Combinational Logic Design Principles. Combinational Circuits Analysis and Synthesis

Doru Todinca

Department of Computers
Politehnica University of Timisoara

Outline

Outline

Combinational circuit analysis

- By analysis we obtain a description of the logic function, starting from the circuit diagram
- What can we do with this description ?
  - we can determine the behaviour of the circuit for different input combinations
  - we can manipulate the algebraic description in order to obtain different circuit structures for the same logic function: e.g., to transform an AND-OR (sum-of-products) description in a NAND-NAND description, or an OR-AND description in a NOR-NOR description
  - we can transform the description in a form suitable for implementation in an available technology: sum-of-products for PLA implementation, truth table for a lookup memory used in FPGAs

Example of a circuit

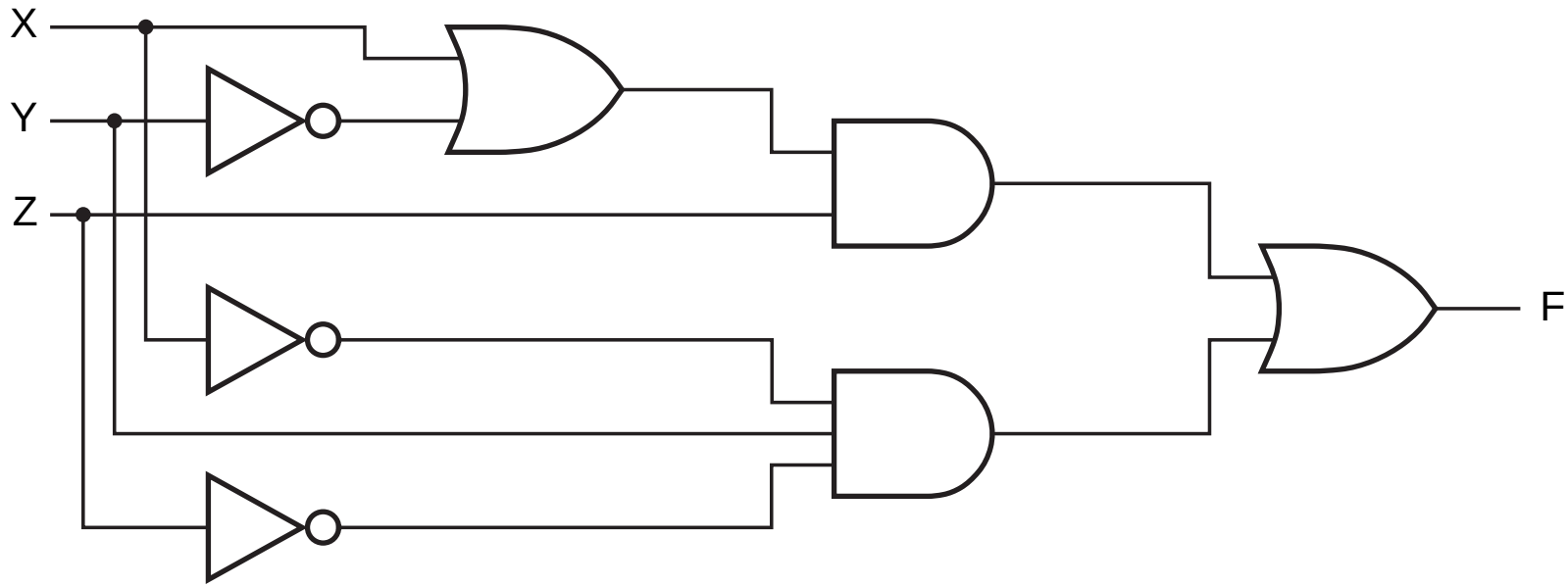

Figure 1: A 3-input, 1-output logic circuit

An "exhaustive" analysis

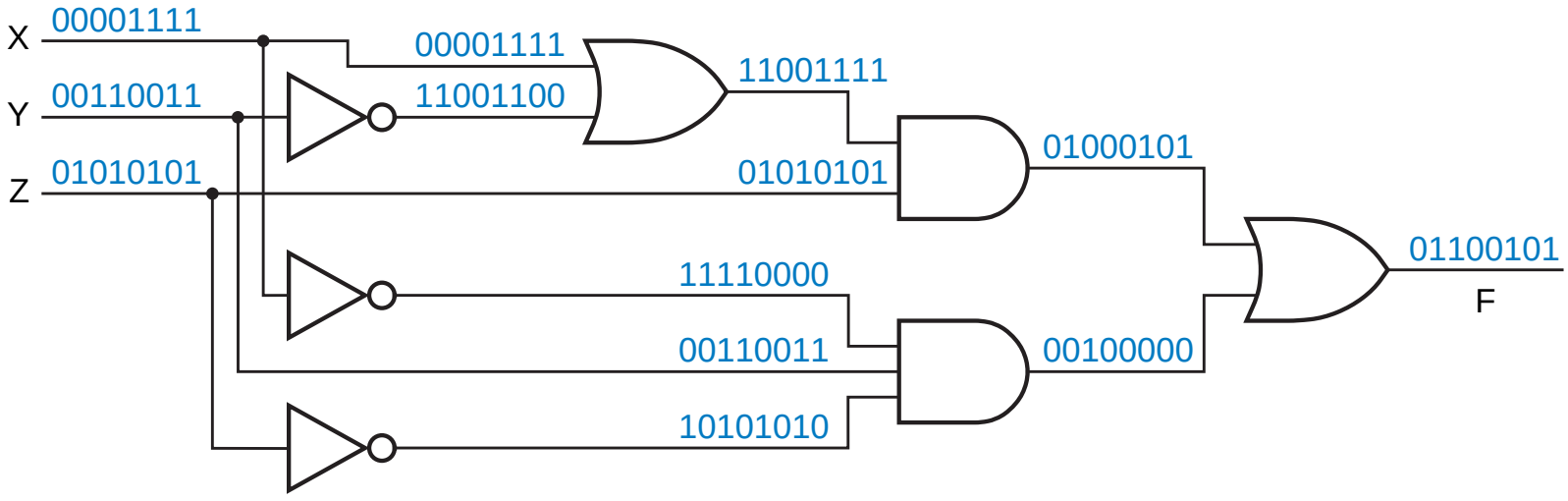

Figure 2: Gate outputs for all input combinations

- We apply all possible input combinations (from 000 to 111), and compute the values at each gate's output until we arrive at the output of the circuit
- At the circuit output we have the truth table output (figure 3)

The truth table for the circuit

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.

Figure 3: Truth table for the circuit from figure 1

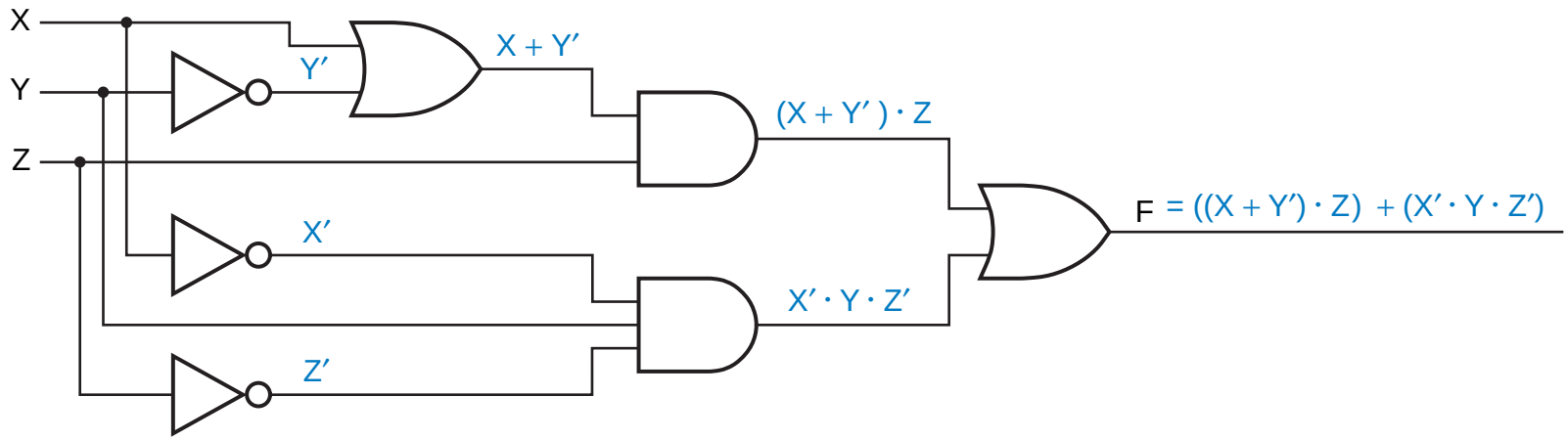

Figure 4: Logic expressions for signal lines

- The complexity of the "exhaustive" method grows exponentially with the number of inputs (there are 2^n input combinations for n inputs)
- An easier way is to use an algebraic approach: to build up a parenthesized logic expression for the circuit
- We start with inputs and compute the logic expression at each gate's output till we reach the circuit's output
- We can simplify the obtained logic expressions as we go, or at the end, after we obtain the output expression

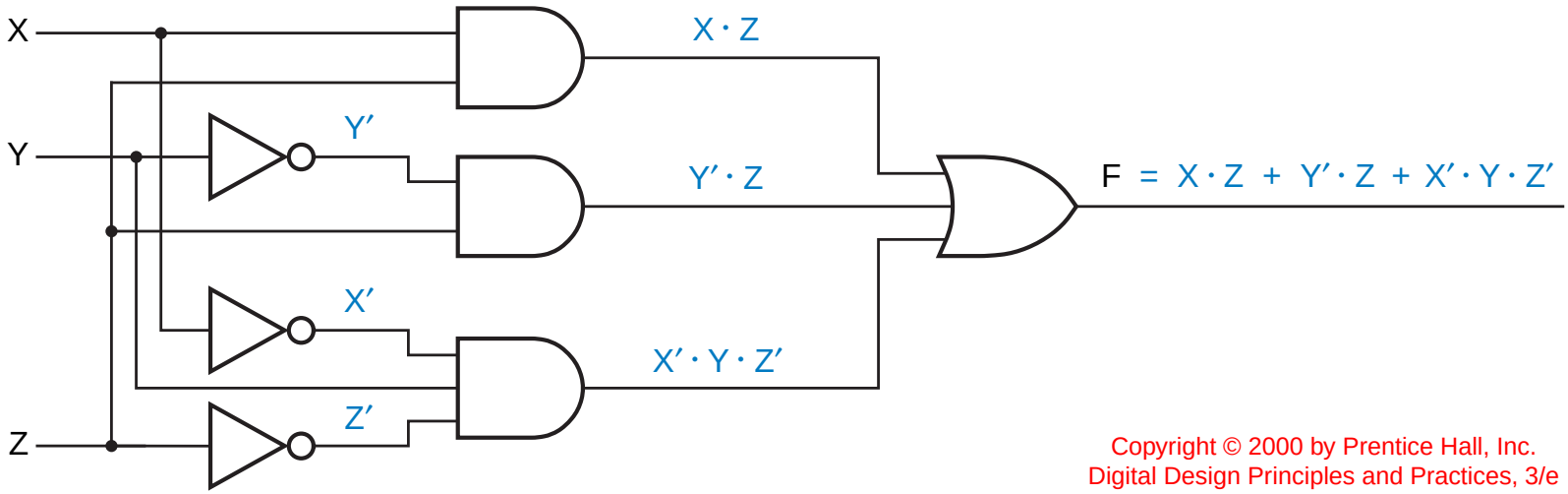

Figure 5: Two-level AND-OR circuit

If we multiply out the expression of F from figure 4, we obtain a sum-of-products expression, corresponding to the circuit from figure 5

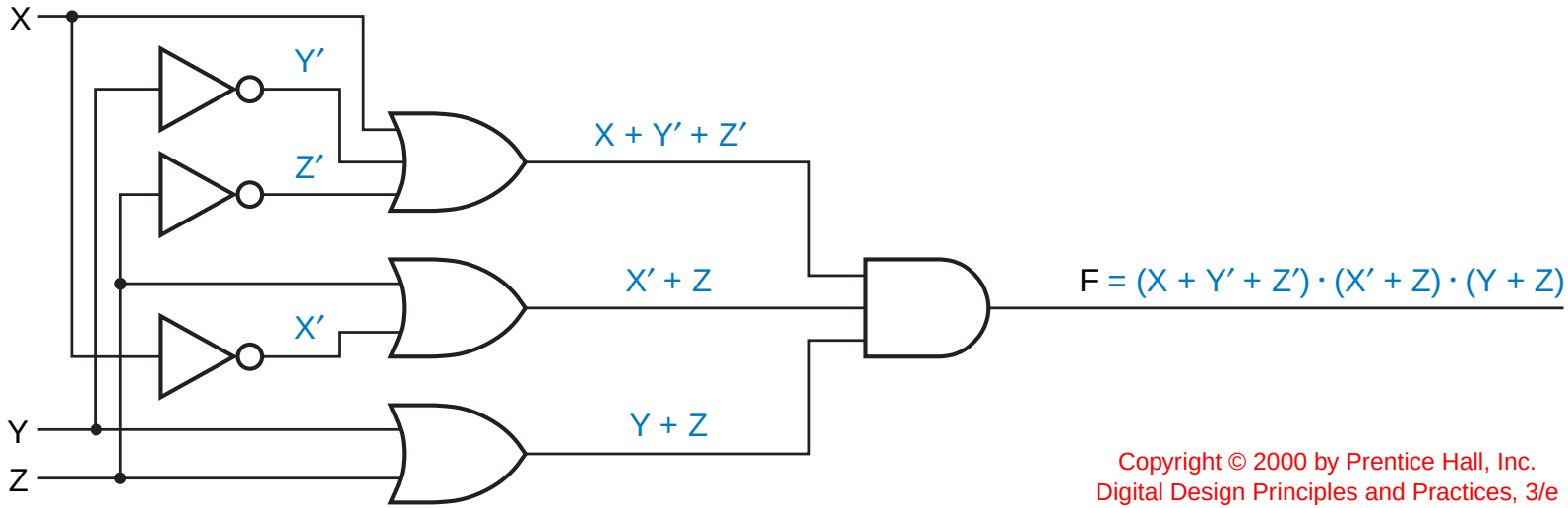

Figure 6: Two-level OR-AND circuit

From the logic expression of F from figure 4 we can obtain a product-of-sums expression, corresponding to the circuit from figure 6.

In order to obtain the product-of-sums expression, we use the relation $(a \cdot b) + (c \cdot d) = (a + c) \cdot (a + d) \cdot (b + c) \cdot (b + d)$, that can be proved using theorem T8' (distributivity of logic addition over logic multiplication)

An example with NAND and NOR gates

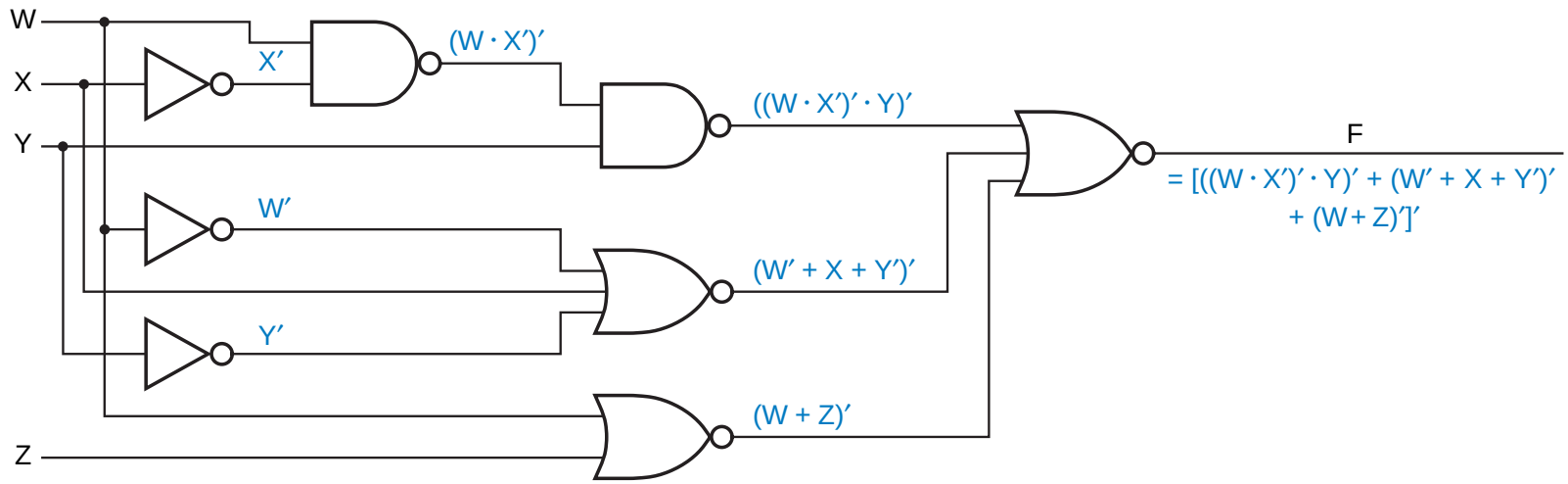

Figure 7: Algebraic analysis of a logic circuit with NAND and NOR gates

Here we have another example of circuit diagram, with NAND and NOR gates, and the logic expression of circuit's output F.

Applying graphically DeMorgan's theorems

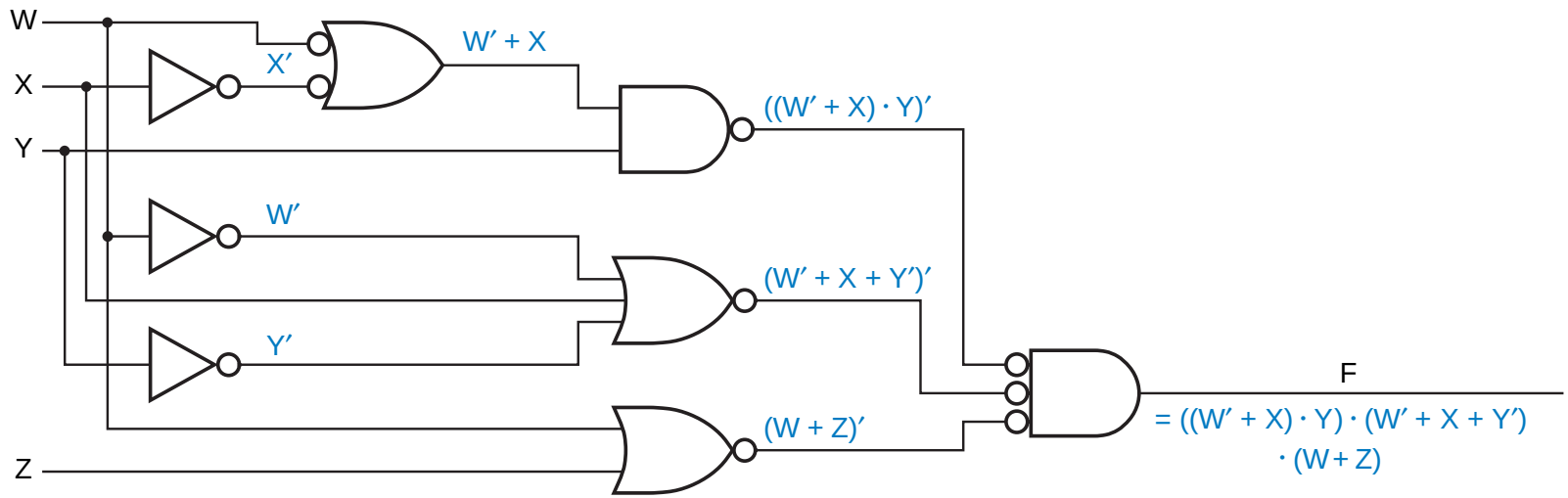

Figure 8: Algebraic analysis of the previous circuit after substituting some NAND and NOR gates

We can apply DeMorgan's theorem *graphically* (here by replacing the last gate symbol with an equivalent symbol) and obtain figure 8, which is the same circuit as that from figure 7.

A different circuit for the same logic function

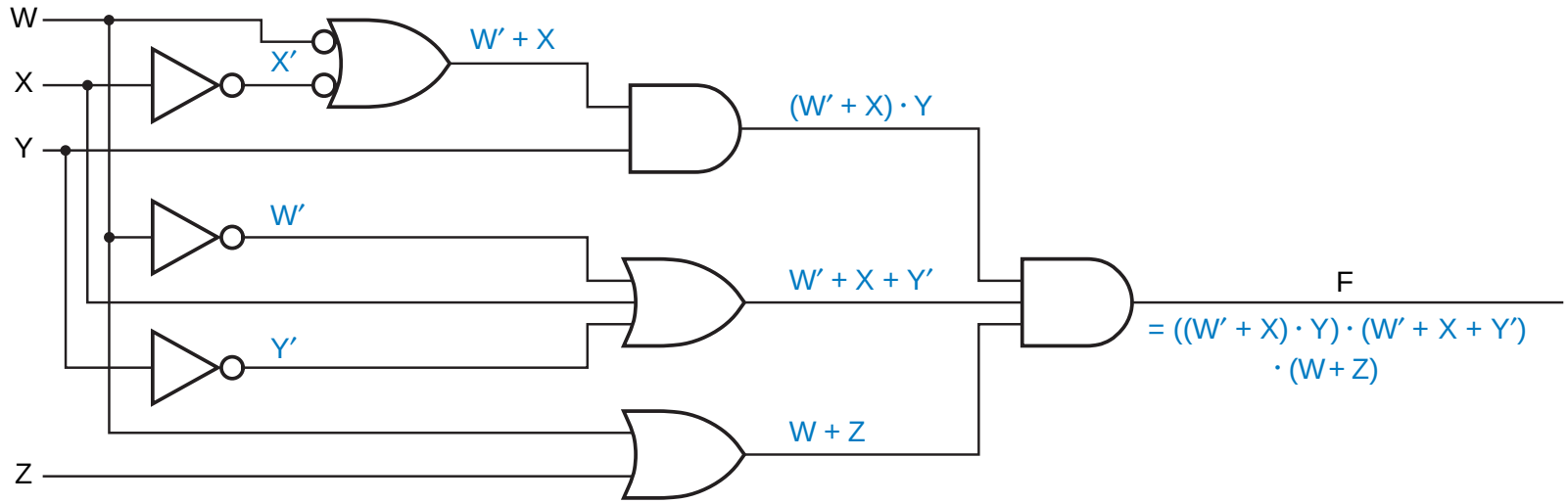

Figure 9: A different circuit for the same logic function

We can process the output function algebraically:
$F = [((W \cdot X')' \cdot Y)' + (W' + X + Y')' + (W + Z)']'$
If we start from the last inversion and apply DeMorgan, we obtain:
$F = ((W \cdot X')' \cdot Y) \cdot (W' + X + Y') \cdot (W + Z) =$
$((W' + X) \cdot Y) \cdot (W' + X + Y') \cdot (W + Z)$
The circuit from figure 9 corresponds to this expression of F. It is a different circuit from that from figures 7 and 8.

Another example: three circuits for the same logic function

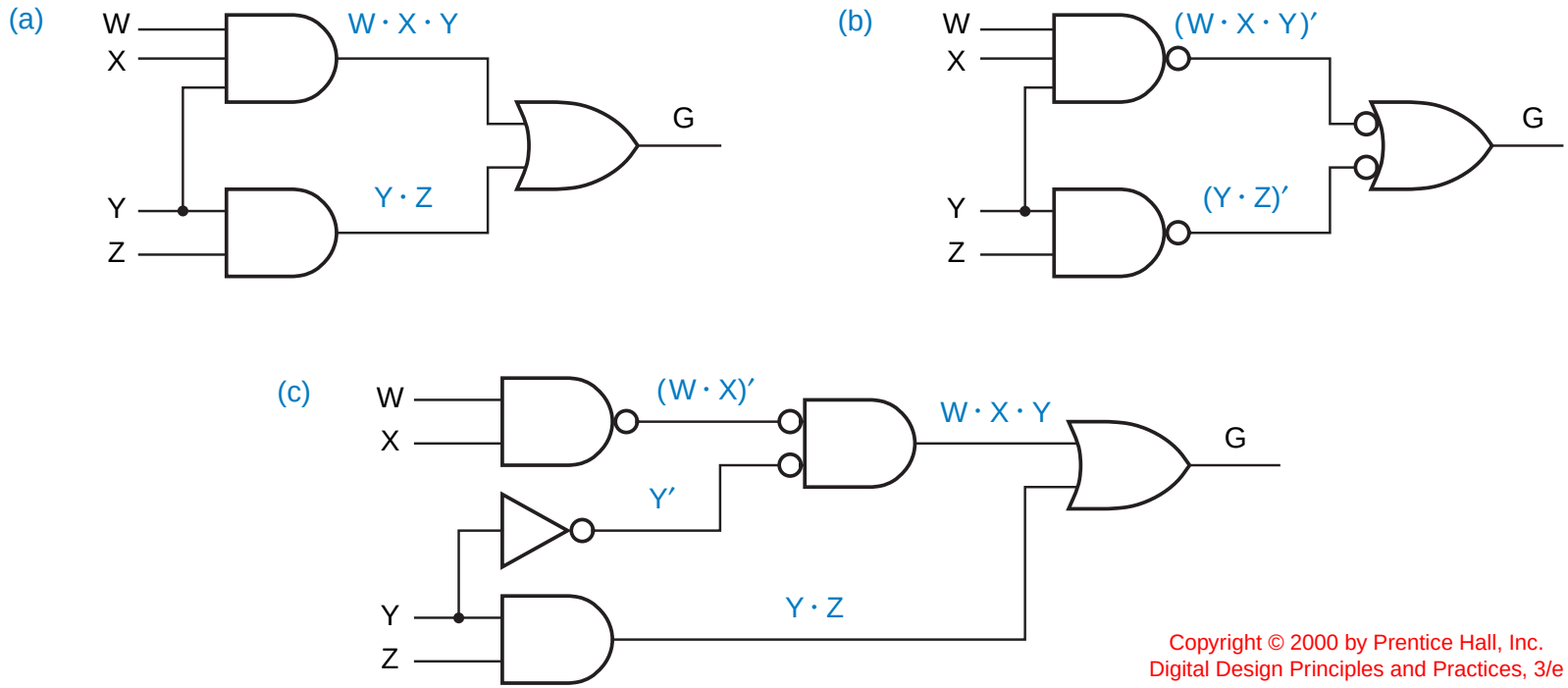

Figure 10: Three circuits for $G(W, X, Y, Z) = W \cdot X \cdot Y + Y \cdot Z$: (a) two-level AND-OR; (b) two-level NAND-NAND; (c) with two-input gates only

Sometimes structural information can be obtained from logical description, like in this figure.

Outline

Combinational-Circuit Synthesis: Introduction

- Usually the starting point for the combinational circuit design is a word description (given or developed by ourselves)
- We can translate this word description in a HDL (see the high-level description of the multiplexer in VHDL)
- Then the HDL tools can realize an automatic synthesis of the circuit
- However, it is important for us to be able to synthesize (combinational) circuits by hand for the following reasons:
  - sometimes the circuit obtained by automatic synthesis simply isn't good enough: e.g., for critical parts of a design, like certain parts from a microprocessor
  - sometimes the automatic tool "runs amok" and obtain a very poor result
  - Hence, it is important to be able at least to evaluate the result of the synthesis process and, if the performance of the obtained circuit is not good enough, to improve it

Outline

Circuit Descriptions

1. Sometimes the description of a circuit is a list of input combinations for which the output should be on (or off).
   - This is a verbal description of the minterm (maxterm) list using $\sum$ ($\prod$), notations, or of the canonical sum (product)
   - Example: the description of a 4-bit prime-number detector
2. More often we describe a logic function in words, using connectives "and", "or" and "not"
   - See the alarm example
   - **Definition:** A circuit *realizes* (i.e., makes real) an expression if its output function equals the expression, and the circuit is called *a realization* of the function.
3. Sometimes we prefer to use the truth table:
   - usually when the word description is imprecise, in the sense that it is incomplete and we need to use our knowledge of the problem (e.g. in order to eliminate some impossible combinations of inputs)

Circuit Descriptions

1. The prime-number detector (figure 11):
   - The description of a 4-bit prime-number detector can be: "Given a 4-bit number combination $N = N_3N_2N_1N_0$, produce a 1 output for $N = 1, 2, 3, 5, 7, 11, 13$, and 0 otherwise"
   - The output function is $F = \sum N_3, N_2, N_1, N_0(1, 2, 3, 5, 7, 11, 13) = N_3' \cdot N_2' \cdot N_1' \cdot N_0 + N_3' \cdot N_2' \cdot N_1 \cdot N_0 + N_3' \cdot N_2 \cdot N_1' \cdot N_0 + N_3' \cdot N_2 \cdot N_1 \cdot N_0 + N_3 \cdot N_2' \cdot N_1 \cdot N_0 + N_3 \cdot N_2 \cdot N_1' \cdot N_0$
2. The alarm circuit (figure 12):
   - Description in words: "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"
   - Translation into algebraic expression: $ALARM = PANIC + ENABLE \cdot EXITING' \cdot SECURE'$
   - $SECURE = WINDOW \cdot DOOR \cdot GARAGE$
   - We defined the auxiliary variable *SECURE*

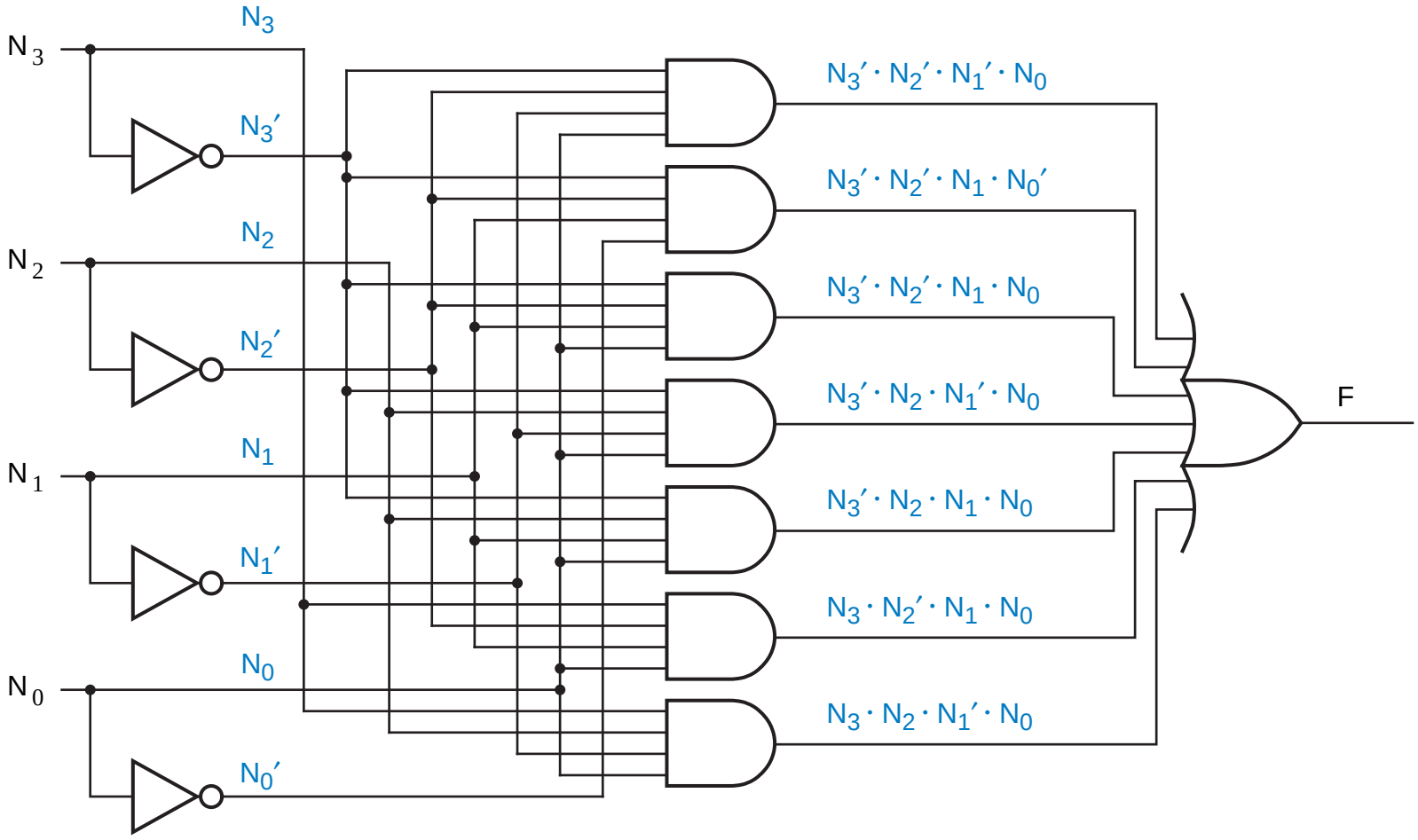

Figure 11: Canonical sum design for 4-bit prime-number detector

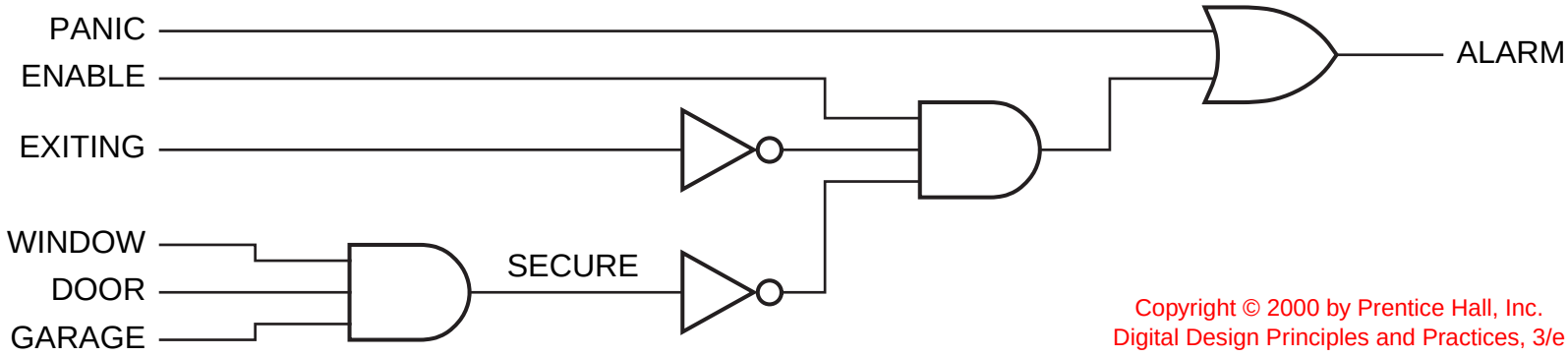

Figure 12: Alarm circuit derived from logic expression

The output function of the alarm circuit is: $ALARM = PANIC + ENABLE \cdot EXITING' \cdot (WINDOW \cdot DOOR \cdot GARAGE)'$

We can multiply out this expression to obtain the sum-of-products circuit (fig 13)

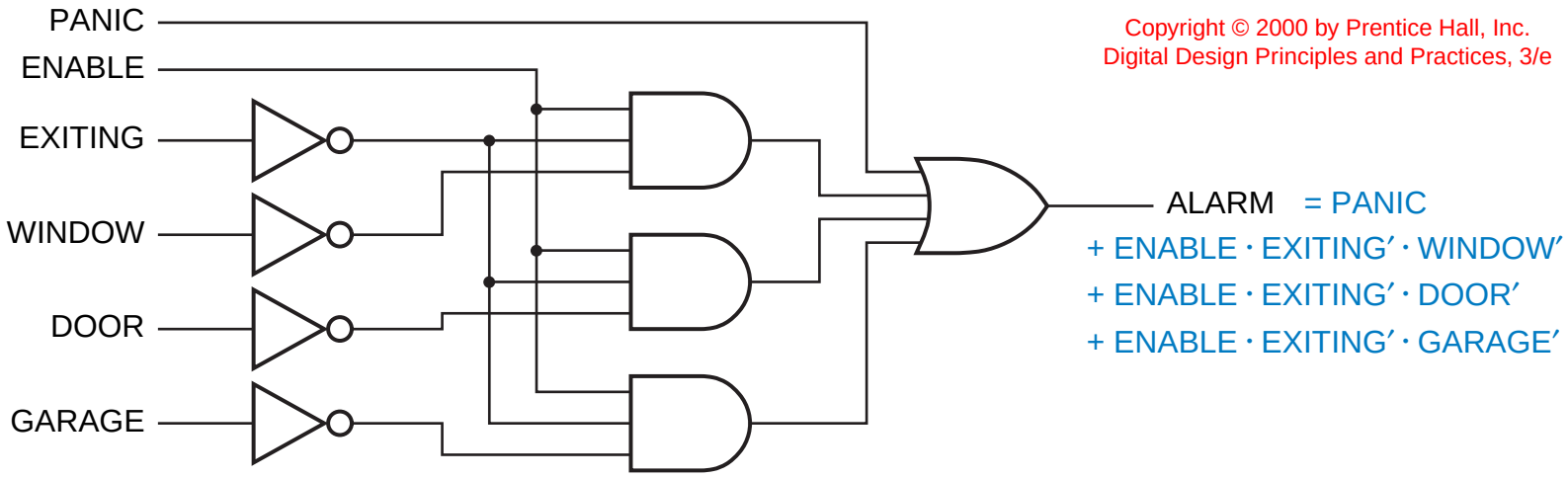

Figure 13: Sum-of-products version of alarm circuit

Outline

Circuit Manipulation

- Remember that NAND and NOR gates are faster (and cheaper) that AND and OR gates in many technologies (e.g. CMOS, TTL)
- However, usually we describe a problem in a "natural" way using propositions with AND and OR, not with NAND and NOR
- It means that sometimes we obtain a circuit diagram with AND and OR gates, and want to transform it with NAND and NOR gates
- Figure 14 gives such an example of transformation, starting from a sum-of-products form
- In figure 14 the inverters needed to complement the inputs are not shown.

Sum-of-products realizations: from AND-OR to NAND-NAND

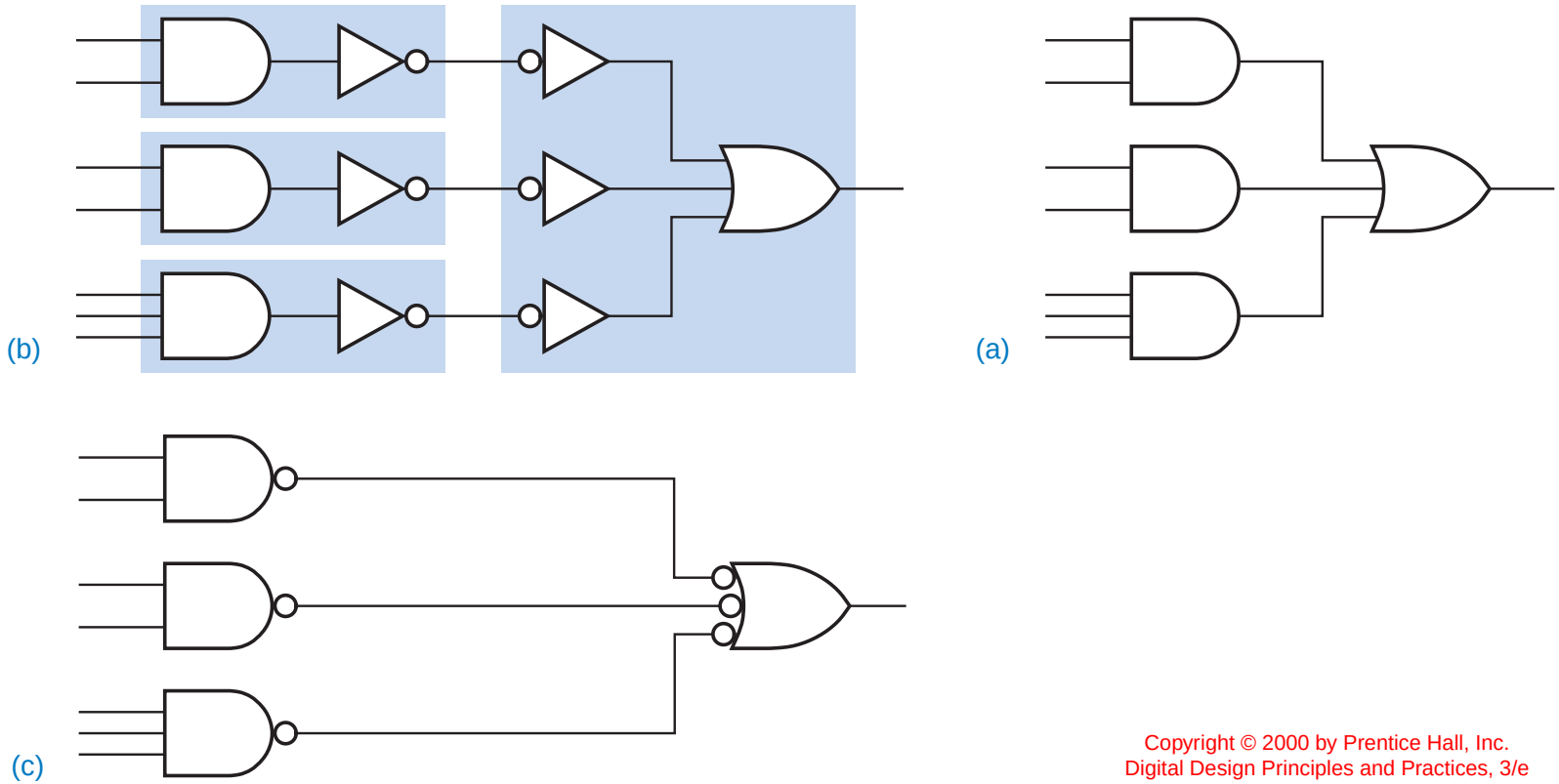

Figure 14: Alternative sum-of-products realizations: (a) AND-OR; (b) AND-OR with extra inverter pairs; (c) NAND-NAND

Transformation of a circuit from AND-OR to NAND-NAND form

1. Obtain the sum-of-products expression by "multiplying out" the given logic expression
2. Represent the circuit diagram for the sum-of-products expression. We obtain a two-level AND-OR circuit.
3. Insert (graphically) a pair of inverters between AND gates' outputs and the corresponding OR gates' inputs
   - According to theorem T4, involution $((X')' = X)$ the inserted inverters do not affect the output expression of the circuit
4. The first level of inserted inverters are absorbed into the outputs of the AND gates, obtaining NAND gates on the first level of gates
5. On the last level of gates we will obtain NOT-OR gates, which are actually different symbols for NAND gates.

Transformation of a circuit from AND-OR to NAND-NAND form

In conclusion, a two-level AND-OR gate circuit can be converted to a two level NAND-NAND circuit by simply substituting gate symbols (AND with NAND and OR with NAND).

If there are product terms (in the sum-of-products) that contain only one literal, then we may gain or lose inverters in the transformations (see fig 15, inputs W and Z).

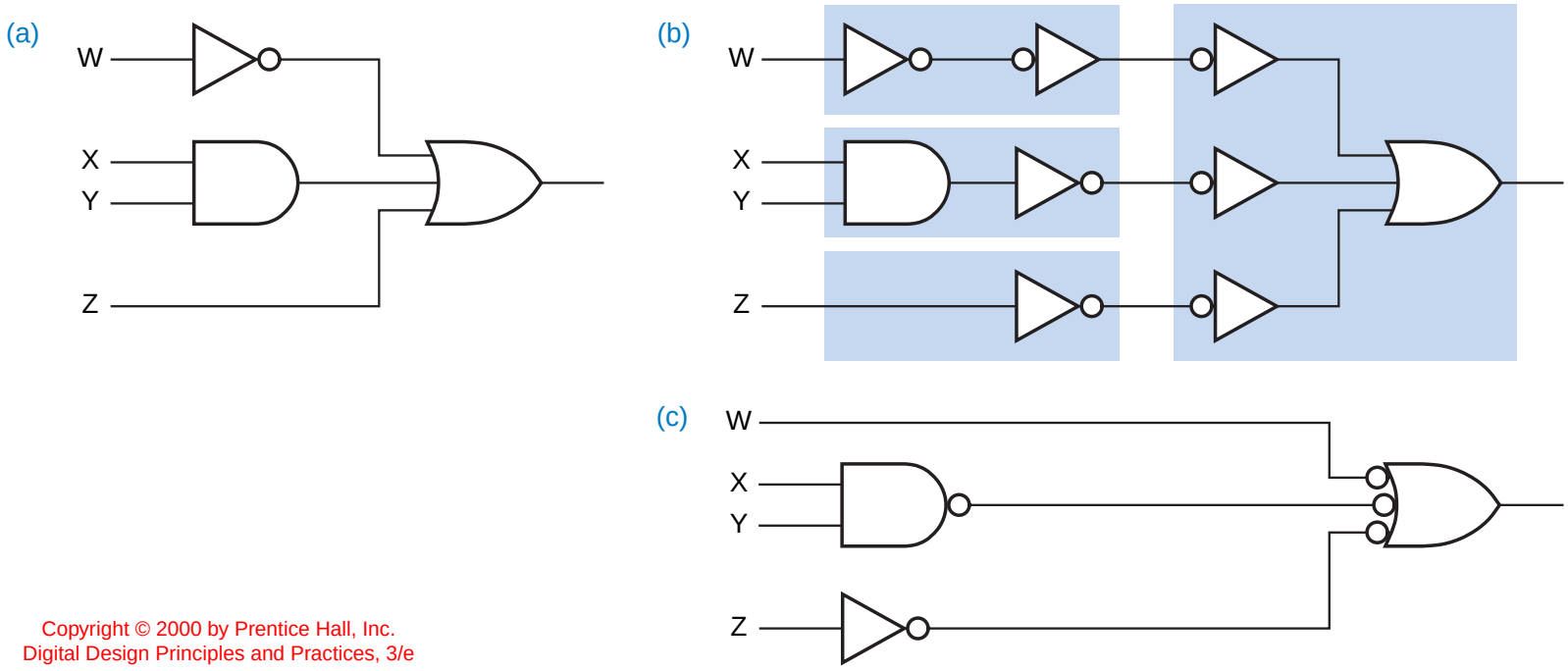

Figure 15: Another two-level sum-of-product circuit: (a) AND-OR; (b) AND-OR with extra inverter pairs; (c) NAND-NAND

Product-of-sums realizations: from OR-AND to NOR-NOR

- A similar approach can be taken for a product-of-sum realization of a circuit:
  1. First, obtain the product-of-sums expression and the corresponding two-level OR-AND circuit
  2. Insert two levels of inverters between the outputs of the OR gates from the first level and the corresponding inputs of the AND gates from the second level
  3. Replace the OR-NOT with NOR symbols and the NOT-AND with NOR symbols
  4. Pay attention to sum terms consisting of a single literal, that may gain or lose inverters
- Figure 17 illustrates this case.

Product-of-sums realizations: from OR-AND to NOR-NOR

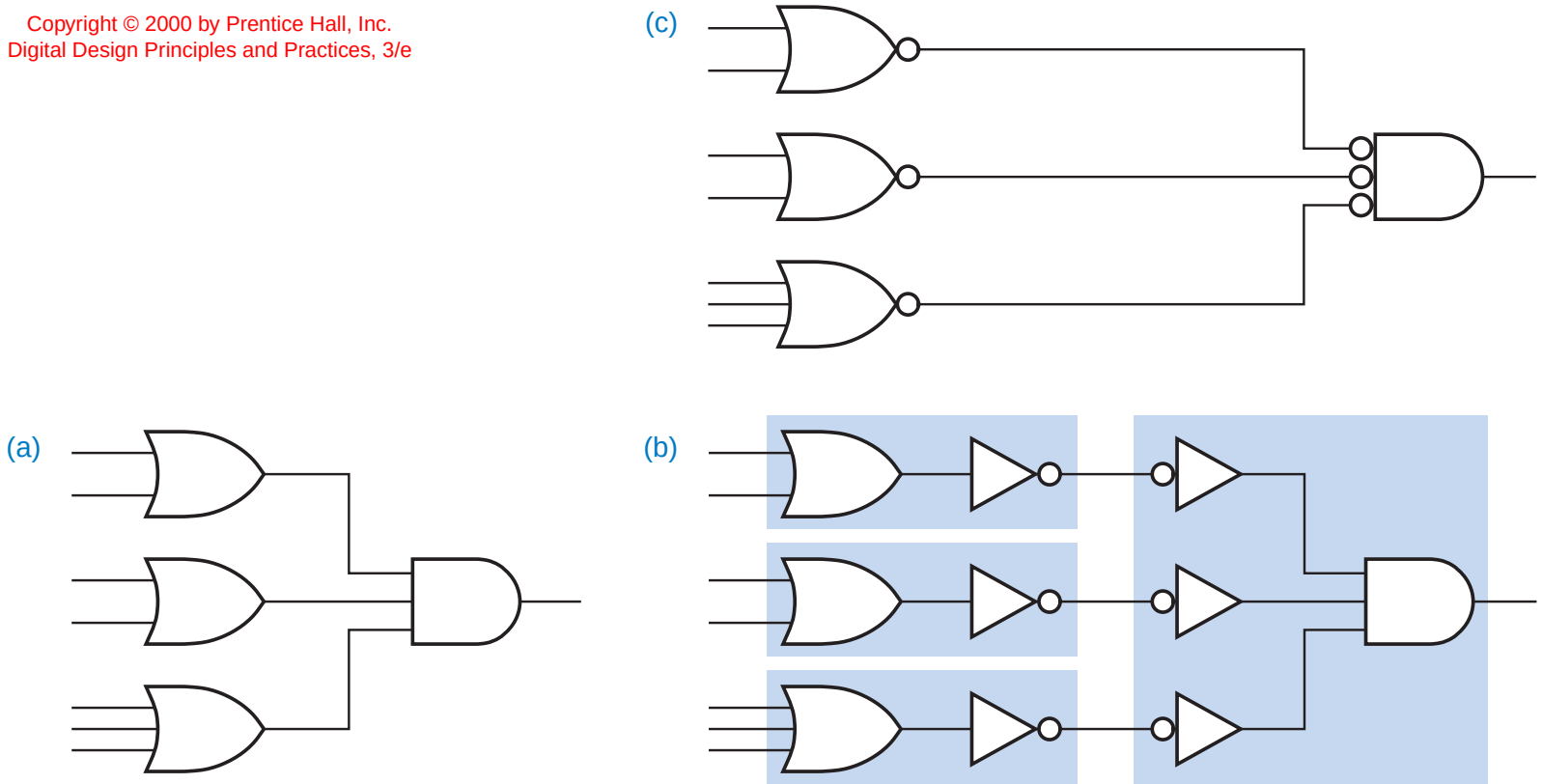

Figure 16: Realization of a product-of-sums expression: (a) OR-AND; (b) OR-AND with extra inverter pairs; (c) NOR-NOR

The general case of a two-level circuit with AND and OR gates

- We can apply the same method for any two-levels circuit consisting of AND and OR gates, and transform it into a circuit with NAND and NOR gates
- Figure 17 (a) shows the original circuit
- Figure 17 (b) presents the circuit after the insertion of two-levels of inverters (only inverting bubbles are shown on the figure)
- After this step we can obtain non-standard gates (e.g. a gate with some inputs inverted and some inputs non-inverted).
- Figure 17 (c) and (d) solve the problem of the non-standard gate, by inserting another inverter
- Solution from (d) is better than in (c) because fig (c) adds a supplementary level of gate delay

The general case of a two-level circuit with AND and OR gates

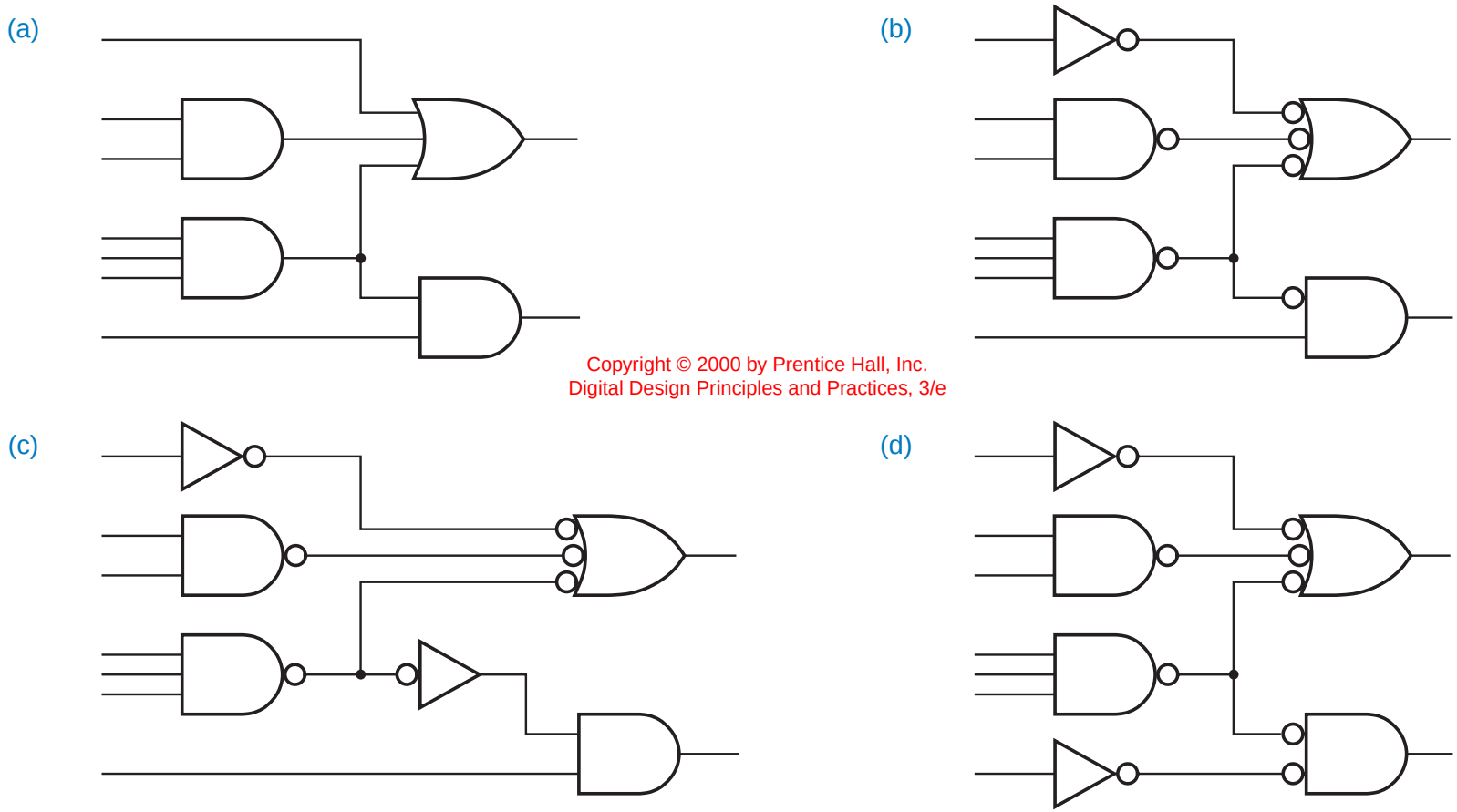

Figure 17: Logic symbol manipulation: (a) original circuit; (b) transformation with a nonstandard gate; (c) inverter used to eliminate the nonstandard gate; (d) preferred inverter placement

Outline

Combinational-Circuit Minimization

- *Minimization* of a combinational circuit means the reduction of the number and size of gates that are needed to build the circuit.
- The methods presented here start from the truth table or a minterm list or a maxterm list.
- If the circuit is not described in one of these ways, we must bring the logic function that described the circuit to one of these forms.
- The minimization methods reduce the cost of a two-level AND-OR, OR-AND, NAND-NAND or NOR-NOR circuit in three ways:
  1. By minimizing the number of first-level gates
  2. By minimizing the number of inputs of the first-level gates
  3. By minimizing the number of inputs of the second-level gates. This is a consequence of the reduction of the number of first-level gates.

Combinational-Circuit Minimization

- The minimization methods assume that both true and complemented inputs are available (they do not consider the cost of input inverters).
- In some technologies (e.g. PLDs) inputs are available in both true and complemented form, but in other technologies they are not (e.g. in ASICs)
- Minimizations are based on a generalization of combining theorems (T10 and T10'):
  - $given\_product\_term \cdot Y + given\_product\_term \cdot Y' = given\_product\_term$
  - $(given\_sum\_term + Y) \cdot (given\_sum\_term + Y') = given\_sum\_term$
- In words: if two product or sum terms differ only in the complementing or not of one variable, then we can combine them into a single term without that variable

Combinational-Circuit Minimization

- If we try a minimization on the 4 bit prime-number detector circuit by applying repeatedly this method we can obtain the circuit from figure 18
- Starting from function $F = \sum_{N_3,N_2,N_1,N_0}(1, 3, 5, 7, 2, 11, 13)$, we combine minterms 1, 3, 5 and 7 to obtain the function realized by the circuit from figure 18.
- The obtained circuit has 3 fewer gates and one of the input gates has 2 fewer inputs.
- However, if the algebraic expression that described the circuit is quite complex, this method does not guaranty "the best" solution (minimization)
- Hence, we will use the Karnaugh maps for minimization of logic functions

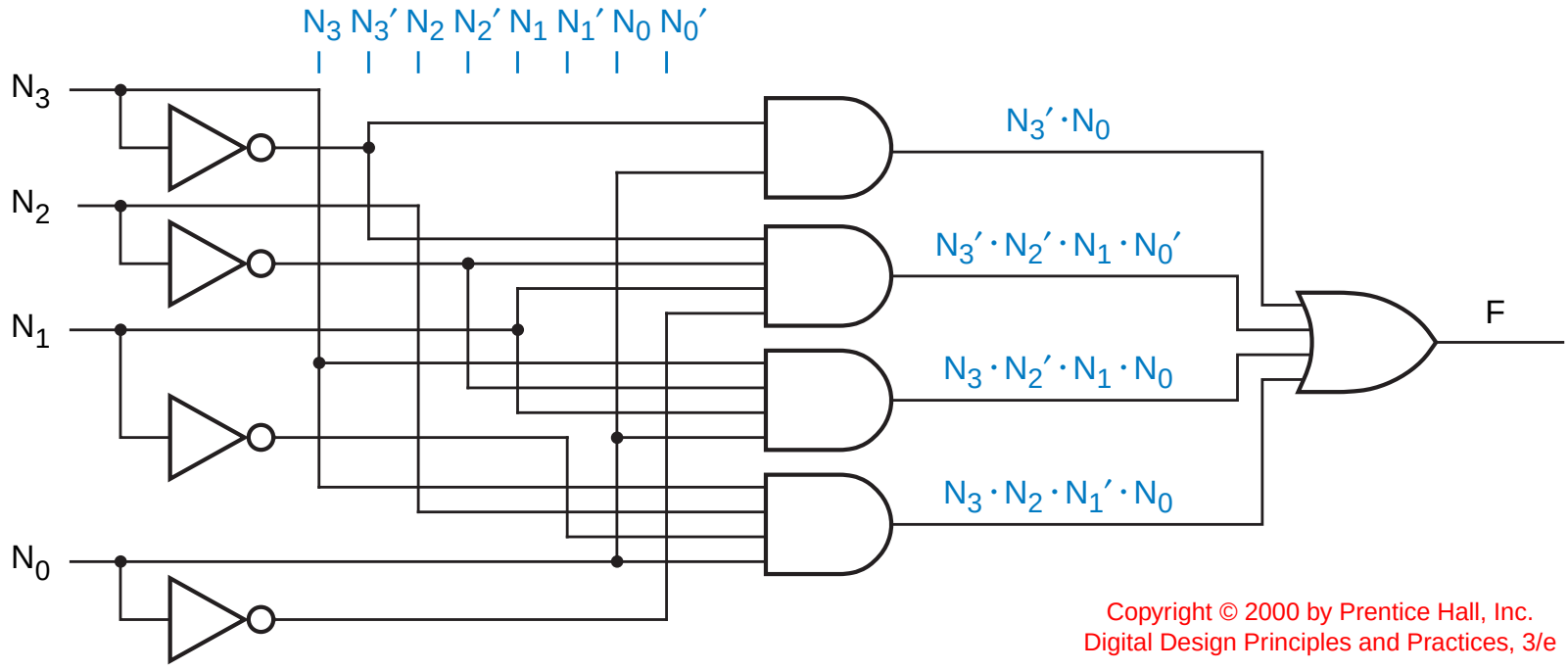

Figure 18: Simplified sum-of-products realization for 4-bit prime-number detector