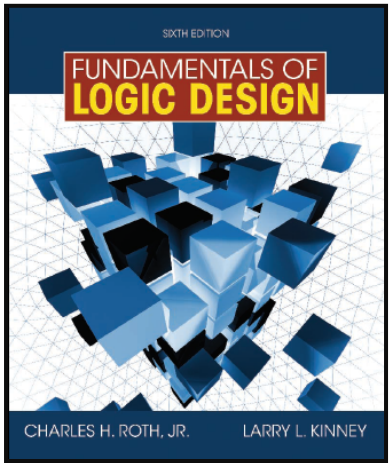
SLIDES FOR

CHAPTER 8

COMBINATIONAL CIRCUIT DESIGN AND SIMULATION USING GATES



This chapter in the book includes:

Click the mouse to move to the next page.
Use the ESC key to exit this chapter.

Design of Circuits with Limited Gate Fan-In

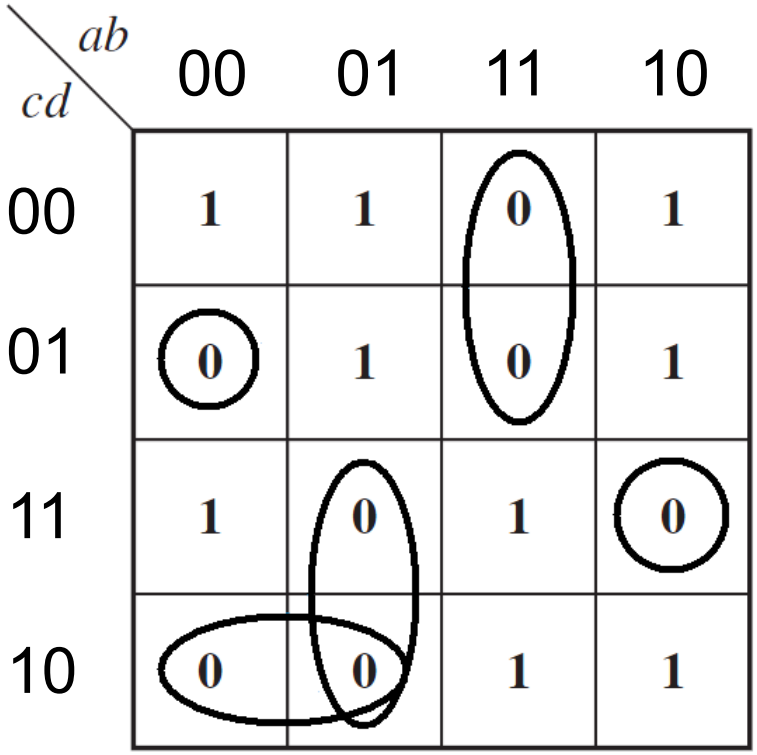
In practical logic design problems, the maximum number of inputs on each gate (or the fan-in) is limited.

If a two-level realization of a circuit requires more gate inputs than allowed, factoring the logic expression to obtain a multi-level realization is necessary.

Section 8.2 (p. 220)

Example 1

Realize $f(a, b, c, d) = \Sigma\, m(0, 3, 4, 5, 8, 9, 10, 14, 15)$ using three-input NOR gates.



First we find a minimum sum-of-products of f'.

$$f' = a'b'c'd + ab'cd + abc' + a'bc + a'cd'$$

Section 8.2 (p. 220)

Then the expression for f' is factored to reduce the maximum number of gate inputs to three and, then it is complemented.

$$f' = b'd(a'c' + ac) + a'c(b + d') + abc'$$
$$f = [b + d' + (a + c)(a' + c')][a + c' + b'd][a' + b' + c]$$

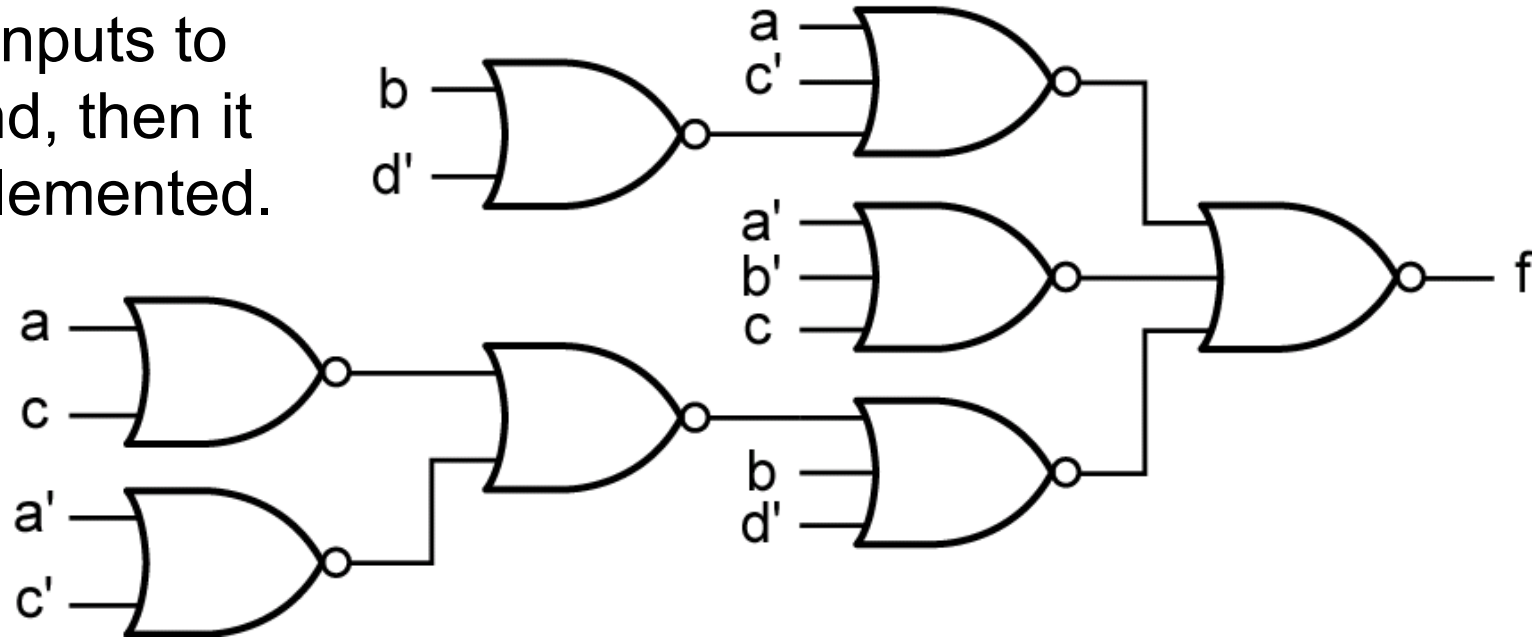


Figure 8-1

Example 2

Realize the functions given in Figure 8-2, using only two-input NAND gates and inverters. If we minimize each function separately, the result is

$$f_1 = b'c' + ab' + a'b$$
$$f_2 = b'c' + bc + a'b$$
$$f_3 = a'b'c + ab + bc'$$

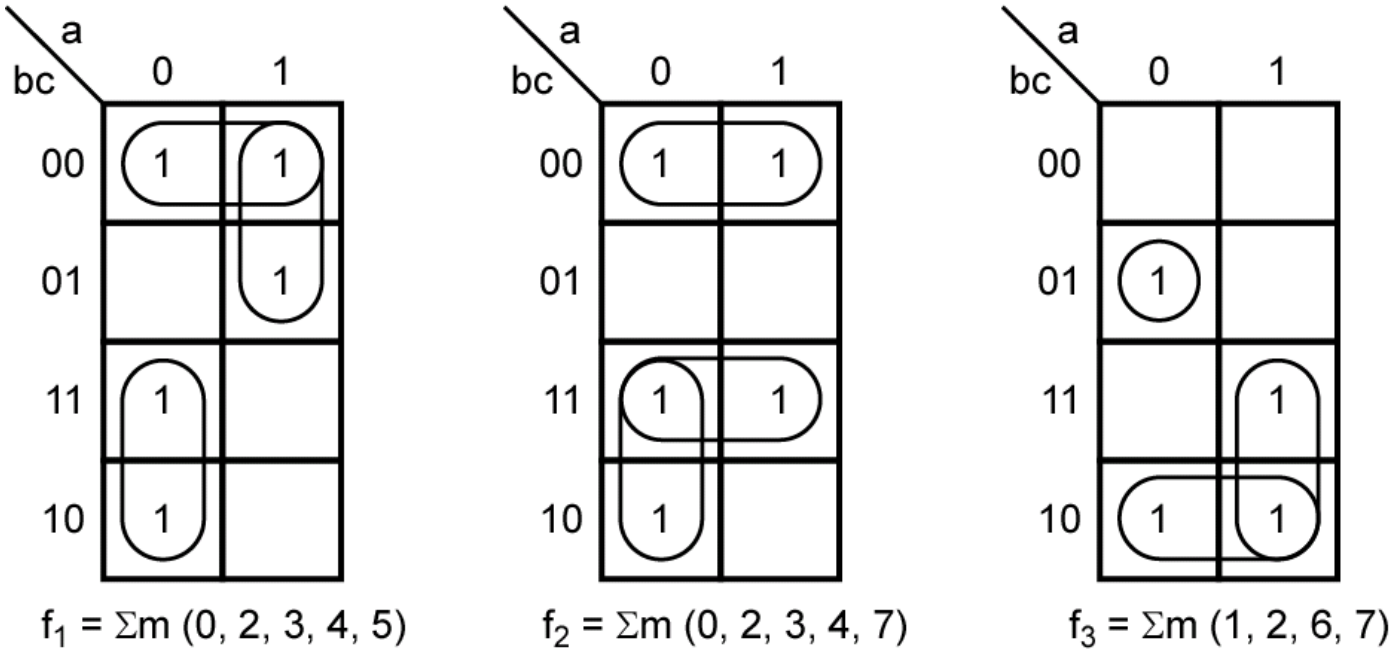


Figure 8-2

Each function requires a three-input OR gate, so we will factor to reduce the number of gate inputs:

$f_1 = b'(\underline{a + c'}) + \underline{a'b}$

$f_2 = b(a' + c) + b'c'$ or $f_2 = (b' + c)(b + c') + \underline{a'b}$

$f_3 = a'b'c + b(\underline{a + c'})$

The second expression for f_2 has a term common to f_1, so we will choose the second expression. We can eliminate the remaining three-input gate from f_3 by noting that

$$a'b'c = a'(b'c) = a'(b + c')'$$

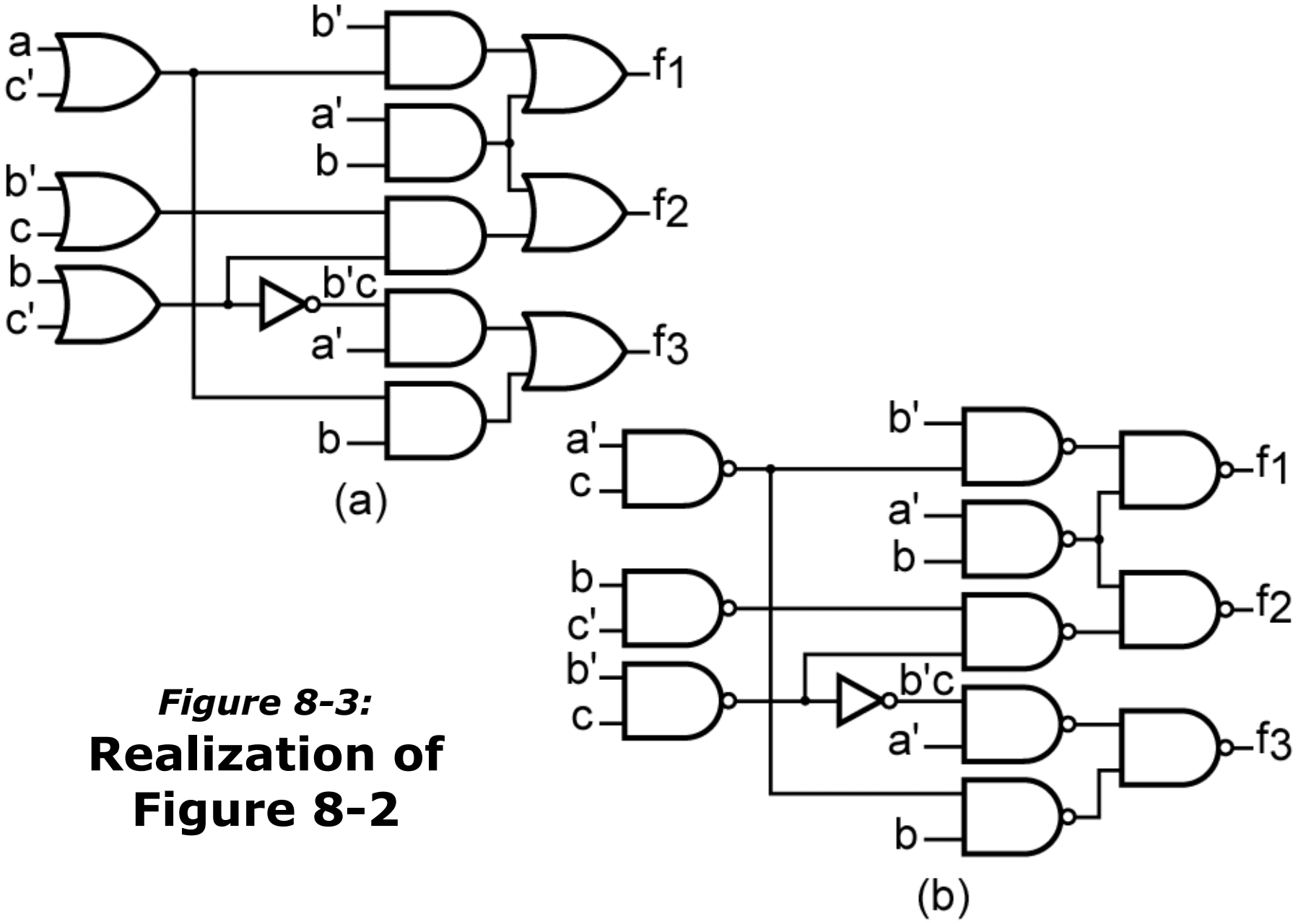


Figure 8-3:

Realization of Figure 8-2

Gate Delays and Timing Diagrams

When the input to a logic gate is changed, the output will not change instantaneously. The transistors or other switching elements within the gate take a finite time to react to a change in input, so that the change in the gate output is delayed with respect to the input change.

Section 8.3 (p. 222)

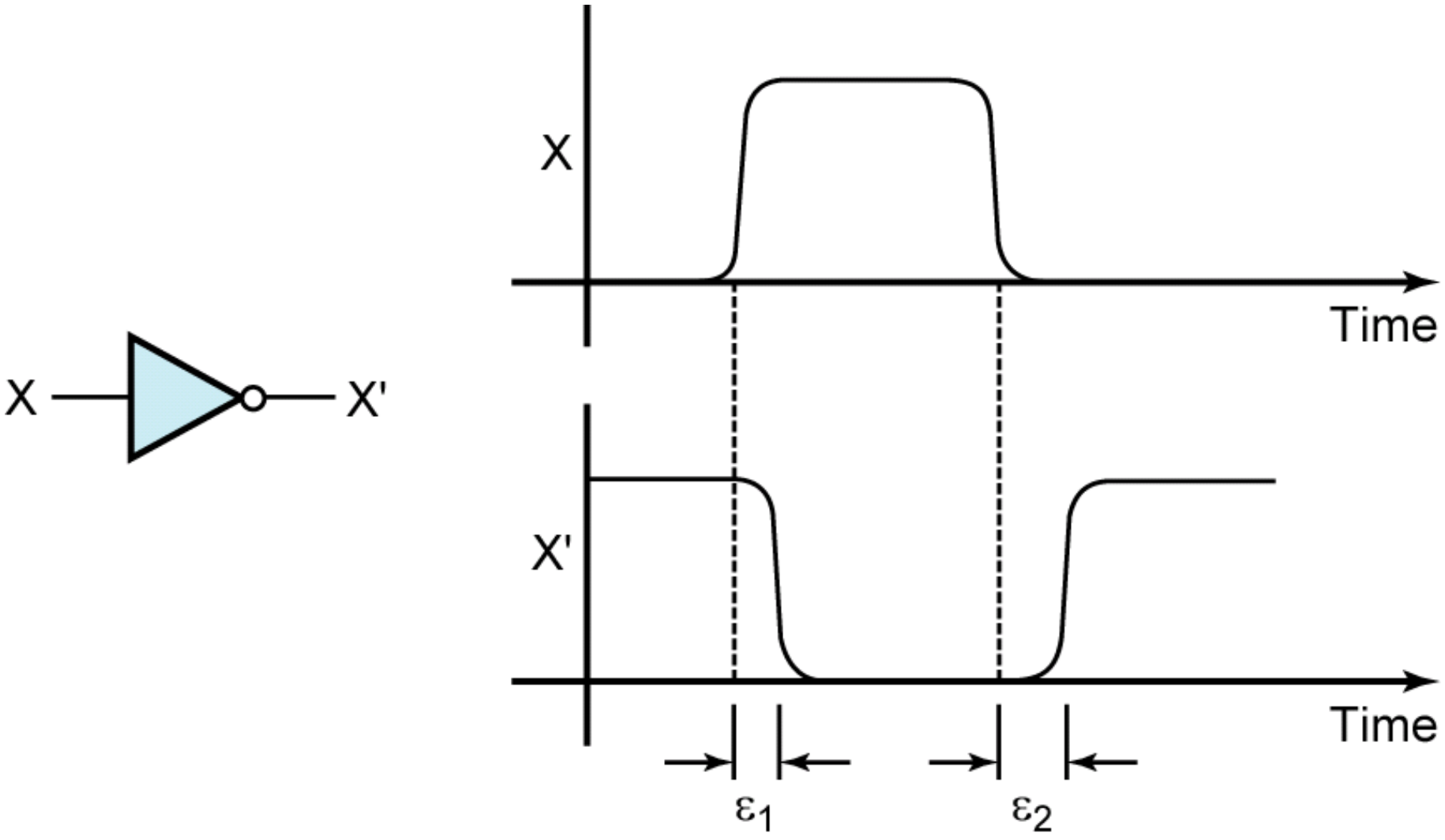


Figure 8-4: **Propagation Delay in an Inverter**

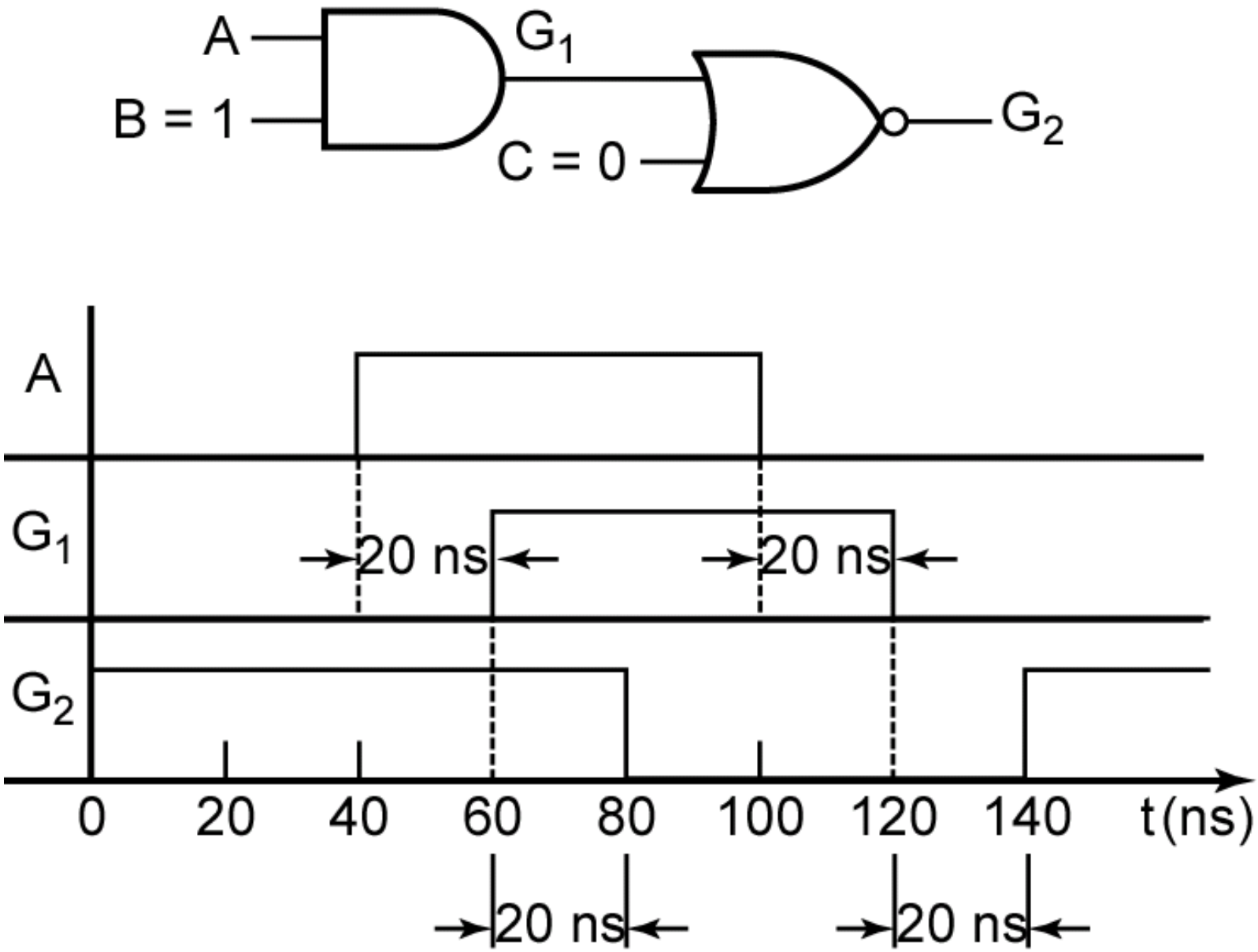


Figure 8-5: **Timing Diagram for AND-NOR Circuit**

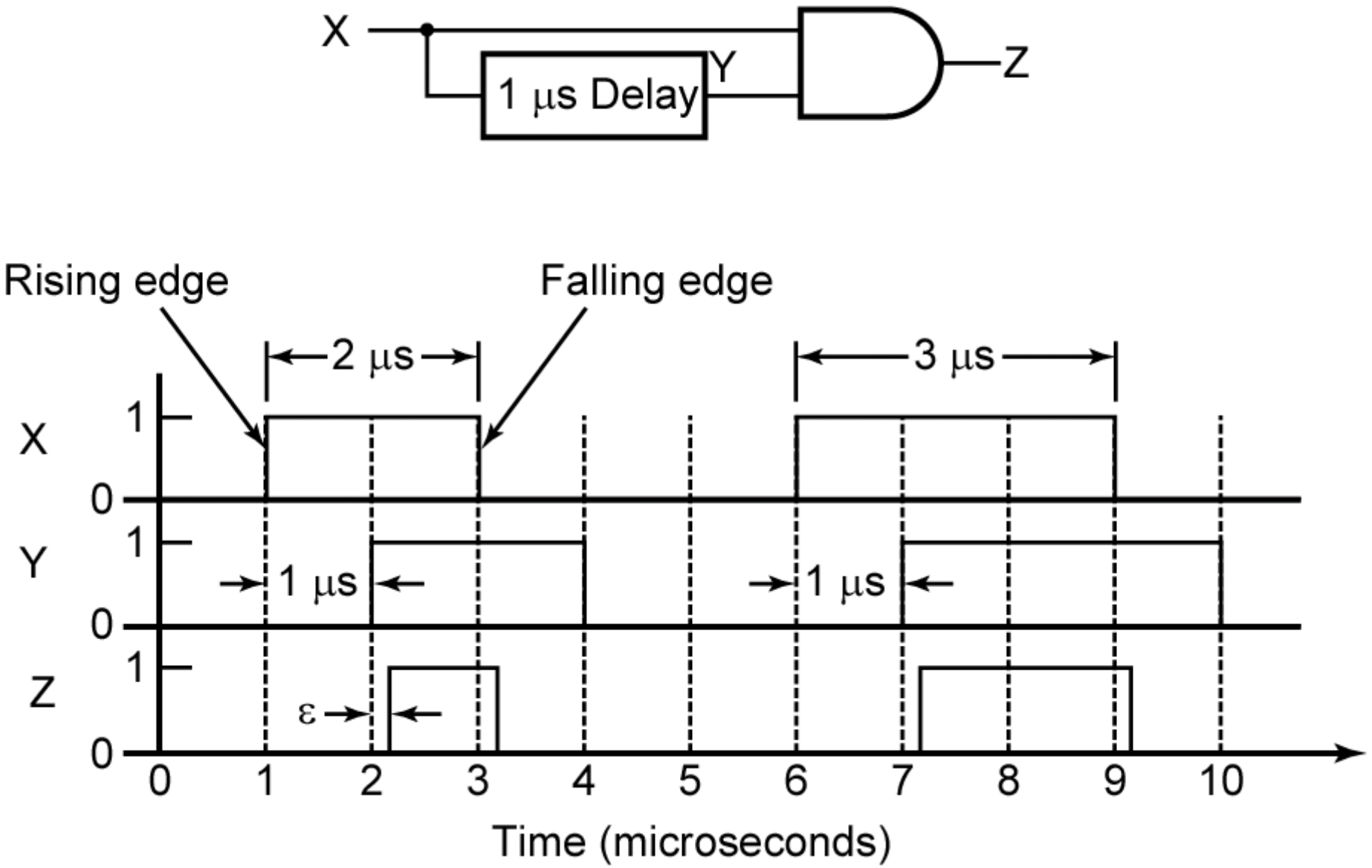


Figure 8-6: **Timing Diagram for Circuit with Delay**

Hazards in Combinational Logic

When the input to a combinational circuit changes, unwanted switching transients may appear in the output. These transients occur when different paths from input to output have different propagation delays.

Section 8.4 (p. 224)

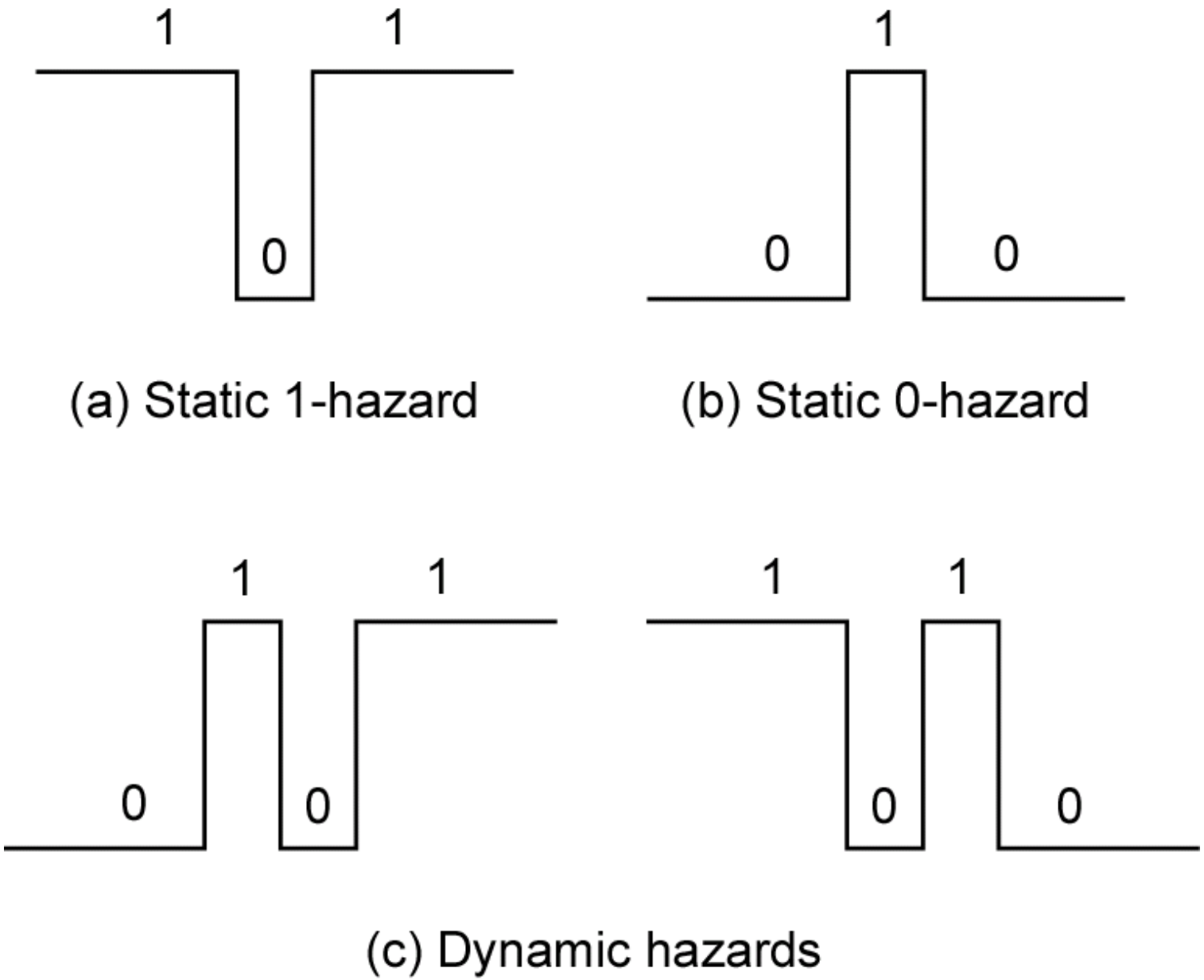
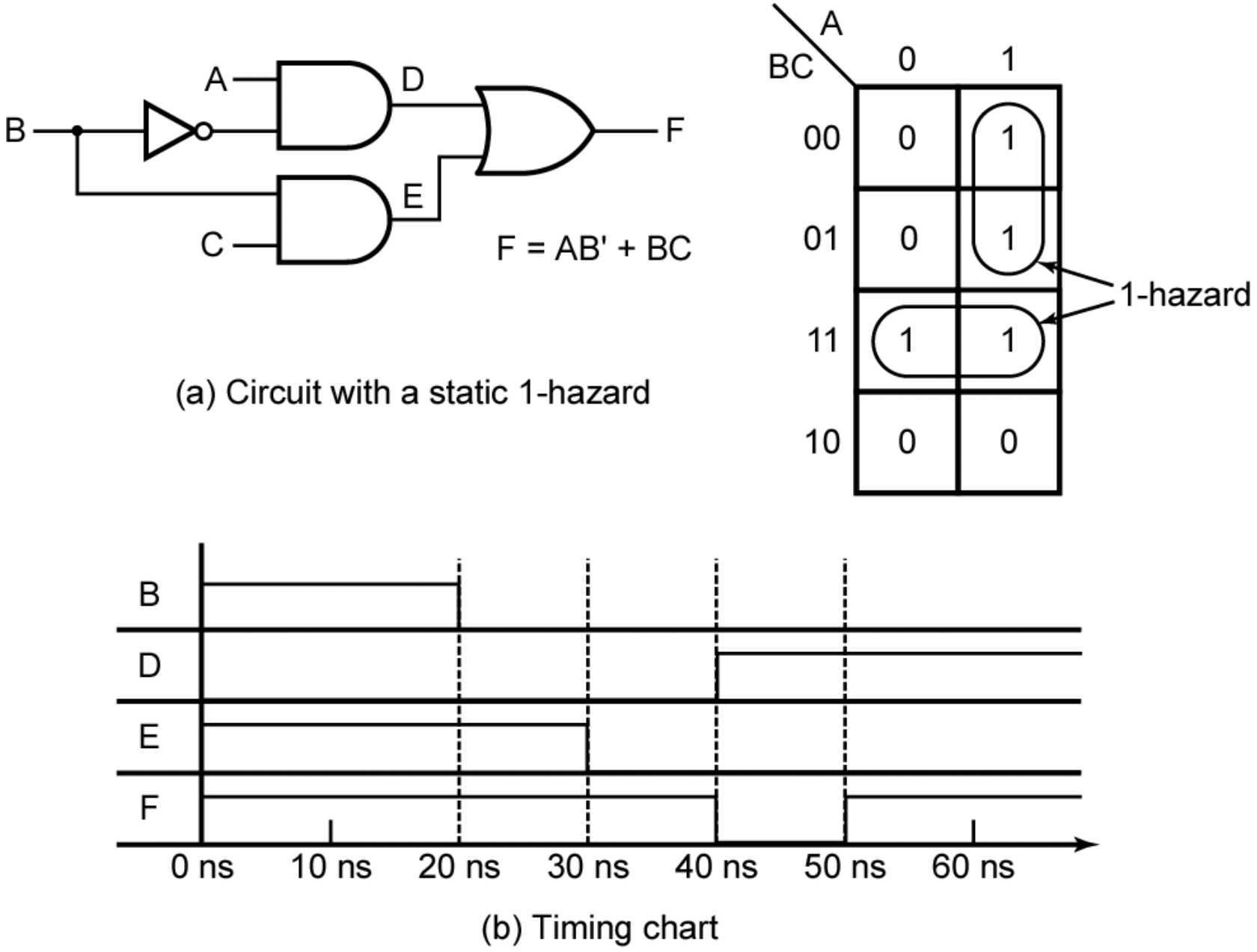


Figure 8-7: **Types of Hazards**

We can detect hazards in a two-level AND-OR circuit using the following procedure:

1. Write down the sum-of-products expression for the circuit.
2. Plot each term on the map and loop it.
3. If any two adjacent 1′s are not covered by the same loop, a 1-hazard exists for the transition between the two 1′s. For an *n*-variable map, this transition occurs when one variable changes and the other $n - 1$ variables are held constant.



(a) Circuit with a static 1-hazard

(b) Timing chart

Figure 8-8: **Detection of a 1-Hazard**

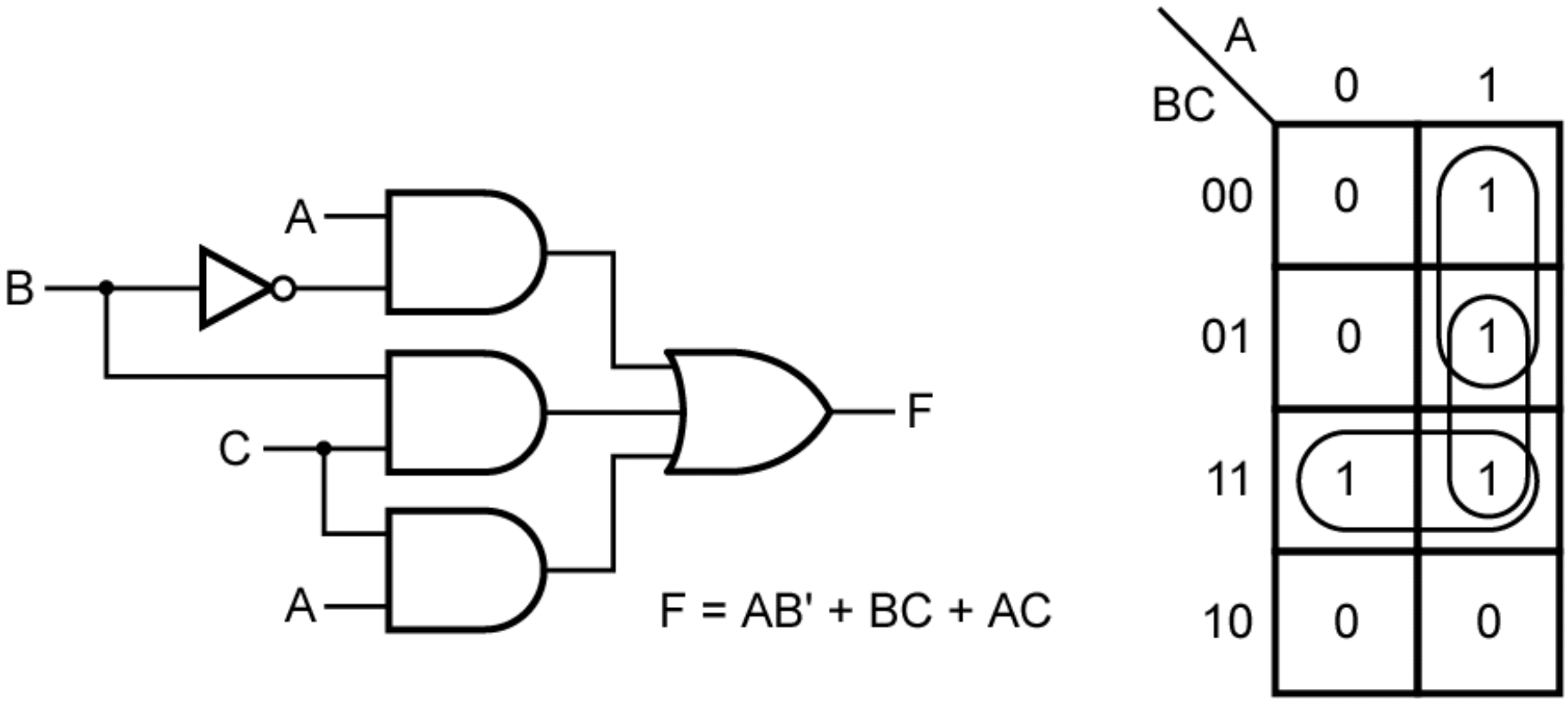
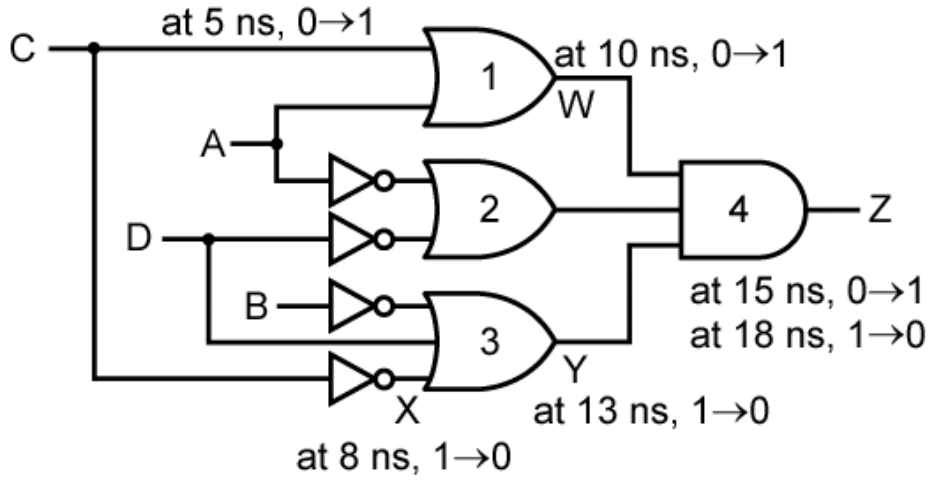
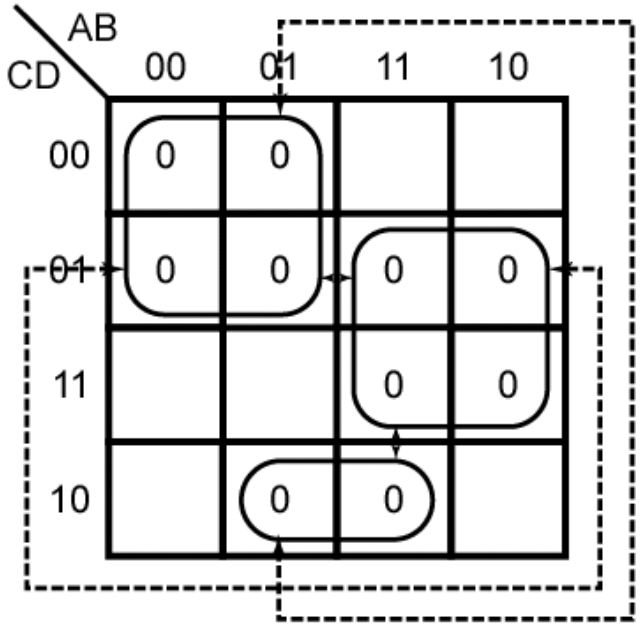


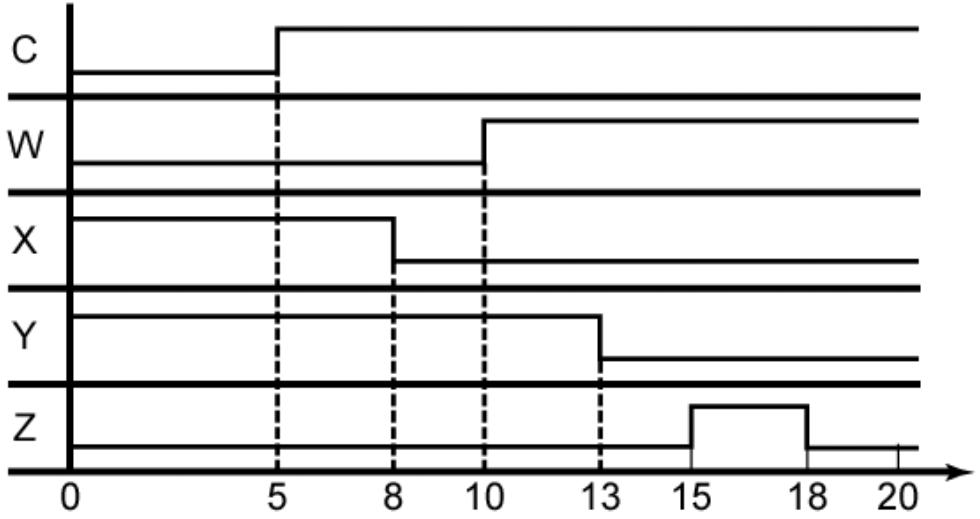
Figure 8-9: **Circuit with Hazard Removed**



(a) Circuit with a static 0-hazard



(b) Karnaugh map for circuit of (a)



(c) Timing diagram illustrating 0-hazard of (a)

Figure 8-10: **Detection of a Static 0-Hazard**

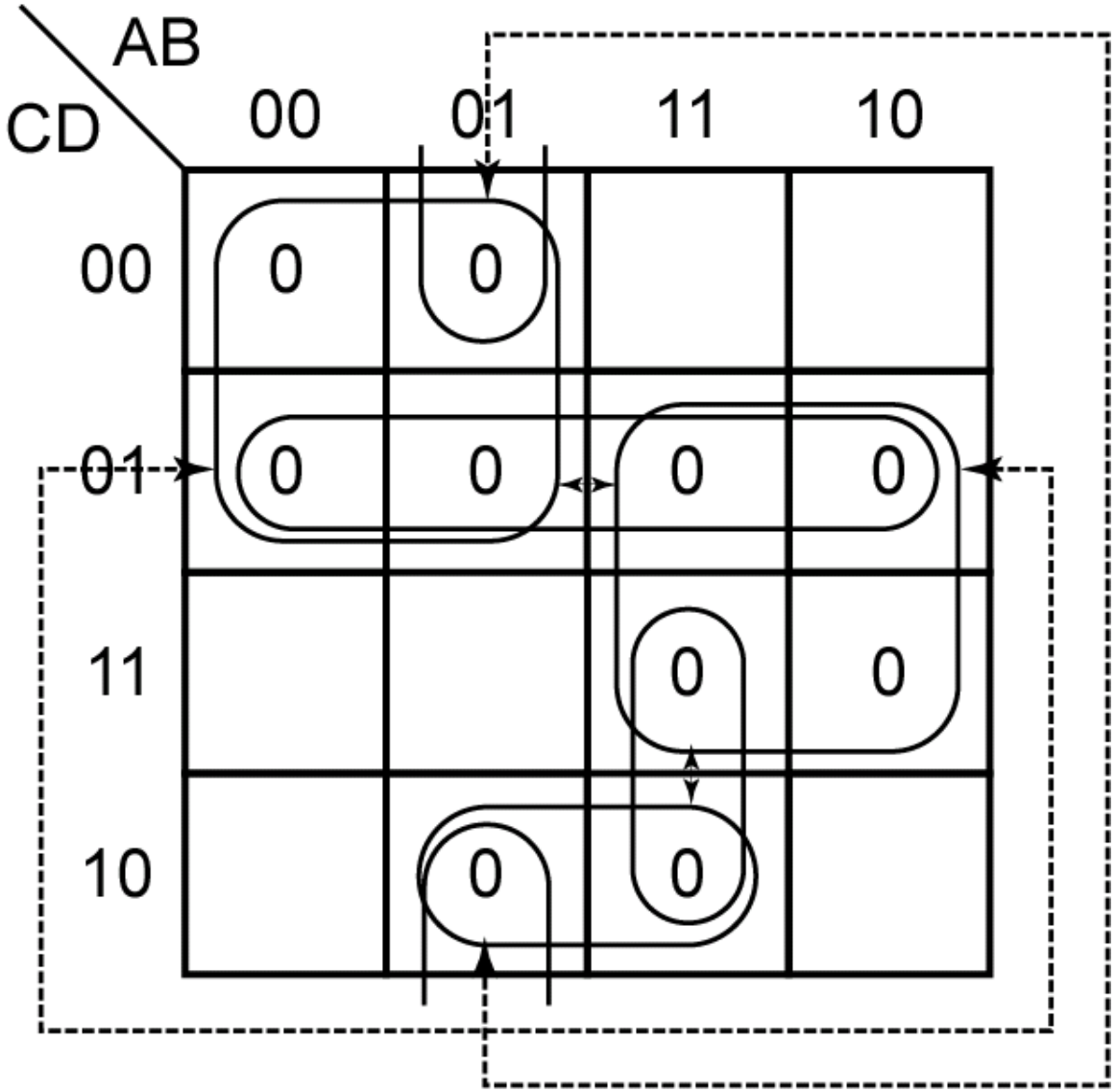


Figure 8-11: **Karnaugh Map Removing Hazards of Figure 8-10**

To design a circuit which is free of static and dynamic hazards, the following procedure may be used:

1. Find a sum-of-products expression (F^t) for the output in which every pair of adjacent 1′s is covered by a 1-term. (The sum of all prime implicants will always satisfy this condition.) A two-level AND-OR circuit based on this F^t will be free of 1-, 0-, and dynamic hazards.

2. If a different form of the circuit is desired, manipulate F^t to the desired form by simple factoring, DeMorgan′s laws, etc. Treat each x_i and x_i' as independent variables to prevent introduction of hazards.

Simulation and Testing of Logic Circuits

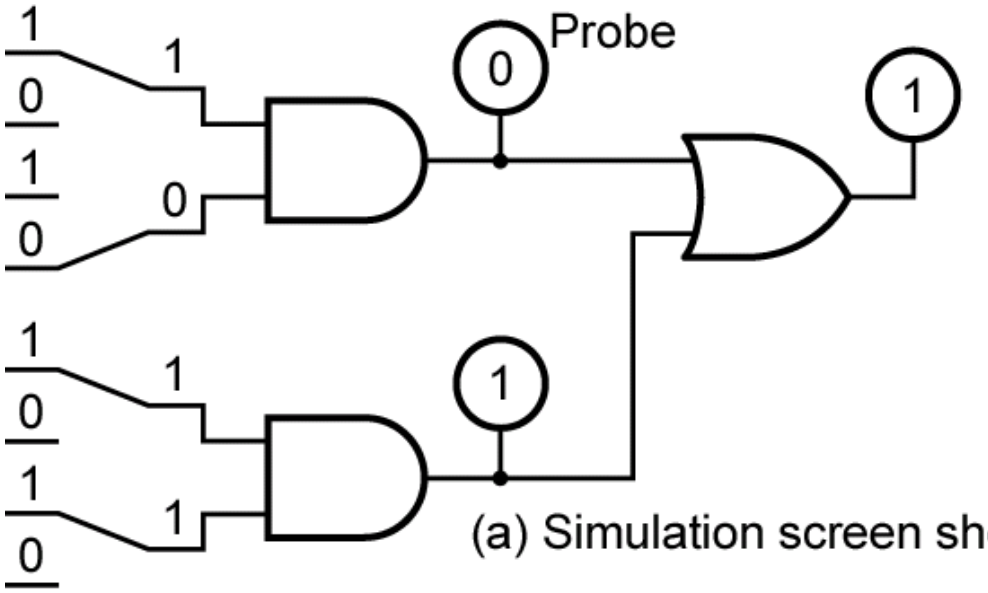
An important part of the logic design process is verifying that the final design is correct and debugging the design if necessary. Logic circuits may be tested either by actually building them or by simulating them on a computer.

Simulation is generally easier, faster, and more economical. As logic circuits become more and more complex, it is very important to simulate a design before actually building it.

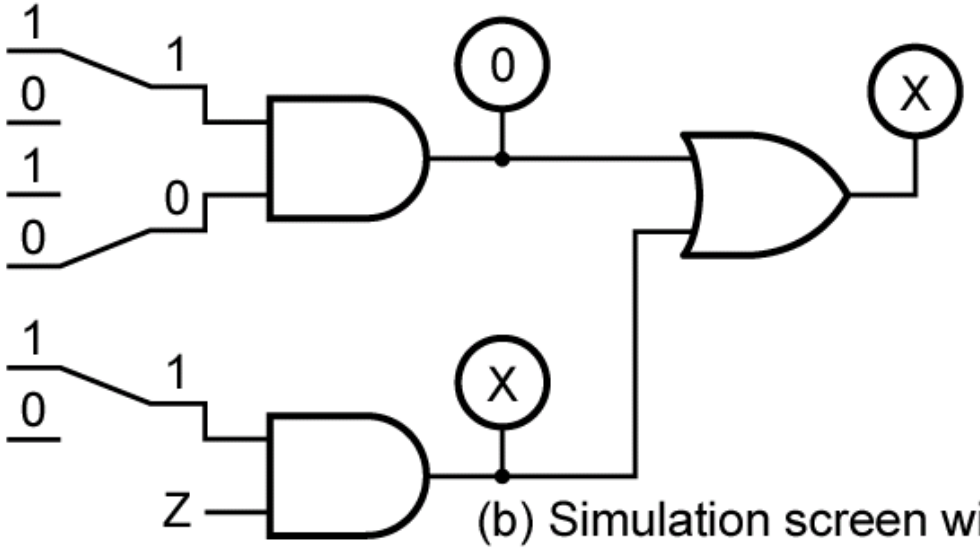
Section 8.5 (p. 229)

A simple simulator for combinational logic works as follows:

1. The circuit inputs are applied to the first set of gates in the circuit, and the outputs of those gates are calculated.
2. The outputs of the gates which changed in the previous step are fed into the next level of gate inputs. If the input to any gate has changed, then the output of that gate is calculated.
3. Step 2 is repeated until no more changes in gate inputs occur. The circuit is then in a steady-state condition, and the outputs may be read.
4. Steps 1 through 3 are repeated every time a circuit input changes.



(a) Simulation screen showing switches



(b) Simulation screen with missing gate input

The bottom gate has no connection to one of its inputs. Because that gate has a 1 input and a hi-Z input, we do not know what the hardware will do, and the gate output is unknown.

Figure 8-12

Four-Valued Logic Simulation

Definition of variables in four-valued logic simulation:

0: Logic Low

1: Logic High

X: Unknown

Z: High Impedance (also known as hi-Z, an open circuit)

Section 8.5 (p. 230)

•	0	1	X	Z
0	0	0	0	0
1	0	1	X	X
X	0	X	X	X
Z	0	X	X	X

+	0	1	X	Z
0	0	1	X	X
1	1	1	1	1
X	X	1	X	X
Z	X	1	X	X

Table 8-1.
AND and OR Functions for Four-Valued Simulation

Possible Causes for an Incorrect Output

If a circuit output is wrong for some set of input values, this may be due to several possible causes:

1. Incorrect design
2. Gates connected wrong
3. Wrong input signals to the circuit

If the circuit is built in lab, other possible causes include:

4. Defective gates
5. Defective connecting wires

Example

Suppose we build the following circuit in a simulator and set *A* = *B* = *C* = *D* = 1, the output *F* should be 0, but as seen in the diagram, it is 1.

How do we determine why the output is incorrect?

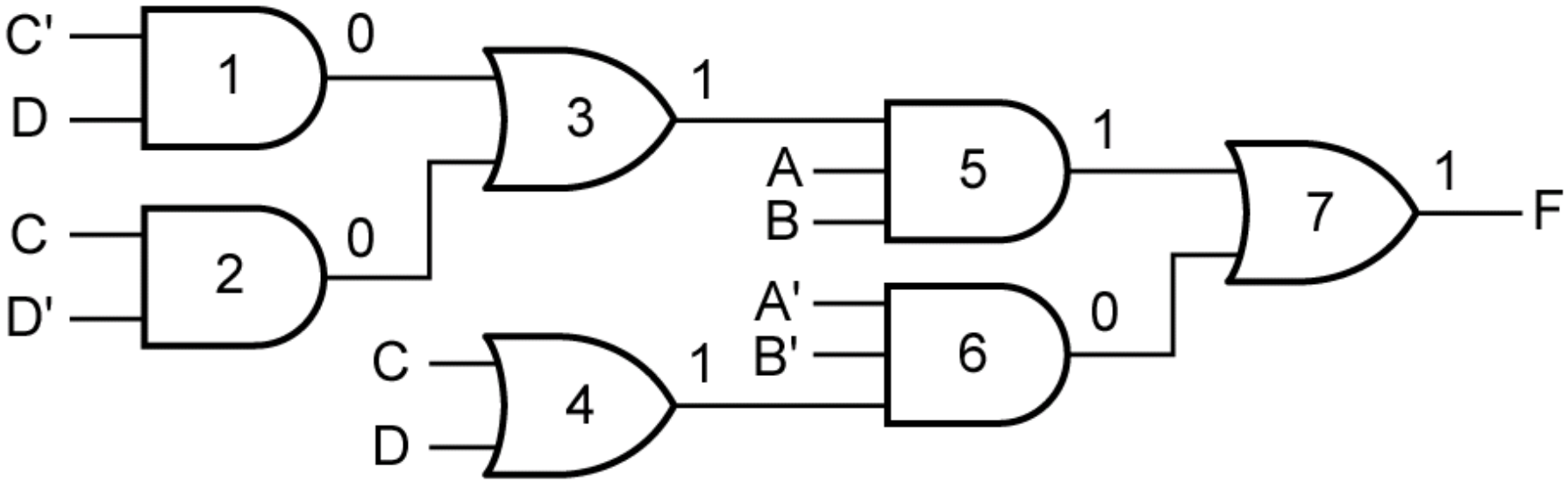


Figure 8-13: **Logic Circuit with Incorrect Output**

The reason for the incorrect value of *F* can be determined as follows:

1. The output of gate 7 (*F*) is wrong, but this wrong output is consistent with the inputs to gate 7, that is, 1 + 0 = 1. Therefore, one of the inputs to gate 7 must be wrong.

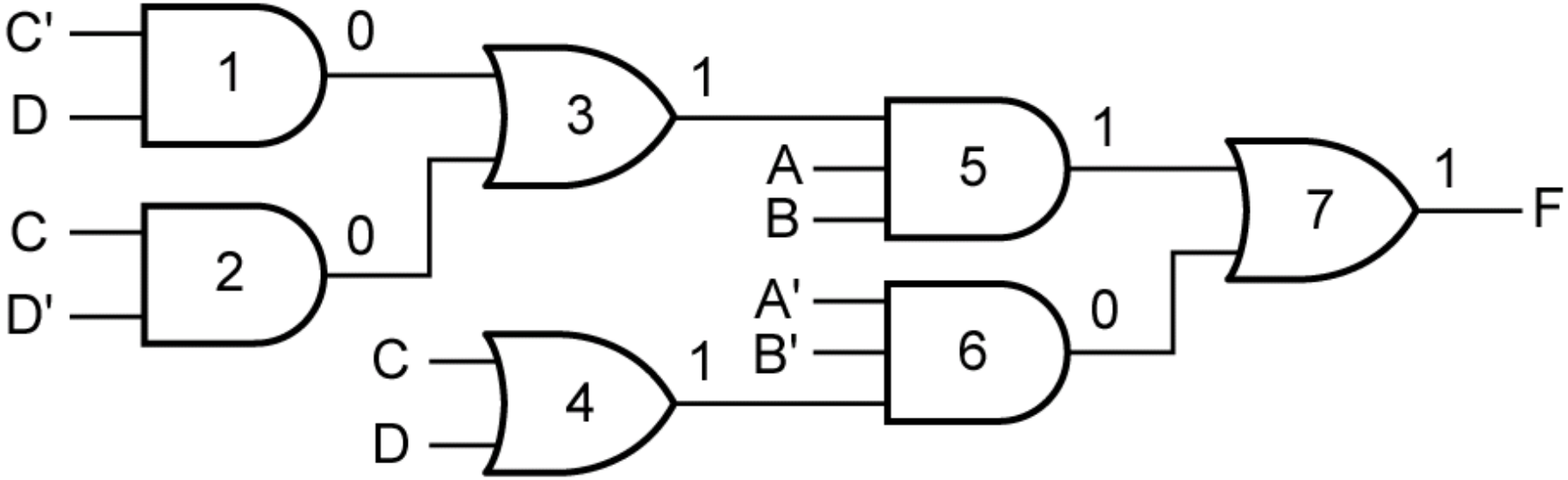


Figure 8-13: **Logic Circuit with Incorrect Output**

2. In order for gate 7 to have the correct output ($F = 0$), both inputs must be 0. Therefore, the output of gate 5 is wrong. However, the output of gate 5 is consistent with its inputs because $1 \cdot 1 \cdot 1 = 1$. Therefore, one of the inputs to gate 5 must be wrong.

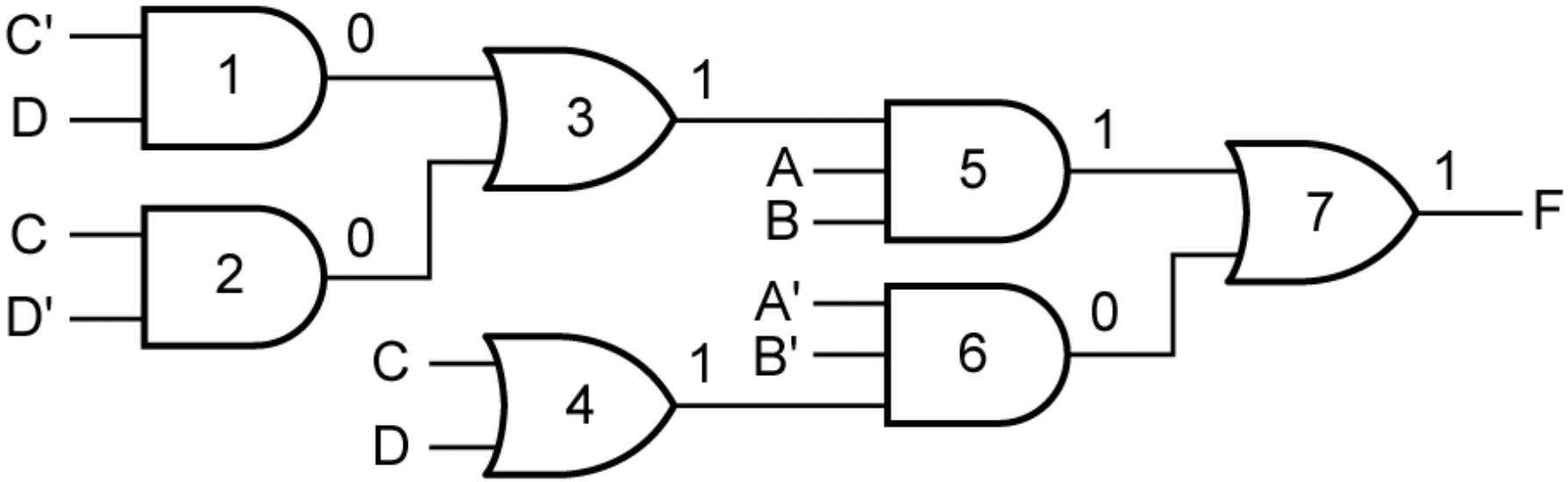


Figure 8-13: **Logic Circuit with Incorrect Output**

3. Either the output of gate 3 is wrong, or the *A* or *B* input to gate 5 is wrong. Because *C'D* + *CD'* = 0, the output of gate 3 is wrong.

4. The output of gate 3 is not consistent with the outputs of gates 1 and 2 because 0 + 0 ≠ 1. Therefore, either one of the inputs to gate 3 is connected wrong, gate 3 is defective, or one of the input connections to gate 3 is defective.

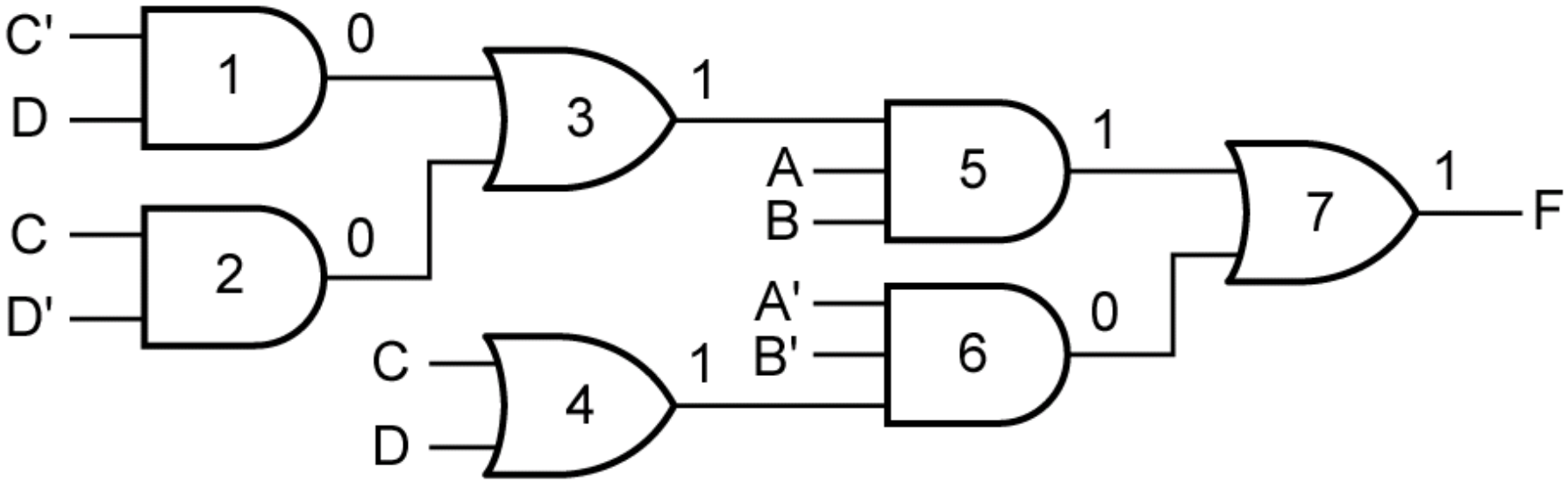


Figure 8-13: **Logic Circuit with Incorrect Output**

Seven-Segment Indicator

The seven-segment indicator can be used to display any one of the decimal digits 0 through 9. For example, “1” is displayed by lighting segments 2 and 3.

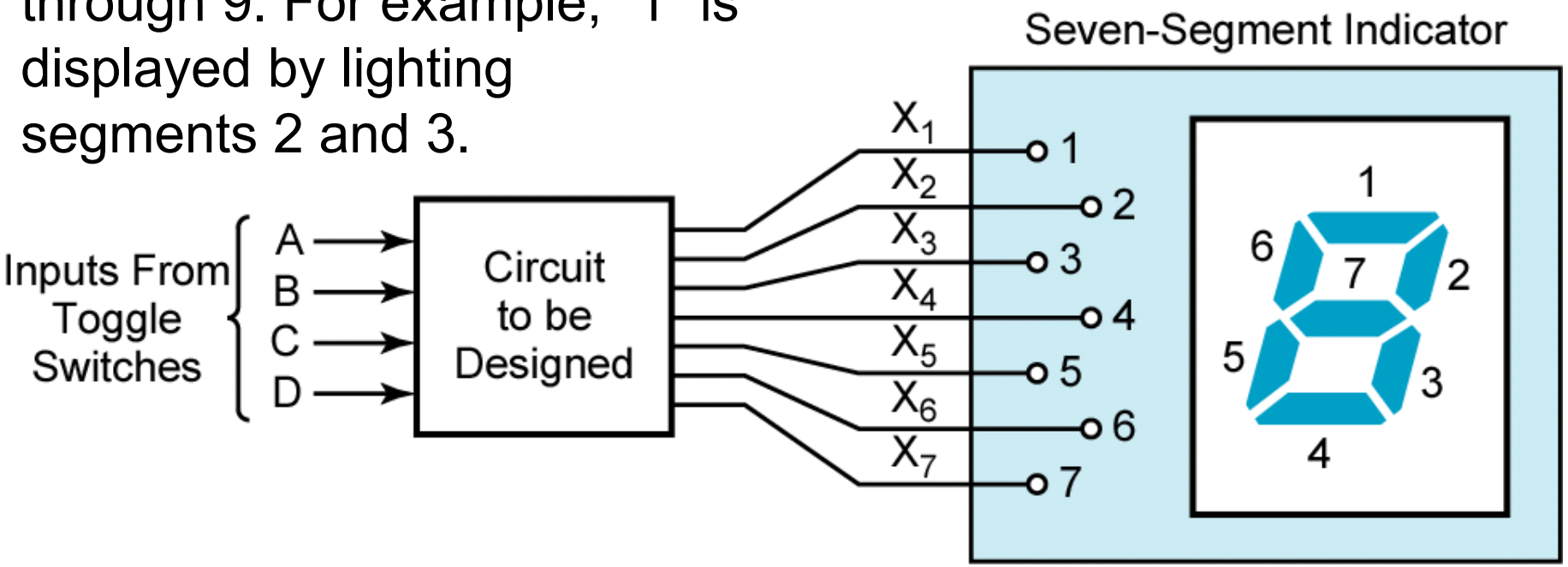


Figure 8.14: **Circuit Driving Seven-Segment Module**