).$$

Figure 2.10 A product of sums implementation.



Implementation of Functions with AND, OR, and NOT Gates (4/9)

- When we implement functions that are in neither SOP nor POS form, the resulting circuits are more than two levels. For example,

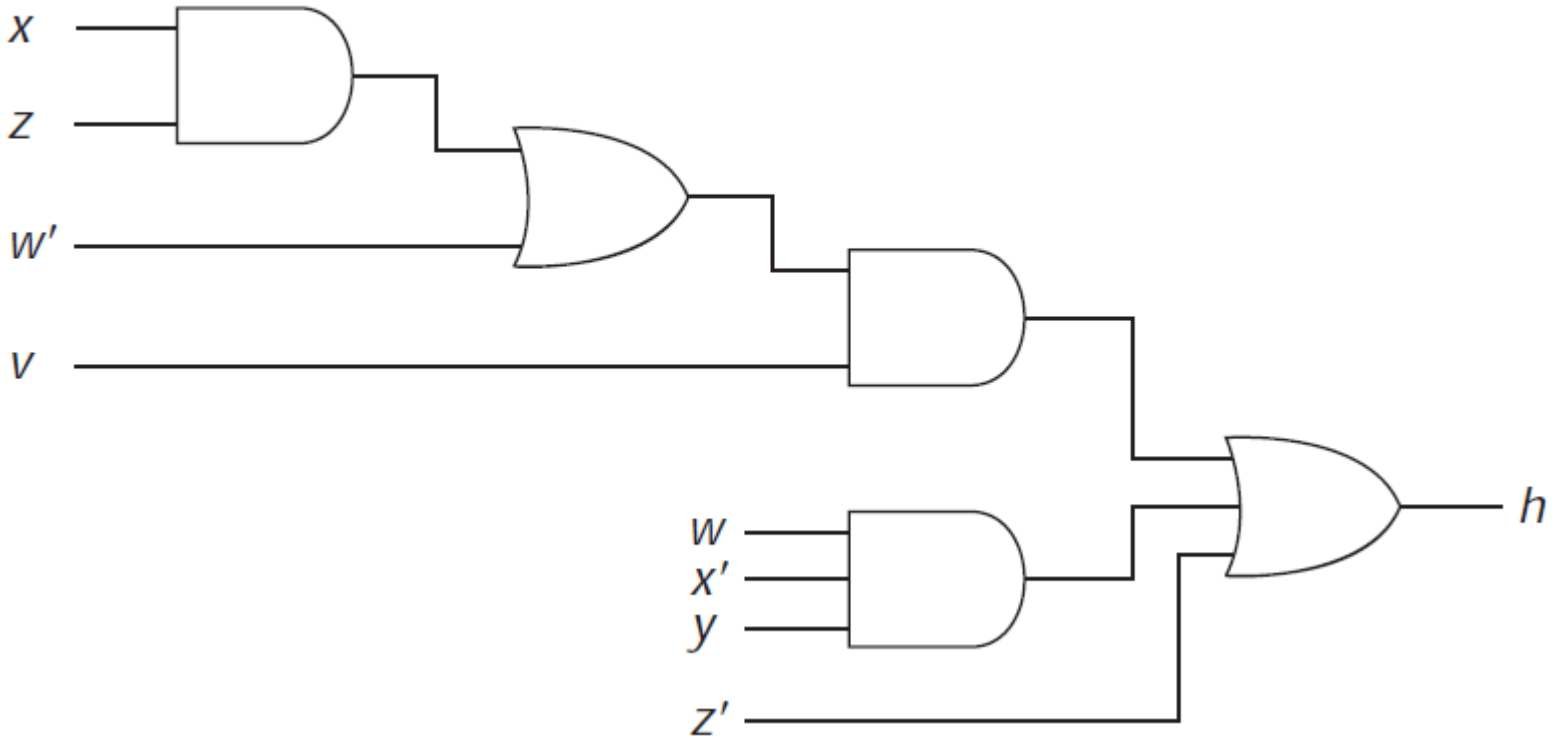
$$h=z'+wx'y+v(xz+w')$$

  is a four-level circuit because the signals x and z pass first through an AND gate, then an OR, then an AND, and finally through an OR.

- The gate network is shown in Fig. 2.11.

Implementation of Functions with AND, OR, and NOT Gates (5/9)

Figure 2.11 A multilevel circuit.



Implementation of Functions with AND, OR, and NOT Gates (6/9)

- Gates are typically available in dual in-line pin packages (DIPs) of 14 connector pins. These packages contain integrated circuits (ICs)
- Integrated circuits are categorized as *small-scale integration* (SSI) when they contain just a few gates. *Medium scale integration* (MSI) circuits contain as many as 100 gates. The terminology *large-scale integration* (LSI), *very large-scale integration* (VLSI), and *giga-scale integration* (GSI) are used for even more complex packages

Implementation of Functions with AND, OR, and NOT Gates (7/9)

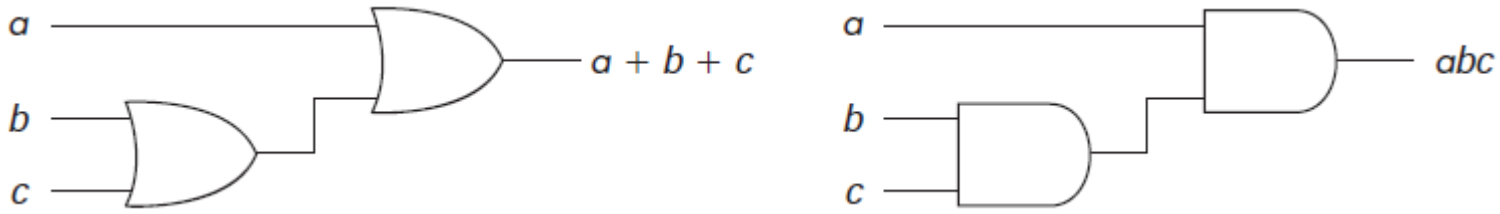
- A list of the common AND, OR, and NOT integrated circuits that might be encountered in the laboratory is

7404	6 (hex) NOT gates
7408	4 (quadruple) two-input AND gates
7411	3 (triple) three-input AND gates
7421	2 four-input (dual) AND gates
7432	4 (quadruple) two-input OR gates

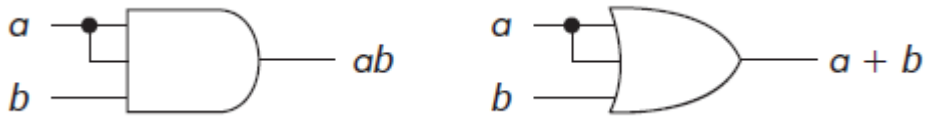
- The 7400 series of chips belong to transistor-transistor logic (TTL).

Implementation of Functions with AND, OR, and NOT Gates (8/9)

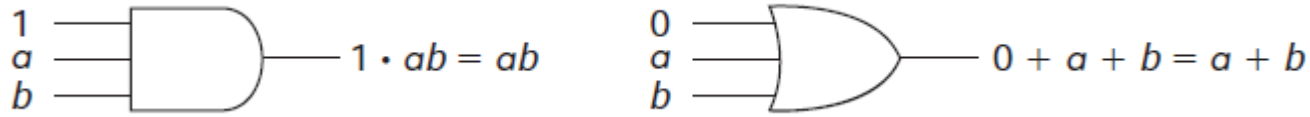
- We can build a three-input OR (or AND) gate using only two-input ones



- If we need a two-input gate, we can build it from a three-input one



- Similarly,



Implementation of Functions with AND, OR, and NOT Gates (9/9)

- Logic 0 and logic 1 are represented by two voltages. Most commonly, the higher voltage is used to represent 1 and the lower voltage to represent 0. This is referred to as *positive logic*. The opposite choice is also possible, which is referred to as *negative logic*

Table 2.9

a. High/Low

a	*b*	*f*
L	L	L
L	H	H
H	L	H
H	H	H

b. Positive logic

a	*b*	*f*
0	0	0
0	1	1
1	0	1
1	1	1

c. Negative logic

a	*b*	*f*
1	1	1
1	0	0
0	1	0
0	0	0

The Complement (1/2)

- We state the *DeMorgan's theorem*.

P11a. $(a + b)' = a'b'$ **P11b.** $(ab)' = a' + b'$

Table 2.10 Proof of DeMorgan's theorem.

a	b	$a + b$	$(a + b)'$	a'	b'	$a'b'$	ab	$(ab)'$	$a' + b'$
0	0	0	1	1	1	1	0	1	1
0	1	1	0	1	0	0	0	1	1
1	0	1	0	0	1	0	0	1	1
1	1	1	0	0	0	0	1	0	0
			11a			11a		11b	11b

COMMON MISTAKE: The NOT does not distribute through the parentheses. Thus,

$(ab)' \neq a'b'$ and $(a + b)' \neq a' + b'$

and, for example,

$ab + a'b' \neq 1$

The Complement (2/2)

- **Example 2.5.** Please use DeMorgan's theorem to find f'.

$$f = wx'y + xy' + wxz$$

then

$$\begin{aligned} f' &= (wx'y + xy' + wxz)' \\ &= (wx'y)'(xy')'(wxz)' \\ &= (w' + x + y')(x' + y)(w' + x' + z') \end{aligned}$$