Chapter 2 Combinational Systems (II)

From the Truth Table to Algebraic Expressions (1/9)

- Consider a two-variable truth table shown in Table 2.11
- What the table says is that

$$f = a'b + ab' + ab$$

Table 2.11 A two-variable truth table.

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

- *f* can also be represented by using *minterms*, which mean the terms that include all of the variables of the system inputs. Minterms are often referred to by number, by just converting the binary number in the row of the truth table to decimal

From the Truth Table to Algebraic Expressions (2/9)

- We can also represent f into the following two forms:

$$f(a, b) = m_1 + m_2 + m_3$$
$$f(a, b) = \Sigma m(1, 2, 3)$$

- **Example 2.7.** Represent f and f' in terms of A, B, and C.

ABC	f	f'
0 0 0	0	1
0 0 1	1	0
0 1 0	1	0
0 1 1	1	0
1 0 0	1	0
1 0 1	1	0
1 1 0	0	1
1 1 1	0	1

Table 2.12 Minterms.

ABC	Minterm	Number
0 0 0	$A'B'C'$	0
0 0 1	$A'B'C$	1
0 1 0	$A'BC'$	2
0 1 1	$A'BC$	3
1 0 0	$AB'C'$	4
1 0 1	$AB'C$	5
1 1 0	ABC'	6
1 1 1	ABC	7

From the Truth Table to Algebraic Expressions (3/9)

- **Example 2.7 (Cont'd)** For a specific function, those terms for which the function is 1 are used to form an SOP expression for f
- Those terms for which the function is 0 are used to form an SOP expression for f'. We can then complement f' to form a POS expression for f
- We have

$$f(A, B, C) = \Sigma m(1, 2, 3, 4, 5)$$
$$= A'B'C + A'BC' + A'BC + AB'C' + AB'C$$

and

$$f'(A, B, C) = \Sigma m(0, 6, 7)$$
$$= A'B'C' + ABC' + ABC$$

From the Truth Table to Algebraic Expressions (4/9)

- **Example 2.7 (Cont'd)** We can then complement *f*' to get a sum of maxterms

$$f = (f')' = (A + B + C)(A' + B' + C)(A' + B' + C')$$

- In most cases, including this one, the sum of minterms expression is not a minimum sum of products expression
- We could reduce *f* from 5 terms with 15 literals to either of two functions with 3 terms and 6 literal

$$\begin{aligned} f &= A'B'C + A'BC' + A'BC + AB'C' + AB'C \\ &= A'B'C + A'B + AB' \qquad \textbf{[P9a, P9a]} \\ &= A'C + A'B + AB' \\ &= B'C + A'B + AB' \end{aligned}$$

From the Truth Table to Algebraic Expressions (5/9)

- **Example 2.7 (Cont'd)** We can reduce *f'* from 3 terms with 9 literals to 2 terms with 5 literals. Using P9a:

$$f' = A'B'C' + AB$$

- Using P11, we can then obtain the minimum POS expression for *f*

$$f = (A + B + C)(A' + B')$$

From the Truth Table to Algebraic Expressions (6/9)

- **Example 2.8** $f(a, b, c) = \Sigma m(1, 2, 5) + \Sigma d(0, 3)$

  implies that minterms 1, 2, and 5 are included in the function and that 0 and 3 are don't cares. The truth table is as follows:

abc	*f*
0 0 0	X
0 0 1	1
0 1 0	1
0 1 1	X
1 0 0	0
1 0 1	1
1 1 0	0
1 1 1	0

From the Truth Table to Algebraic Expressions (7/9)

- **Question: How many different functions can be formed for *n* input variables?**
- **Answer:** 2^{2^n}. The details are as follows:
- For two variables, there are 16 possible truth tables, resulting in 16 different functions. The truth table of Table 2.13 shows all of these functions.

Table 2.13 All two-variable functions.

a	b	f_0	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}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

From the Truth Table to Algebraic Expressions (8/9)

- **Question: (Cont'd)** The set of functions, written in minimum SOP forms, are

$f_0 = 0$ $\quad f_6 = a'b + ab'$ $\quad f_{12} = a'$

$f_1 = ab$ $\quad f_7 = a + b$ $\quad f_{13} = a' + b$

$f_2 = ab'$ $\quad f_8 = a'b'$ $\quad f_{14} = a' + b'$

$f_3 = a$ $\quad f_9 = a'b' + ab$ $\quad f_{15} = 1$

$f_4 = a'b$ $\quad f_{10} = b'$

$f_5 = b$ $\quad f_{11} = a + b'$

- For n input variables, the truth table has 2^n rows, and thus, we can choose any 2^n-bit number for a column. Thus, there are 2^{2^n} different functions of n input variables

From the Truth Table to Algebraic Expressions (9/9)

- **Question: (Cont'd)** The number of functions can be huge if *n* is large.

Table 2.14 Number of functions of *n* variables.

Variables	Terms
1	4
2	16
3	256
4	65,536
5	4,294,967,296

NAND, NOR, and Exclusive-OR Gates (1/10)

- We will introduce three other commonly used types of gates, the NAND, the NOR, and the Exclusive-OR gates
- The NAND gate (NOT-AND) is commercially available in several sizes, typically two-, three-, four-, and eight-input varieties

Figure 2.12 NAND gates.

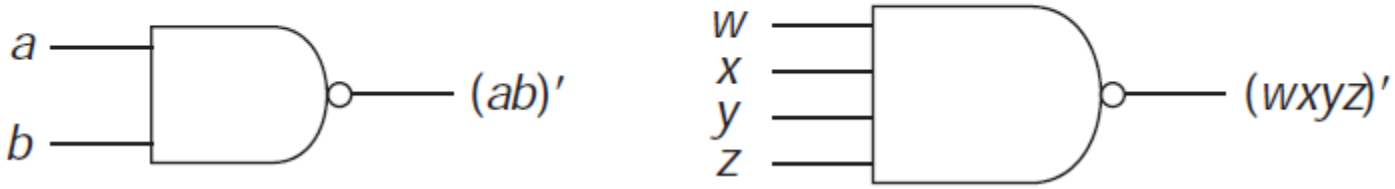
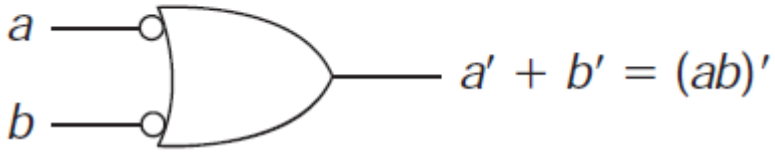


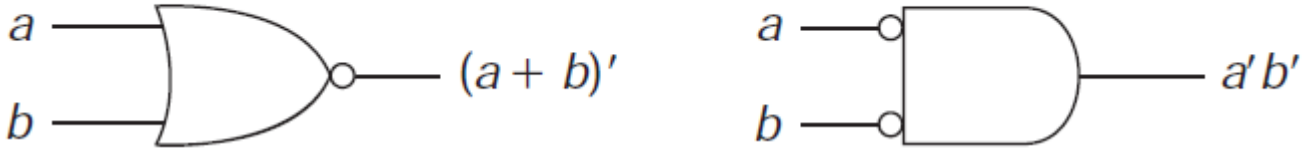
Figure 2.13 Alternative symbol for NAND.



NAND, NOR, and Exclusive-OR Gates (2/10)

- The NOR gate (NOT-OR) uses the symbols shown in Fig. 2.14

Figure 2.14 Symbols for NOR gate.



- We could implement gates with more than two inputs using NANDs. We could also implement AND, OR, and NOT gates using only NORs. These operators are said to be *functionally complete*

NAND, NOR, and Exclusive-OR Gates (3/10)

Figure 2.15 Functional completeness of NAND.

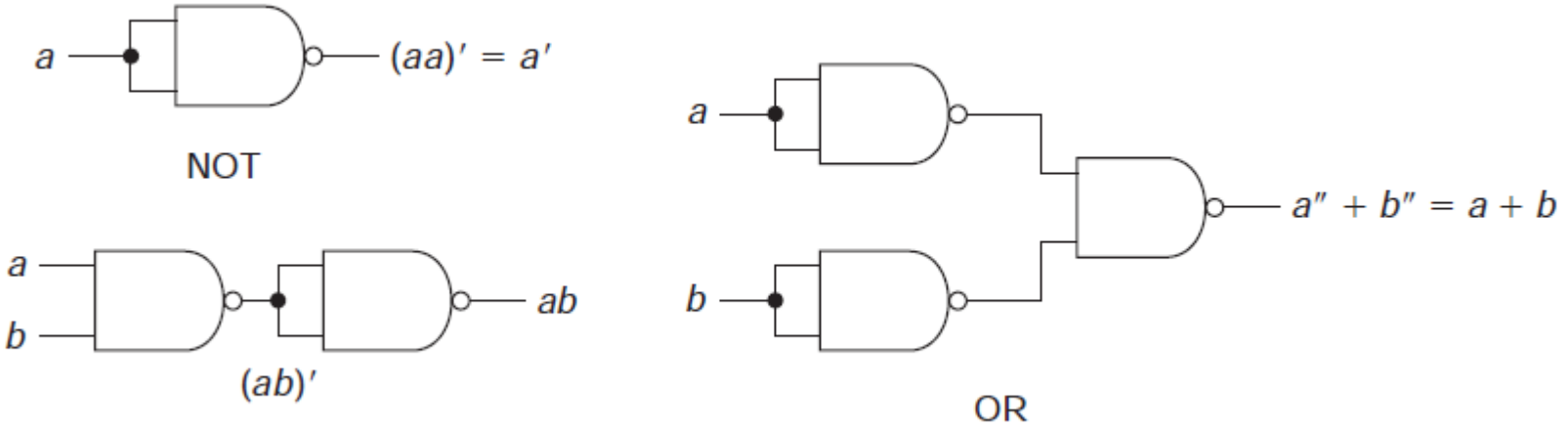
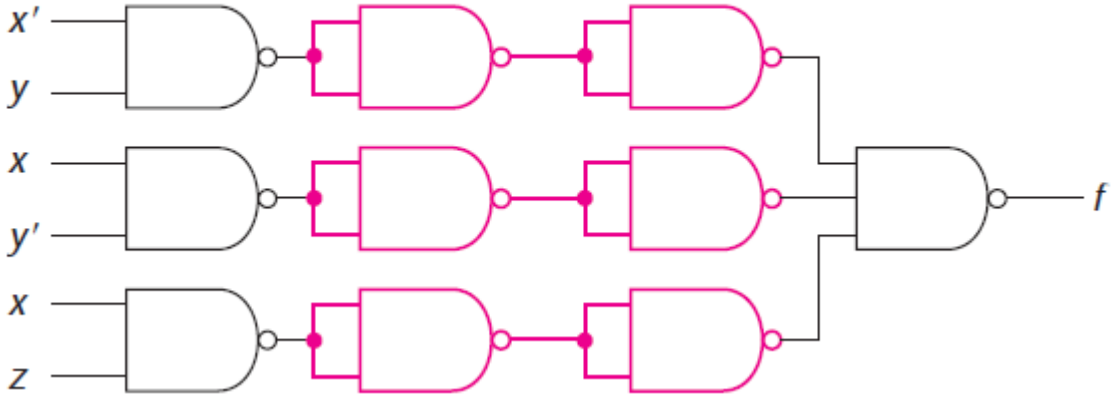


Figure 2.16 NAND gate implementation.

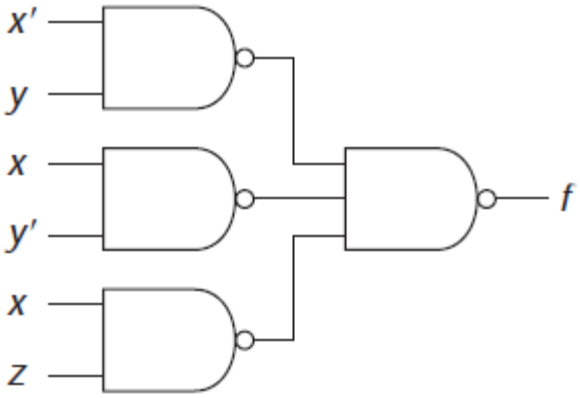


- We have $f=x'y+xy'+xz$

NAND, NOR, and Exclusive-OR Gates (4/10)

- A better NAND gate implementation of Fig. 2.16 is shown in Fig. 2.17

Figure 2.17 Better NAND gate implementation.



- All of the AND and OR gates of the original circuit can be replaced by NANDs. The output function is unchanged.

NAND, NOR, and Exclusive-OR Gates (5/10)

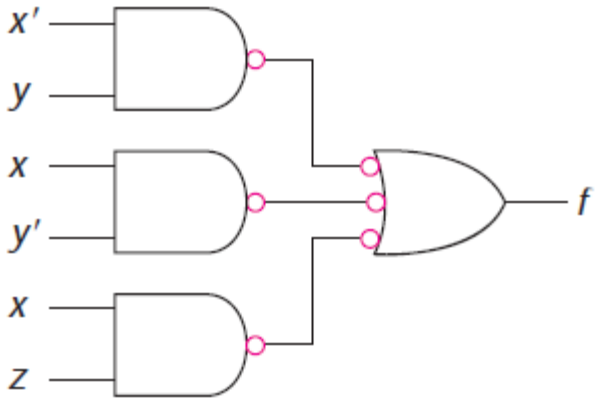
- The process to transform a circuit consisting of AND and OR gates to all NANDs can be simplified such that

1. The output of the circuit comes from an OR
2. The inputs to all OR gates come either from the system input or from the output of an AND
3. The inputs to all AND gates come either from the system input or from the output of an OR

NAND, NOR, and Exclusive-OR Gates (6/10)

- Fig. 2.18 shows the double NOT gate approach to transform *f* into an all NAND system.

Figure 2.18 Double NOT gate approach.



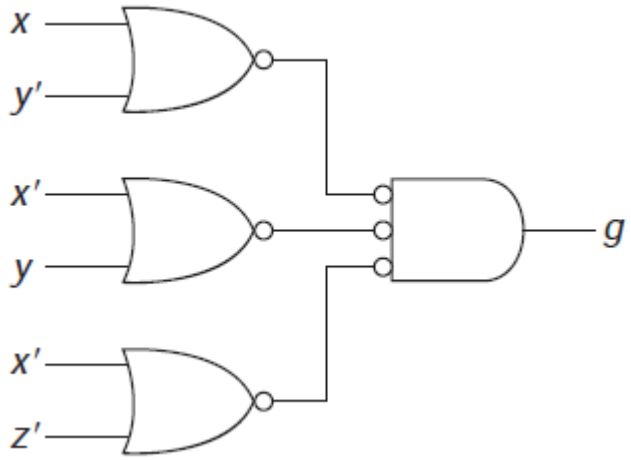
NAND, NOR, and Exclusive-OR Gates (7/10)

- The dual approach to transform a circuit consisting of AND and OR gates to all NOR gates can be simplified such that

1. The output of the circuit comes from an AND
2. The inputs to all OR gates come either from the system input or from the output of an AND
3. The inputs to all AND gates come either from the system input or from the output of an OR

NAND, NOR, and Exclusive-OR Gates (8/10)

- **Example 2.13.** $g=(x+y')(x'+y)(x'+z')$ is implemented as follows where all gates are NOR gates.



NAND, NOR, and Exclusive-OR Gates (9/10)

- The Exclusive-OR gate implements the expression

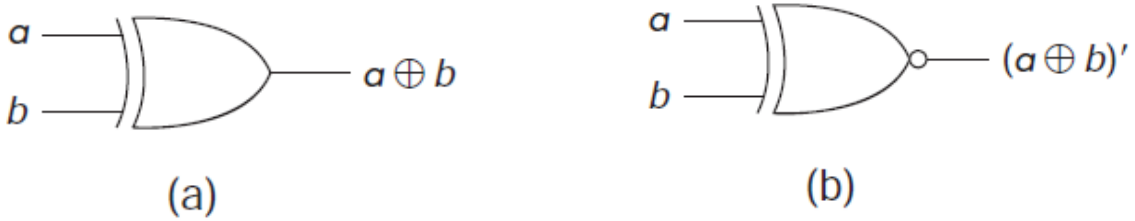
  $a'b+ab'$

  which is written as $a \oplus b$, is 1 if a=1 (and b=0) or if b=1 (and a=0), but not both a=1 and b=1

- Exclusive-NOR gate is just an Exclusive-OR with a NOT on the output and produces the function

$$(a \oplus b)' = a'b' + ab.$$

Figure 2.20 (a) An Exclusive-OR gate. (b) An Exclusive-NOR gate.



NAND, NOR, and Exclusive-OR Gates (10/10)

- Exclusive-NOR is referred to as a comparator, since the Exclusive-NOR is 1 if $a=b$, and is 0 if $a \neq b$
- A list of some common NAND, NOR, and Exclusive-OR integrated circuit packages that we may encounter in the laboratory is as follows:

7400	4 (quadruple) two-input NAND gates
7410	3 (triple) three-input NAND gates
7420	2 (dual) four-input NAND gates
7430	1 eight-input NAND gate
7402	4 (quadruple) two-input NOR gates
7427	3 (triple) three-input NOR gates
7486	4 (quadruple) two-input Exclusive-OR gates

Simplification of Algebraic Expressions (1/6)

- Switching algebra property 12:

  P12a. $a + ab = a$ **P12b.** $a(a + b) = a$

- **Proof of P12a:**

  $a+ab=a(1+b)=a \cdot 1=a$

- **Proof of P12b:**

  $a(a+b)=a \cdot a+ab=a+ab=a$

Simplification of Algebraic Expressions (2/6)

- **Example 2.15.** Simplify $xyz+x'y+x'y'$.

$$xyz + x'y + x'y'$$
$$= xyz + x'$$
$$= x' + yz$$

where $a = x'$, $a' = x$, and $b = yz$

Simplification of Algebraic Expressions (3/6)

- The operator consensus (indicated by the symbol ¢) is defined as follows:
- For any two product terms where exactly one variable appears uncomplemented in one and complemented in the other, the consensus is defined as the product of the remaining literals. If no such variable exists or if more than one such variable exists, then the consensus is undefined. If we write one term as at_1 and the second as $a't_2$ (where t_1 and t_2 represent product terms), then, if the consensus is defined,

$$at_1 \,¢\, a't_2 = t_1 t_2$$

Simplification of Algebraic Expressions (4/6)

- **Example 2.19.**

  $ab'c \not\subset a'd = b'cd$

  $ab'c \not\subset a'cd = b'cd$

  $abc' \not\subset bcd' = abd'$

  $b'c'd' \not\subset b'cd' = b'd'$

  $abc' \not\subset bc'd =$ undefined—no such variable

  $a'bd \not\subset ab'cd =$ undefined—two variables, a and b

- We then have the following property that is useful in reducing functions.

  P13a. $at_1 + a't_2 + t_1t_2 = at_1 + a't_2$

  P13b. $(a + t_1)(a' + t_2)(t_1 + t_2) = (a + t_1)(a' + t_2)$

- P13a states that the consensus term is redundant and can be removed from an SOP expression. P13b is in POS expression of P13a

Simplification of Algebraic Expressions (5/6)

- Proof of P13a by algebra

$$\begin{aligned} at_1 + a't_2 &= (at_1 + at_1t_2) + (a't_2 + a't_1t_2) \\ &= at_1 + a't_2 + (at_1t_2 + a't_1t_2) \\ &= at_1 + a't_2 + t_1t_2 \end{aligned}$$

- Proof of P13a by truth table

Table 2.15 Consensus.

a	t_1	t_2	at_1	$a't_2$	RHS	t_1t_2	LHS
0	0	0	0	0	0	0	0
0	0	1	0	1	1	0	1
0	1	0	0	0	0	0	0
0	1	1	0	1	1	1	1
1	0	0	0	0	0	0	0
1	0	1	0	0	0	0	0
1	1	0	1	0	1	0	1
1	1	1	1	0	1	1	1

Simplification of Algebraic Expressions (6/6)

- **Example 2.21.** Simplifiy $g=bc'+abd+acd$
- The only consensus term defined is

$$bc' \not\subset acd = abd$$

Property 13 now allows us to remove the consensus term. Thus,

$$g = bc' + acd$$

Manipulation of Algebraic Functions and NAND Gate Implementations (1/7)

- If we have an SOP expression and need to expand it to sum of minterms, we have two operations.

1. First, we can create a truth table and produce a sum of minterms. This approach will work for an expression in any format.
2. The other approach is to add variables to a term until all terms become minterms

- **Example 2.26.**

$$g = x' + xyz = x'y + x'y' + xyz$$
$$= x'yz + x'yz' + x'y'z + x'y'z' + xyz$$

$$g(x, y, z) = \Sigma m(3, 2, 1, 0, 7) = \Sigma m(0, 1, 2, 3, 7)$$

Note minterm numbers are usually written in numeric order

Manipulation of Algebraic Functions and NAND Gate Implementations (2/7)

- One other property

  P14a. $ab + a'c = (a + c)(a' + b)$

- Proof of P14a.

  RHS=$aa'+ab+a'c+bc$

  $=ab+a'c+bc$

  $=ab+a'c$=LHS

Manipulation of Algebraic Functions and NAND Gate Implementations (3/7)

- **Example 2.34.** Consider the design of a 1-bit full adder. The two binary input are represented as a and b, while the carry input is represented as c. From the truth table, we know the outputs s and c_{out} are

$$s = a'b'c + a'bc' + ab'c' + abc$$
$$c_{out} = bc + ac + ab$$

- Although s and c_{out} are in minimum SOP form, we can manipulate the algebra to reduce the gate requirements by reorganizing s and c_{out} as

$$s = c(a'b' + ab) + c'(ab' + a'b)$$
$$c_{out} = c(a + b) + ab$$

Manipulation of Algebraic Functions and NAND Gate Implementations (4/7)

- **Example 2.34. (Cont'd)** Returning to the expression for sum and carry

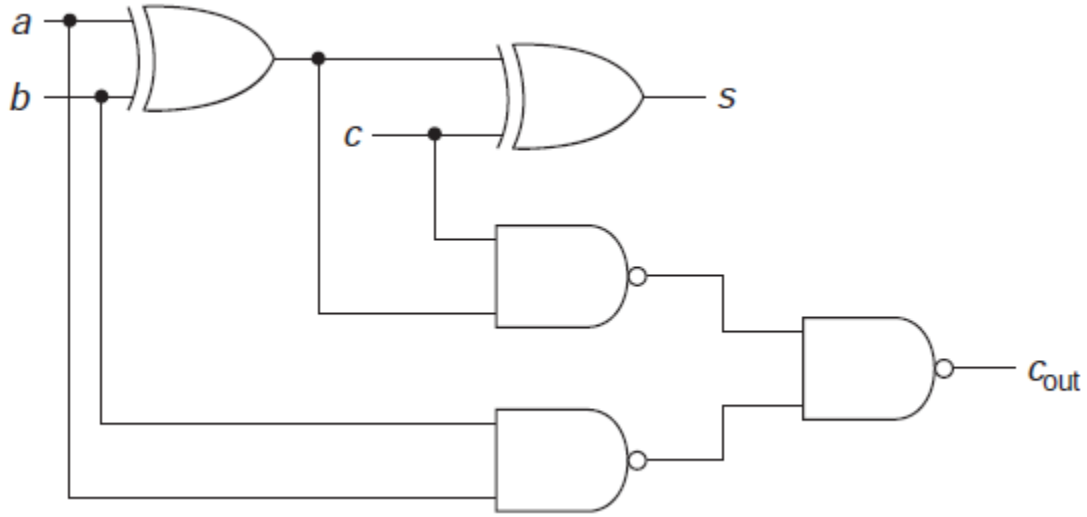
$$s = c(a \oplus b)' + c'(a \oplus b) = c \oplus (a \oplus b)$$

$$c_{out} = c(a \oplus b) + ab$$

- Note we use a little trick that is not obvious from any of the properties. The difference between $a+b$ and $a \oplus b$ is that the former is 1 when both a and b are 1, but the latter is not. But the expression for c_{out} is 1 for $a=b=1$ because of the ab term.

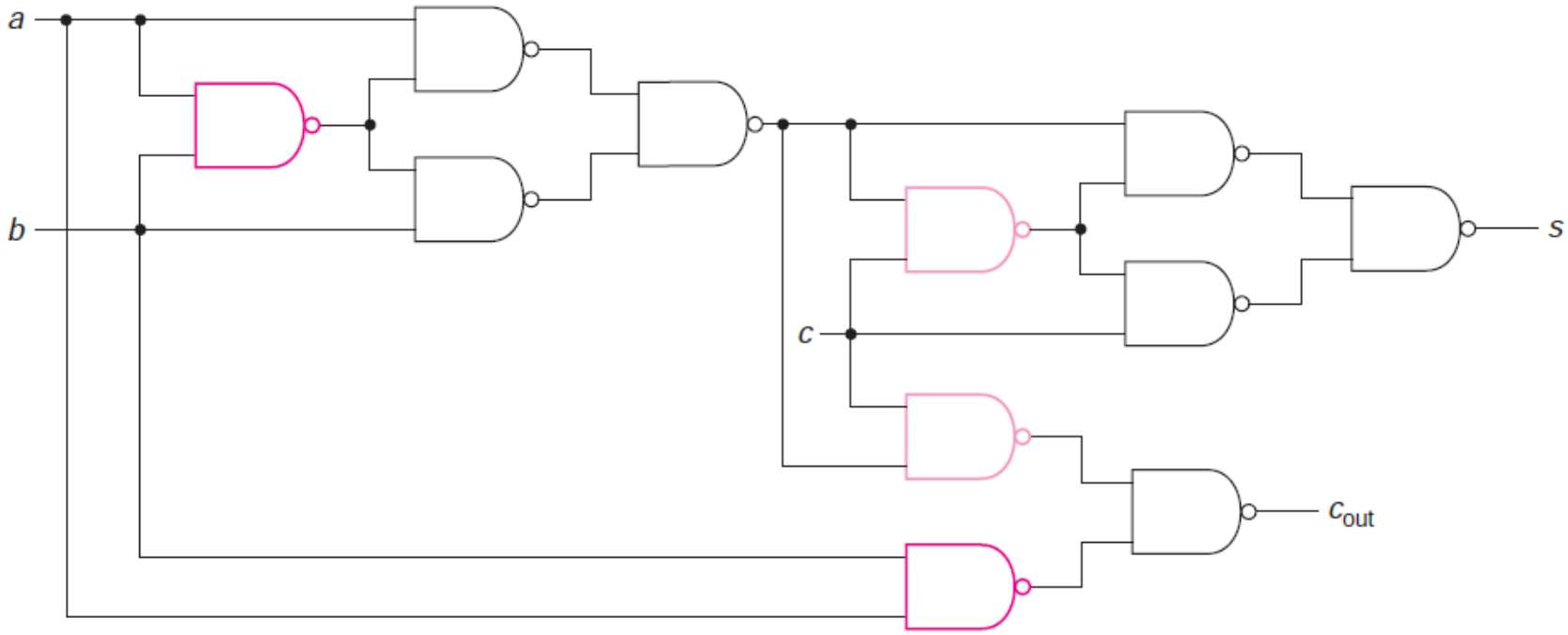
Manipulation of Algebraic Functions and NAND Gate Implementations (5/7)

- **Example 2.34. (Cont'd)** We have



Manipulation of Algebraic Functions and NAND Gate Implementations (6/7)

- **Example 2.34. (Cont'd)** The XOR gate can be replaced by a set of four NAND gates.



Manipulation of Algebraic Functions and NAND Gate Implementations (7/7)

- **Example 2.34. (Cont'd)** Note the four pink NAND gates can be further simplified.