Chapter 3

Introduction to Logic Circuit Analysis and Design

Chapter Outline

3.1 INTRODUCTION

|导读|

本节简要介绍本章的内容。

In this chapter, you will learn about integrated circuit devices, how to analyze and design logic circuits, and how to generate detailed schematic diagrams. You will learn how to manually design circuits in AND/OR form, OR/AND form, NAND/NAND form, and NOR/NOR form. All logic circuits have a delay time, so we will discuss how to determine the worst-case delay time through a circuit. Two very important logic devices—decoders and multiplexers—are introduced. Simple procedures are presented for manually designing digital circuits with decoders and also with multiplexers. Function hazards and logic hazards that generate glitches that can cause circuits to fail are covered. The design of most of the circuits is followed by a listing that shows the complete VHDL design of the circuits using Boolean equations.

3.2 INTEGRATED CIRCUIT DEVICES

|导读|

本节介绍集成电路器件的封装结构以及5种封装方式。

There are many manufacturers that provide physical hardware devices called **integrated circuits (ICs)** that are capable of carrying out two-valued Boolean functions. These devices can contain tens to literally millions of transistors on a small silicon semiconductor crystal called a die or chip. Because the circuitry contains mainly transistors, diodes, and resistors, which are all interconnected inside the chip, power consumption can be quite low and reliability quite high. The die is constructed and then welded to a frame as illustrated in Figure 3.1. Its input

and output leads are connected by thin gold wires to the package's leads or pins. The unit is encapsulated using glass, ceramic, or plastic. Finally, the unit is hermetically sealed. ICs that are hermetically sealed guard against die contamination in many different environments.

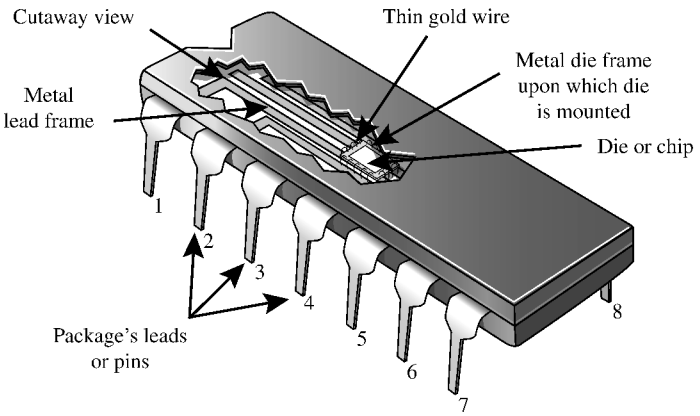


FIGURE 3.1 Cutaway view of an IC package showing the die or chip, the die frame, the gold wire, the lead frame, the package's leads or pins, and the pin numbers or pin outs

Five different types of integrated circuit packages are shown in Figure 3.2.

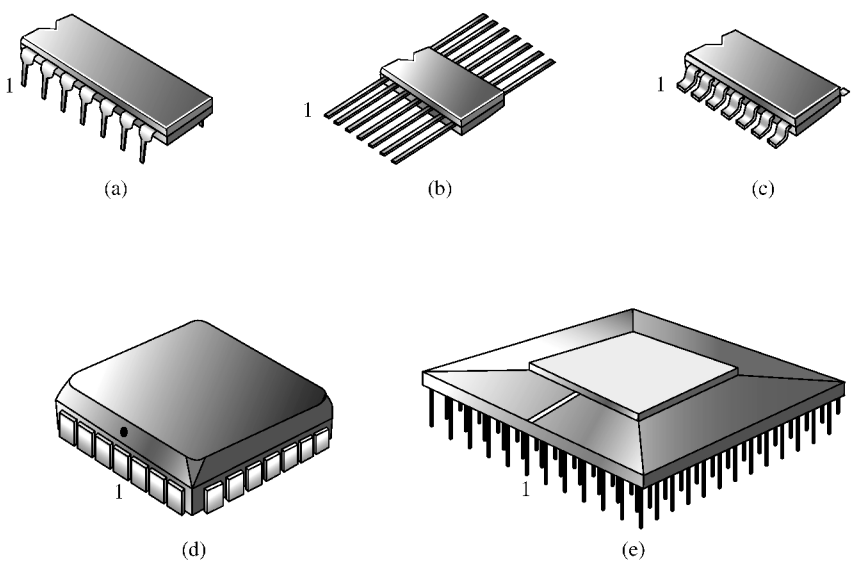


FIGURE 3.2 Packages for integrated circuits: (a) dual-in-line package; (b) flat package; (c) surface mounting package; (d) plastic leaded chip carrier package; (e) pin grid array package

集成电路：
Integrated Circuit(IC), 能够实现两值布尔函数的物理硬件器件。

The package shown in Figure 3.2a is the common **dual-in-line (DIP) package**. The packages shown in Figures 2b and c are the **flat package (flat pack)** and the **surface mount (small outline) package**. These packages are generally used in applications in which real estate on a **printed circuit board (PCB)** is critical and/or a lower cost must be achieved for high volume application. The packages shown in Figures 2d and e are the **plastic leaded chip carrier (PLCC) package** and the **pin grid array (PGA) package**, respectively, which are used for very large IC designs especially when the pin count—that is, the package inputs and outputs—for the designs become very large. Note the location of pin 1 for each package type. The integrated circuit packages shown in Figure 3.2 are only a few among many different types of packages available. For very large integrated circuit devices, a newer package is available called a **ball grid array (BGA) package** (the BGA package is not shown in Figure 3.2). The BGA has balls of solder on its pins that are soldered directly to a PC board. Manufacturers have a website that provides the data sheets for their parts. The data sheets provide a list of IC packages available, so engineers can choose the ones they prefer to use.

3.3 ANALYZING AND DESIGNING LOGIC CIRCUITS

We call **circuit analysis** the process of obtaining a Boolean function for a schematic or a circuit diagram. We call **circuit design** or **synthesis** the process of obtaining a schematic or a circuit diagram for a Boolean function. Keep in mind that the schematics or circuit diagrams we will cover are combinational or combinatorial logic circuits; that is, the outputs of these circuits depend only on the external inputs applied to the circuits. Combinational logic circuits do not have feedback (the outputs are never fed back as inputs), and they do not have memory capability.

|导读|

本节首先分析和设计继电器组合逻辑电路，然后分析和设计IC组合逻辑电路。

3.3.1 Analyzing and Designing Relay Logic Circuits

Before we introduce the analysis/design process using IC logic circuits, let's first analyze a logic switching circuit that uses relays. Switching circuits of this type are used in heavy power equipment. Figure 3.3a shows a physical representation for relay contacts that are normally closed (n.c.) and its corresponding symbol. Figure 3.3b shows a physical representation for relay contacts that are normally open (n.o.) and its corresponding symbol.

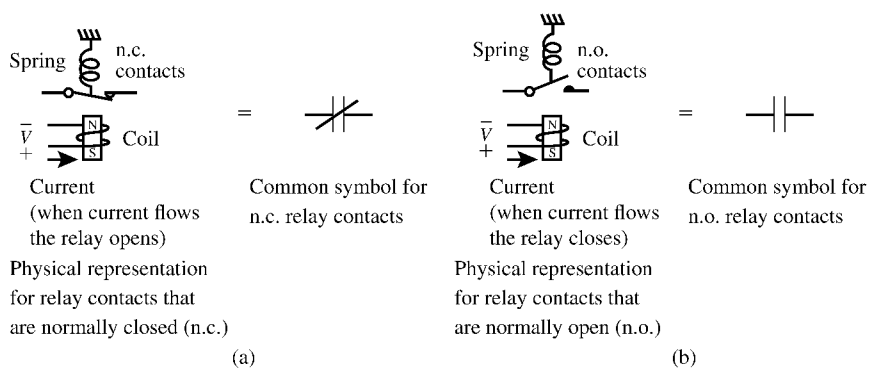


FIGURE 3.3 Relay contacts: (a) physical representation and common symbol for normally closed (n.c.) relay contacts and (b) physical representation and common symbol for normally open (n.o.) relay contacts

A logic switching circuit that uses relays connected up to a drive a motor is shown in Figure 3.4.

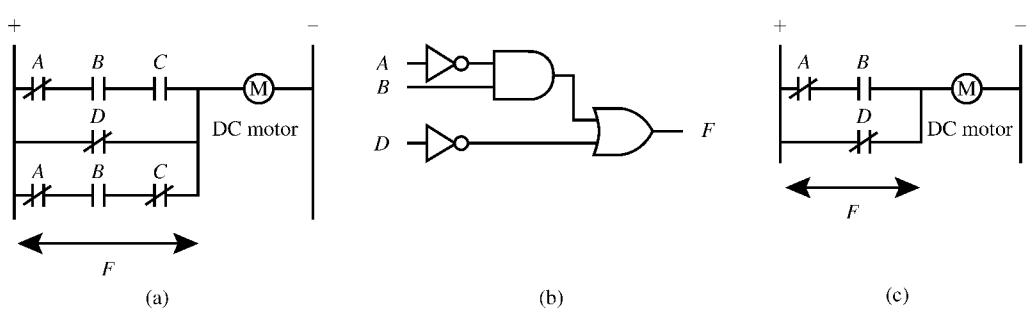


FIGURE 3.4 Relay logic switching circuits: (a) original relay circuit; (b) minimized symbolic logic circuit; (c) minimized relay circuit

Logic switching circuits used in power applications are generally referred to as **ladder logic circuits**. This term is used because it refers to the physical layout of the circuit (it looks like a ladder that may be climbed). In the switching circuit in Figure 3.4a, when F is 1, the motor M is turned on. Otherwise, the motor is turned off—that is, $F = 0$. To analyze relay logic circuits, one must remember that relays or switches connected in series provide an AND operation, while relays or switches connected in parallel provide an OR operation. The same series and parallel principles apply to transistor circuits, which you will study in a later course—that is, transistors connected in series provide an AND operation while transistors connected in parallel provide an OR operation.

Analyzing the circuit in Figure 3.4a, we obtain the following Boolean equation for the circuits: $F = \overline{A}\cdot B\cdot C + \overline{D} + A\cdot B\cdot\overline{C}$. The expression $\overline{A}\cdot B\cdot C$ indicates that signal A must be 0 AND signal B must be 1 AND signal C must be 1, so $F = 1$ to turn the motor on; OR the expression $\overline{D}$

indicates that the signal D must be 0, so $F = 1$ to turn the motor on; OR the expression $\overline{A}\cdot B\cdot\overline{C}$ indicates that the signal A must be 0 AND the signal B must be 1 AND the signal C must be 0, so $F = 1$ to turn the motor on.

Applying Boolean algebra or using a K-map, the function F can be reduced to the following minimum form: $F = \overline{A}\cdot B + \overline{D}$. Using the minimum form of the function, we can design the circuit in symbolic form as shown in Figure 3.4b. The relay logic circuit drawn for the minimum form of the function is shown in Figure 3.4c. In general, logic circuits that use ICs may not be hefty enough to drive high-powered motors or electric light bulbs; however, logic circuits that use relays can be used for these applications.

电路分析：
Circuit Analysis,从原理图或者电路图中获得布尔函数的过程。

To manually design a logic circuit using relays as illustrated in the last example, first obtain a minimum Boolean equation for the function. Next, substitute the appropriate relay type (either normally open or normally closed) for each input, and connect each relay either in series or in parallel depending on the form of the function. Connect the resulting relay circuit in series with an output device such as a motor, a lamp, or a control relay (CR). A control relay is represented by CR with a circle around it and signifies the coil of a relay. When a control relay is energized, its associated contacts close, thus allowing the CR to control another circuit. The voltage for a ladder logic circuit is applied across the vertical lines labeled + and −.

3.3.2 Analyzing IC Logic Circuits

Now let's analyze the symbolic logic circuits shown in Figure 3.5.

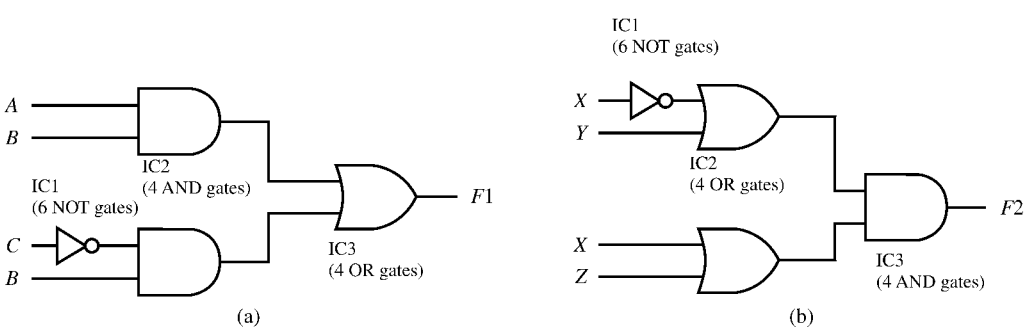


FIGURE 3.5 Circuits to be analyzed to obtain their Boolean functions

The circuit in Figure 3.5a is drawn in AND/OR form because AND gates are feeding into an OR gate. The circuit in Figure 3.5b is drawn in OR/AND form because OR gates are feeding into an AND gate. To analyze these circuits, we need to obtain the Boolean functions for $F1$ and $F2$. These circuits could represent relay logic circuits, or they could represent logic circuits constructed with ICs. In this section, we will assume that the circuits are constructed with ICs.

In Figure 3.5a, the function $F1$ is written by obtaining the outputs of IC2 (an AND gate with output $A\cdot B$) and IC2 (another AND gate with output $\overline{C}\cdot B$) and then obtaining the output of IC3 (an OR gate with output $F1 = A\cdot B + C\cdot B$ or $\overline{F1} = (\overline{A} + \overline{B})\cdot(C + \overline{B})$ via DeMorgan's Theorem). The function $F1 = A\cdot B + \overline{C}\cdot B$ is written in SOP form, while the function $\overline{F1} = (\overline{A} + \overline{B})\cdot(C + \overline{B})$ is written in POS form.

In Figure 3.5b, the function $F2$ is written by obtaining the outputs of IC2 (an OR gate with output $\overline{X} + Y$) and IC2 (another OR gate with output $X + Z$) and then obtaining the output of IC3 (an AND gate with output $F2 = (\overline{X} + Y)\cdot(X + Z)$ or $\overline{F2} = X\cdot\overline{Y} + \overline{X}\cdot\overline{Z}$ via DeMorgan's Theorem). The function $F2 = (\overline{X} + Y)\cdot(X + Z)$ is written in POS (product of sums) form, while the function $\overline{F2} = X\cdot\overline{Y} + \overline{X}\cdot\overline{Z}$ is written in SOP (sum of products) form.

As you can see from these examples, the analysis of small IC (or relay) logic circuits is fairly simple. All one needs to do is to write the output function of the circuit in terms of the input variables.

In practice, an IC number is assigned to each IC package on a printed circuit board (PCB). An IC package can contain more than one gate, as shown in Figure 3.5. **Small-scale integration (SSI)** packages are ICs packages that contain only a few gates. For example, each package may contain six Inverters or NOT gates, four 2-input OR gates, and four 2-input AND gates, respectively. An IC number is also assigned to large-scale integration packages such as a **complex programmable logic device (CPLD)** or a **field programmable gate array (FPGA)**. CPLDs contain hundreds to thousands of gates, while FPGAs contain thousands to millions of gates.

电路设计或综合：
Circuit Design or Synthesis,从布尔函数获得原理图或者电路图的过程。

3.3.3 Designing IC Logic Circuits

When you manually design a circuit for a Boolean function, it is a good idea to first reduce the Boolean function. The easiest way to manually design an IC logic circuit is write the Boolean function in reduced SOP form and then draw the circuit in AND/OR form, or to write the Boolean function in reduced POS form and then draw the circuit in OR/AND form. The following four steps may help you draw the circuit:

Step 1: Draw the AND and OR gates for the reduced Boolean function.
Step 2: Show all signals as noncomplemented signal names.
Step 3: Interconnect the gates, including NOT gates where necessary.
Step 4: Cleanup or reduce the number of NOT gates.

Figure 3.6 shows the manual design of an IC logic circuit for the reduced SOP function $F1 = \overline{A}\cdot B\cdot\overline{C} + \overline{B}\cdot C + A\cdot\overline{B}$.

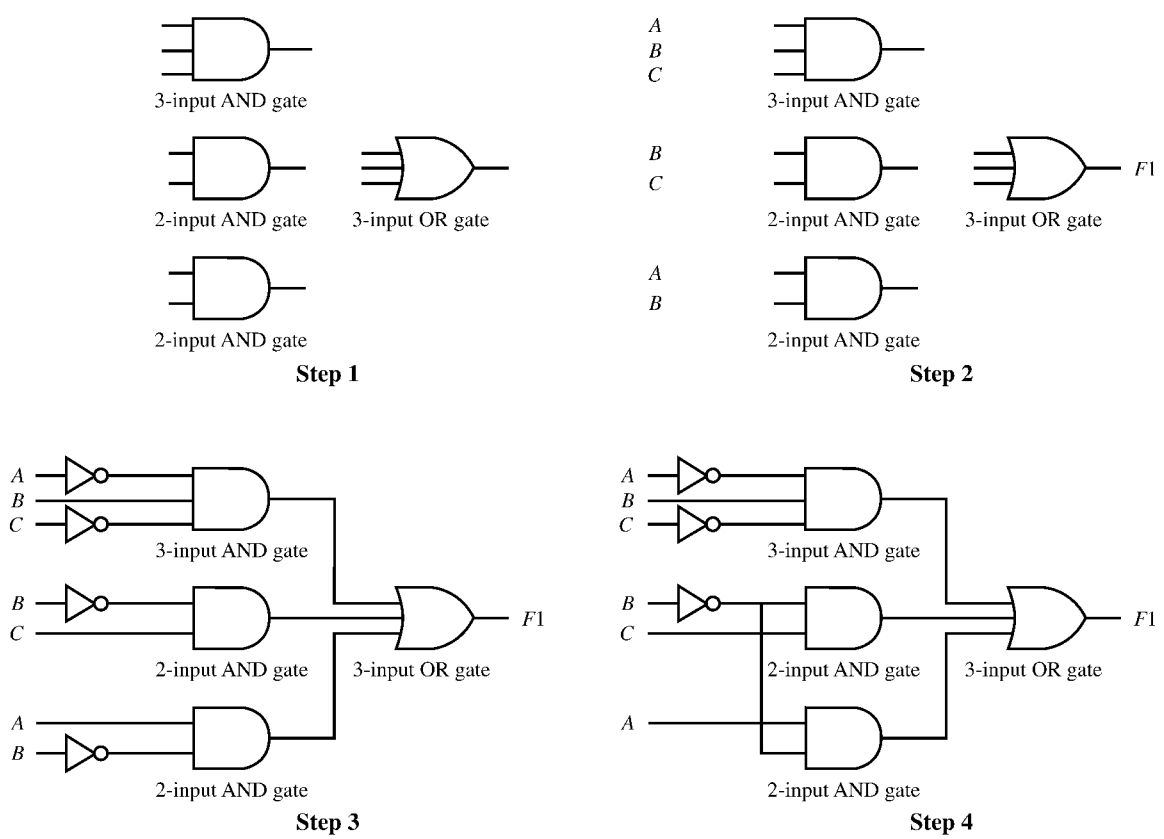


FIGURE 3.6 Manual design of an IC logic circuit for a reduced SOP function

Step 1: The AND and OR gates are drawn for the reduced function.
Step 2: All the input and output signals are shown as noncomplemented signal names.
Step 3: The gates are interconnected, including NOT gates where they were necessary.
Step 4: Cleanup was performed to remove one NOT gate.

组合逻辑电路：Combinational Logic Circuit,这些电路的输出仅依赖于应用在该电路的外部输入。组合逻辑电路不存在反馈(输出永不会被反馈作为输入),而且它们没有记忆能力。

Each IC for a particular logic family such as TTL (transistor transistor logic) and CMOS (complementary metal-oxide semiconductor) has a **fan-out**. The fan-out is the maximum number of inputs to which the IC output can be connected without electrically loading down the output. As long as the fan-out is not exceeded, the IC will function properly. The fan-out for the low-power Schottky (LS) TTL family is 20 (or 20 inputs). The original or standard TTL family has a fan-out of only 10 (or 10 inputs). Another name you should know is **fan-in**. Fan-in is the name used to describe the number of gate inputs—that is, the number of inputs that a gate has.

A **signal line** is a line drawn to an input line of a gate symbol, or a line drawn from an output line of a gate symbol. A **net** is the name used to describe signal lines that are connected together to carry the same signal. In Figure 3.6, signal lines with the same signal names are considered to be connected—that is, they belong to the same net. Be careful to label each signal line with only one name. If you were to wire up the circuit in Figure 3.6 in the laboratory, you would need to connect all signal lines together that have the same name.

In Figure 3.7, we show two alternate solutions for function $F1 = \overline{A}\cdot B\cdot\overline{C} + \overline{B}\cdot C + A\cdot\overline{B}$. Both solutions use a connection scheme for the input signal lines called a vertical-input scheme. Circuits drawn using a vertical-input scheme are usually more organized. With this scheme, one can draw large circuit designs in AND/OR form or OR/AND form quite rapidly. The vertical-input scheme eliminates Step 4 (cleanup) to reduce the number of NOT gates.

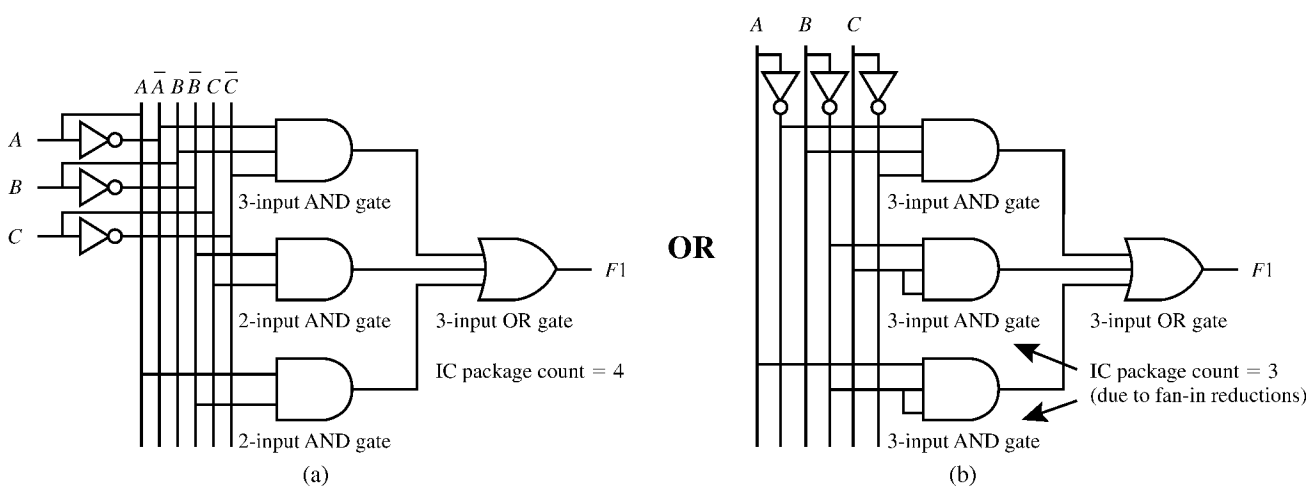


FIGURE 3.7 IC logic circuit designs for a minimum SOP form of a function using a vertical-input scheme

Notice in Figure 3.7a that the IC package count is four, but in Figure 3.7b the IC package count is only three, because a single IC package of three 3-input AND gates is used in the same package. After obtaining a circuit design for a minimum function, it is sometimes possible to build a smaller implementation of the circuit by using fewer ICs. This is done by connecting unused gate inputs to used gate inputs to reduce the fan-in of a gate, as shown in Figure 3.7b. This technique may be referred to as **fan-in reduction**. By using fan-in reduction, the 2-input AND gate IC package is not required.

Manually designing logic circuits is somewhat tedious. Using a hardware description language such as VHDL is a more efficient way to design logic circuits. Listing 3.1 shows a complete VHDL design for the function $F1 = \overline{A}\cdot B\cdot\overline{C} + \overline{B}\cdot C + A\cdot\overline{B}$.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb1 is port (
    A, B, C : in std_logic;
    F1 : out std_logic
    );
end comb1;

architecture Boolean_function of comb1 is
begin
    F1 <= (not A and B and not C) or (not B and C) or (A and not B);
end Boolean_function;
```

LISTING 3.1 Complete VHDL design for the function $F1 = \overline{A}{\cdot}B{\cdot}\overline{C} + \overline{B}{\cdot}C + A{\cdot}\overline{B}$ (project: comb1)

For manual designs, OR gates with more than two inputs for some off-the-shelf logic families (such as TTL) are not available in ICs. This may pose a problem with the circuit designs in Figure 3.6 and Figure 3.7, which required a 3-input OR gate. Connecting (cascading) two 2-input OR gates in series to obtain a 3-input OR gate easily solves this problem as shown in Figure 3.8a.

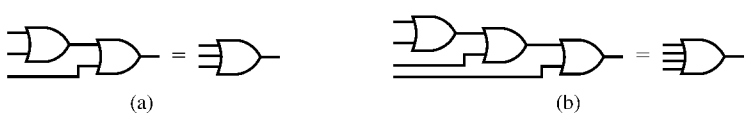


FIGURE 3.8 Cascading IC gates (a) 3-input OR gate (b) 4-input OR gate

Connecting three 2-input OR gates in series provides us with a 4-input OR gate as shown in Figure 3.8b. This solution also has a problem. The resulting cascaded circuit provides an output that responds more slowly to input changes as the number of cascaded stages is increased. We will consider this phenomenon a little later.

Consider the function $F2$ written in a minimum POS form as $F2 = (\overline{X} + Y){\cdot}(X + \overline{Y}){\cdot}(X + \overline{Z})$. A circuit design for this function is shown in Figure 3.9 using a vertical-input scheme.

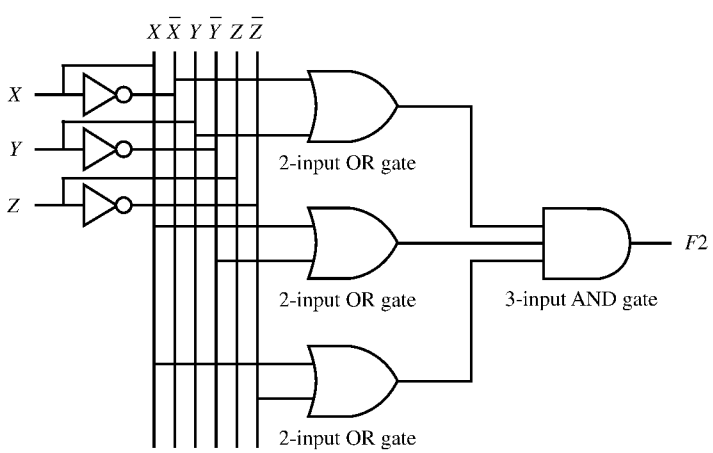


FIGURE 3.9 IC logic circuit design for a minimum POS form of a function using a vertical-input scheme

The main difference between the design of an SOP form (AND/OR form) of circuit and a POS form (OR/AND form) of circuit is the placement of the AND and the OR gates. For an SOP form of circuit, the AND gates feed into an OR gate—hence the name AND/OR form. For a POS form of circuit, the OR gates feed into an AND gate—hence the name OR/AND form.

Here is a more efficient way to design a logic circuit for function $F2$. Listing 3.2 shows a complete VHDL design for the function $F2 = (\overline{X} + Y)\cdot(X + \overline{Y})\cdot(X + \overline{Z})$.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb2 is port (
    X, Y, Z : in std_logic;
    F2 : out std_logic
    );
end comb2;

architecture Boolean_function of comb2 is
begin
    F2 <= (not X or Y) and (X or not Y) and (X or not Z);
end Boolean_function;
```

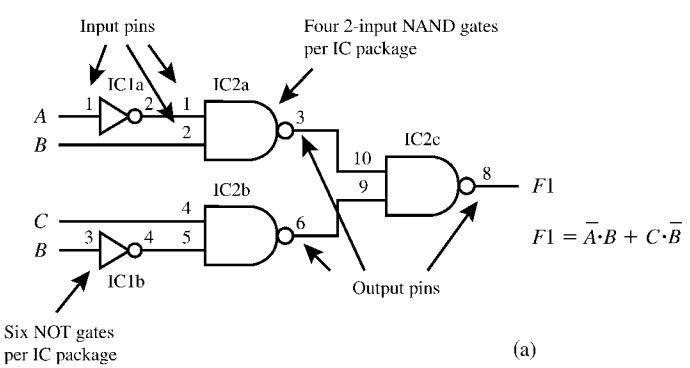
LISTING 3.2 Complete VHDL design for the function $F2 = (\overline{X} + Y)\cdot(X + \overline{Y})\cdot(X + \overline{Z})$ (project: comb2)

3.4 GENERATING DETAILED SCHEMATICS

|导读|

本节介绍如何生成详细的原理图。最后还给出函数 $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ 和 $F2 = A \oplus B$ 的完整VHDL设计以及其仿真波形。

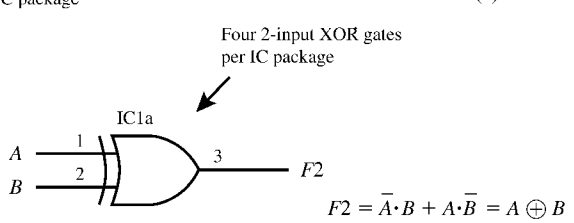
All the circuits that we have drawn up to now are functional logic diagrams; that is, they are functionally correct but lack the details necessary to show the actual IC connections (or wiring) required to build a circuit on a PC board or in the lab. The circuit shown in Figure 3.10a is an example of a **detailed schematic** for the function $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ using off-the-shelf advanced CMOS (complementary metal-oxide semiconductor) devices. The circuit shown in Figure 3.10a is in NAND/NAND form, which will be covered in the next section. Datasheets for Texas Instruments logic devices are available online at http://ti.com. When the ti window opens, select Logic, User Guides, GO; then click on Logic. Choose a logic family such as AC (Advanced CMOS); and select GO. When all the devices for that family appear, click on the device you want, then click on the Datasheet Icon. The datasheet provides the input pins, output pins, and power pins (V_{CC} and GND) for the package.



(a)

Reference designator	Part value	Pin number	
		V_{CC}*	GND
IC1	CD74AC04	14	7
IC2	CD74AC00	14	7

* + 5V



(b)

Reference designator	Part value	Pin number	
		V_{CC}*	GND
IC1	CD74AC86	14	7

* + 5V

FIGURE 3.10 Detailed schematics: (a) circuit using multiple IC devices; (b) circuit using a single gate in an IC device

In Figure 3.10b, we show a detailed schematic for the function $F2 = \overline{A}\cdot B + A\cdot\overline{B} = A \oplus B$. This IC has four XOR gates in the same package, and we are only using one. The following important items are necessary when drawing a detailed schematic:

1. Identify the part number for each IC in the circuit.
2. Show the pin numbers (also referred to as pin assignments) for all ICs in the circuit.
3. Show the power connections (V_{CC} and GND) for all the ICs in the circuit.

Many companies require their designers to provide detailed schematics so that an accurate record can be kept for each design. In addition, they require a written record to explain how the circuit works. This information is archived by companies so that they can keep complete and accurate records of their designs.

Listing 3.3 shows a complete VHDL design for the functions $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ and $F2 = A \oplus B$.

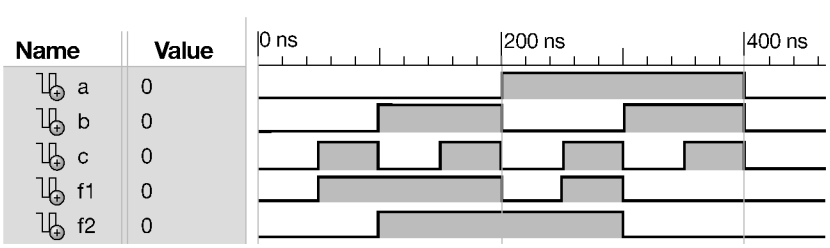
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb3 is port (
     A, B, C : in std_logic;
     F1, F2 : out std_logic
     );
end comb3;

architecture Boolean_funtions of comb3 is
begin
     F1 <= (not A and B) or (C and not B);
     F2 <= A xor B;
end Boolean_funtions;
```

LISTING 3.3 Complete VHDL design for the functions $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ and $F2 = A \oplus B$ (project: comb3)

Waveform 3.1 shows waveform diagrams for the VHDL design for the Boolean functions $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ and $F2 = A \oplus B$.



WAVEFORM 3.1 Waveform diagrams for the VHDL design for the Boolean functions $F1 = \overline{A}\cdot B + C\cdot\overline{B}$ and $F2 = A \oplus B$

Note that function $F1$ is 1 when A is 0 and B is 1, $F1$ is also 1 when C is 1 and B is 0, and $F1$ is 0 for all other combinations of A, B, and C. Note that the function $F2$ is only 1 when A is not equal to B otherwise $F2$ is 0. The simulation shows that the VHDL design functions correctly.

One of the nice things about using a hardware description language such as VHDL is that you do not have to draw detailed schematics to obtain a circuit on a system board such as the BASYS 2 board or the NEXYS 2 board, because the circuits are connected up internally via the bit pattern that is generated by the software. Only the external pin connections for each of the signals in the entity have to be declared. None of the pin connections for power have to be declared. Just apply power to the system board via the USB connector to the computer.

3.5 DESIGNING CIRCUITS IN NAND/NAND AND NOR/NOR FORM

|导读|

本节介绍用全功能门(即与非和或非)来设计与非/与非以及或非/或非形式的电路。

In manual designs using discrete (or separate) IC devices, NAND and NOR gates are preferable to AND and OR gates. There are three reasons this is true: (1) NAND gates are generally faster than AND gates, and NOR gate are generally faster than OR gates in the same logic family; (2) NAND gates and NOR gates are available with a larger variety of fan-ins (gate inputs) to choose from than AND gates and OR gates; and (3) fewer IC packages are required to design circuits that use NAND gates and NOR gates because they are functionally complete gates, as discussed in Chapter 1.

A procedure for manually designing a logic circuit in NAND/NAND form is shown in Figure 3.11. First, design the circuit in AND/OR form, and then convert the circuit into NAND/NAND form, as shown in Figure 3.11 using the **graphical design method for NAND/NAND form**.

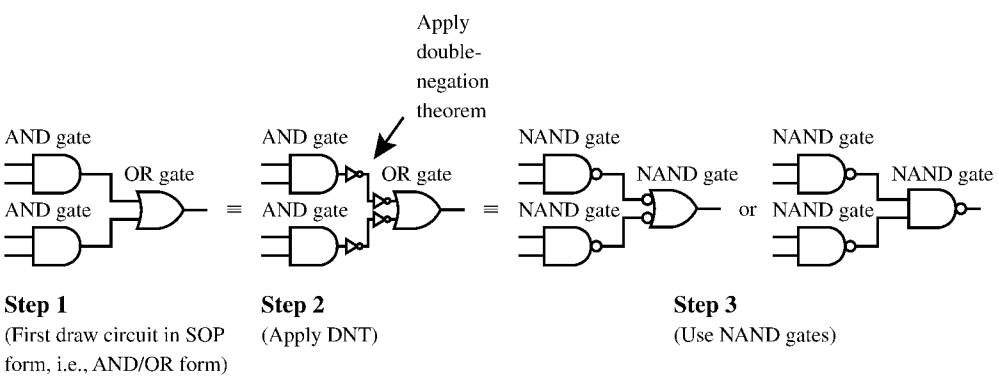


FIGURE 3.11
Graphical Design Method for NAND/NAND form

A procedure for manually designing a logic circuit in NOR/NOR form is shown in Figure 3.12. First, design the circuit in OR/AND form, and then convert the circuit into NOR/NOR form, as shown in Figure 3.12 using the **graphical design method for NOR/NOR form**.

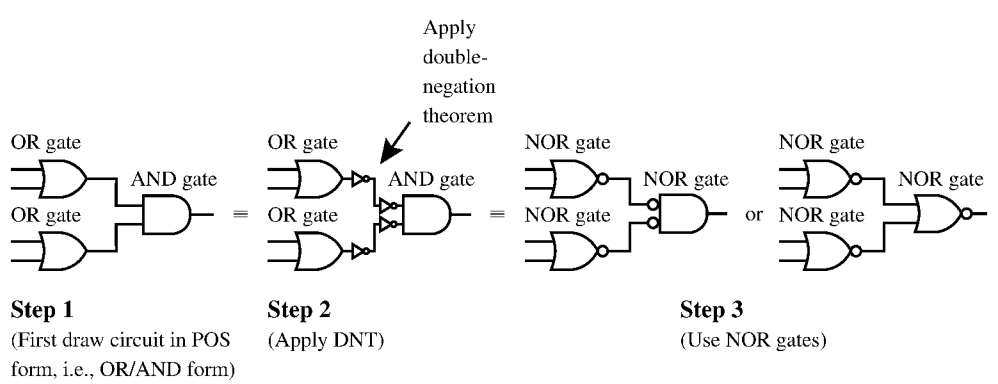


FIGURE 3.12
Graphical design method for NOR/NOR form

Consider the manual design of a circuit to implement the reduced Boolean function $F1 = \overline{A}\cdot\overline{B}\cdot\overline{C} + B\cdot C + A\cdot C$ in NAND/NAND form using the graphical design method. Since the function is already expressed in SOP form, we just have to draw the circuit in AND/OR form then convert the circuit to NAND/NAND form as shown in Figure 3.13.

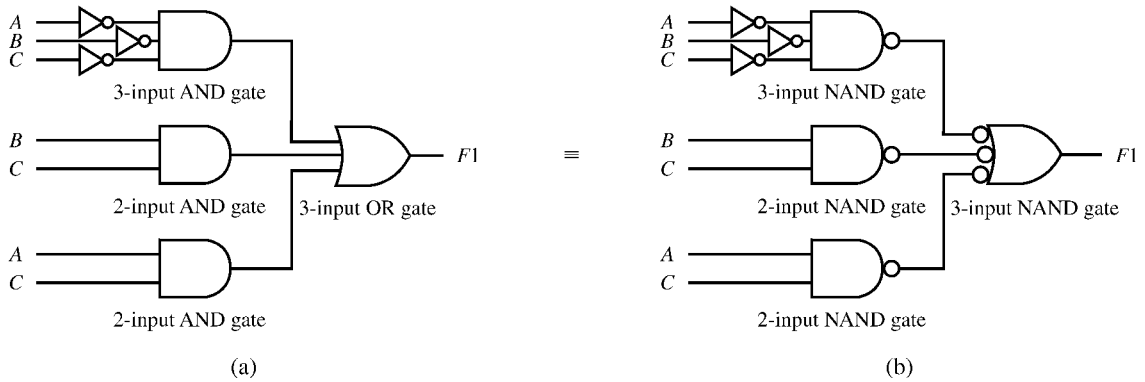


FIGURE 3.13 Converting a circuit from AND/OR form to NAND/NAND form

The application of the double-negation theorem (DNT) is not shown in Figure 3.13 because this can be done mentally without drawing all the NOT gate pairs.

Listing 3.4 shows a complete VHDL design for the function $F1 = \overline{A}\cdot\overline{B}\cdot\overline{C} + B\cdot C + A\cdot C$.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb4 is port (
    A, B, C : in std_logic;
    F1 : out std_logic
    );
end comb4;

architecture Boolean_function of comb4 is
begin
    F1 <= (not A and not B and not C) or (B and C) or (A and C);
end Boolean_function;
```

LISTING 3.4 Complete VHDL design for the function $F1 = \overline{A}\cdot\overline{B}\cdot\overline{C} + B\cdot C + A\cdot C$ (project: comb4)

Consider the manual design of a circuit to implement the reduced Boolean function $F2 = \overline{X}\cdot\overline{Z} + X\cdot Z + Y$ in NOR/NOR form using the graphical design method. To follow the graphical design method, we must first express the function $F2$ in POS form, which is $\overline{F2} = (X + Z)\cdot(\overline{X} + \overline{Z})\cdot\overline{Y}$. Now the circuit must be drawn in OR/AND form and then converted to NOR/NOR form as shown in Figure 3.14.

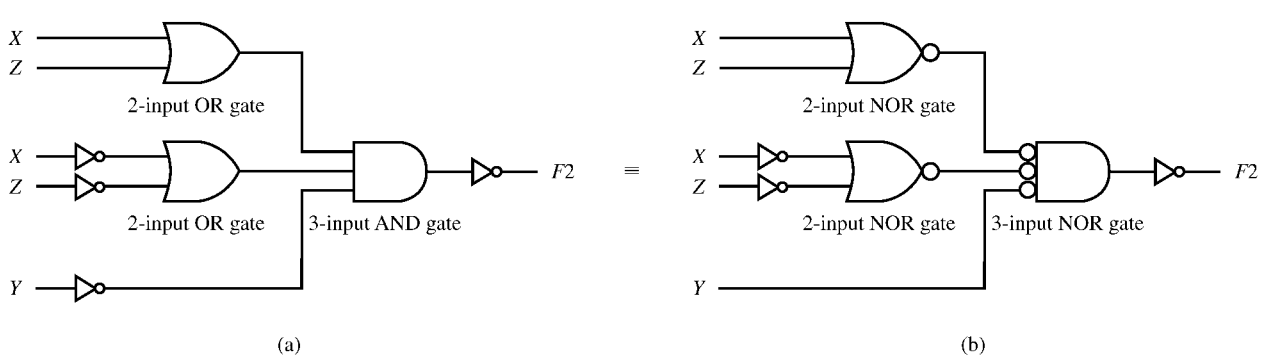


FIGURE 3.14 Converting a circuit from OR/AND form to NOR/NOR form

The application of the DNT is not shown in Figure 3.14 because this can be done mentally without drawing all the NOT gate pairs.

Listing 3.5 shows a complete VHDL design for the function $\overline{F2} = (X + Z)\cdot(\overline{X} + \overline{Z})\cdot\overline{Y}$.

LISTING 3.5
Complete VHDL design for the function $\overline{F2} = (X + Z)\cdot(\overline{X} + \overline{Z})\cdot\overline{Y}$ (project: comb5)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb5 is po.5rt (
    X, Y, Z : in std_logic;
    F2 : out std_logic
    );
end comb5;

architecture Boolean_function of comb5 is
begin
    F2 <= not ((X or Z) and (not X or not Z) and not Y);
              --POS form for F2
    --F2 <= (not X and not Z) or (X and Z) or Y;
              --This is an alternate description for F2
              --i.e., an SOP form for F2
end Boolean_function;
```

Reminder: A comment may be placed in VHDL code by using two hyphens in series, that is, --, as shown in Listing 3.5.

3.6 PROPAGATION DELAY TIME

|导读|

本节介绍组合电路中存在的传输延时，以及延时的累加特性所导致的最坏延时路径。

Circuit delays are caused by signals passing through the components that make up the circuit. Worst-case delay is caused by a signal passing through the slowest delay path in the circuit. Each wire (or connector) and each gate has a **propagation delay time**, which is the time it takes a signal applied at its input to travel from the input to the output. The propagation delay time of a wire is dependent on its length and its cross-sectional area. It is also dependent on the material that the wire is made out of, such as copper, silver, and gold. Gold has the best conductivity, followed by silver and then copper. In terms of just the physical dimensions, a longer wire has a longer propagation delay time than a shorter wire, and a wire with a smaller cross-sectional area has a longer propagation delay time than a wire with a larger cross-sectional area. Remember that even wires have a propagation delay time that usually cannot be ignored in a circuit, if the circuit is operated at a very high frequency.

The abbreviation t_p is used as a relative measure of the time it takes for a signal to propagate through a gate. Figure 3.15 shows a NOT gate, an AND gate, and an OR gate with their symbols, their Boolean functions, and their propagation delay times. The propagation delay times are generally different because they are all different circuits. The propagation delay times for single-gate circuits is in the order of only a few nanoseconds. A nanosecond is one billionth of a second (or 10^{-9} seconds) and is the time it takes for electricity to travel through a length of wire approximately 1 foot or about $\frac{1}{3}$ meter.

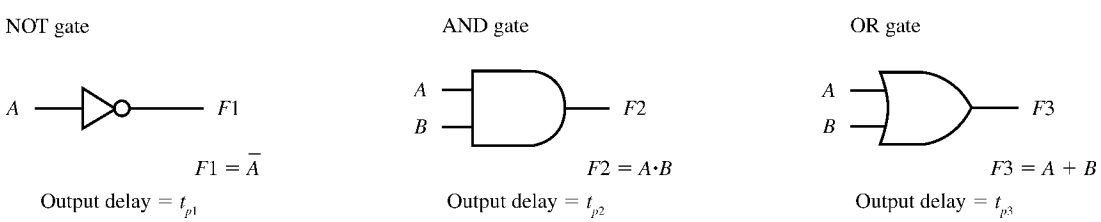
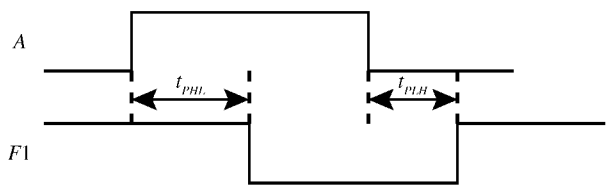


FIGURE 3.15 A NOT gate, an AND gate, and an OR gate with their symbols, their Boolean functions, and their propagation delay times

The propagation delay time, t_p, is the average of t_{PLH} and t_{PHL}, which are specified in the data sheets for the ICs. The propagation delay time (high-to-low-level output, or t_{PHL}) is the delay time through a gate when the output changes from a high (H) value to a low (L) value. The propagation delay time (low-to-high-level output, or t_{PLH}) is the delay time through a gate when the output changes from a low (L) value to a high (H) value. Waveform 3.2 shows a waveform diagram that illustrates both t_{PHL} and t_{PLH} for the NOT gate in Figure 3.15.



WAVEFORM 3.2 Waveform diagram illustrating both t_{PLH} and t_{PHL} for the NOT gate in Figure 3.15

最坏延时：
Worst-case delay,由于信号通过电路中最慢的延时路径而导致的传输延时。

Delays add up. For example, three similar NOT gates connected in cascade or in series (where one output feeds into the next) cause a propagation delay time of three times the propagation delay time of one of the NOT gates, or $3t_{p1}$. Sometimes NOT gates are used to slow down or delay a signal through a circuit.

Delays also add up for all gate types, including NOT gates, AND gates, and OR gates as shown in Figure 3.16 for the function $F4 = A \cdot \overline{C} + \overline{A} \cdot C + \overline{B}$.

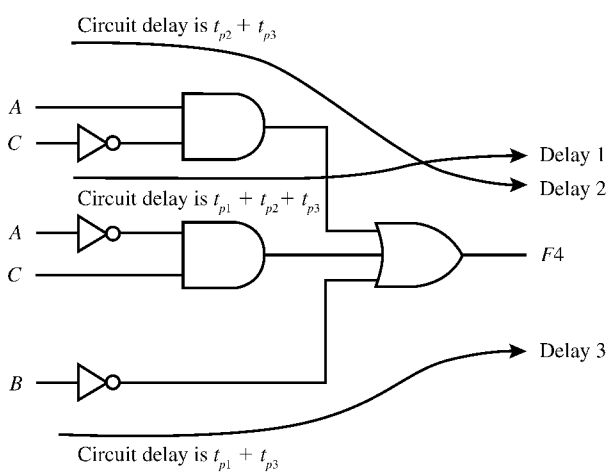


FIGURE 3.16 Worst-case delay time through the circuit

传输延时：
Propagation delay time,电路将一个作用于其输入的信号从输入传递到输出所花费的时间。

The worst-case delay time through the circuit in Figure 3.16 is the path from the input to the output that has the longest delay time—that is, delay 1 in the circuit in Figure 3.16. The delay times through the wires in the circuit in Figure 3.16 were ignored, but they would increase the overall delay time through the circuit slightly. So,

$$\text{Worst-case delay time} = t_{p1} + t_{p2} + t_{p3} = t_{pNOT\ gate} + t_{pAND\ gate} + t_{pOR\ gate}$$

In general, faster circuits have shorter delay times. Also, faster circuits have the fewest number of cascaded components from the input to the output of the circuit.

3.7 DECODERS

Figure 3.17a shows a very useful circuit called a **decoder** that utilizes NOT gates and AND gates. A circuit that converts a binary code applied to n input lines to one of 2^n different output lines is called an n-to-2^n line decoder. A decoder with n input lines can convert 2^n different binary codes applied to its input lines into 2^n mutually exclusive outputs. Each code applied to

|导读|
本节介绍常见组合逻辑电路之一

的译码器的定义、逻辑电路和真值表，并给出2-4线译码器的VHDL设计和仿真结果，以及3-8线译码器的VHDL设计。最后，阐述用译码器和单个门设计组合逻辑电路的方法。

its input is converted to a corresponding single bit on the output. Figure 3.17a shows the circuit diagram for a 2-line to 4-line decoder, which we will just call a 2-to-4 decoder.

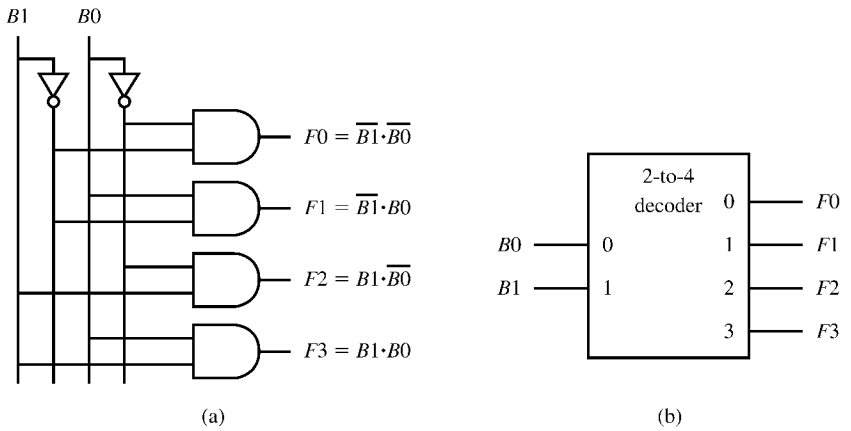


FIGURE 3.17 2-to-4 decoder: (a) discrete IC circuit diagram; (b) logic symbol

Figure 3.17b shows a logic symbol for a 2-to-4 Decoder. A decoder can also be thought of as a minterm generator because it generates the minterms at its outputs for each of the binary values applied to its inputs. For the inputs $B1\ B0$ in Figure 3.17a, observe that the outputs are $F0(B1,B0) = \overline{B1}\cdot\overline{B0} = m0$, $F1(B1,B0) = \overline{B1}\cdot B0 = m1$, $F2(B1,B0) = B1\cdot\overline{B0} = m2$, and $F3(B1,B0) = B1\cdot B0 = m3$. Using this fact, it is very easy to manually design combination logic circuits for Boolean functions using a decoder with discrete ICs gates, as we will show in the next section.

In Figure 3.17a, when the binary input $B1B0$ is 00, output $F0$ evaluates to 1 and outputs $F1$, $F2$, and $F3$ evaluate to 0. When the binary input $B1B0$ is 01, output $F1$ evaluates to 1 and outputs $F0$, $F2$, and $F3$ evaluate to 0. When the binary input $B1B0$ is 10, output $F2$ evaluates to 1 and outputs $F0$, $F1$, and $F3$ evaluate to 0. When the binary input $B1B0$ is 11, output $F3$ evaluates to 1 and outputs $F0$, $F1$, and $F2$ evaluate to 0. This explanation is represented by the truth table for the 2-to-4 decoder shown in Table 3.1.

n-2^n译码器：Decoder，将作用于n个输入线的一个二进制码转换到2^n个不同的输出线中的一个的电路。

TABLE 3.1 Truth table for the 2-to-4 decoder in Figure 3.17

Select inputs		Outputs			
$B1$	$B0$	$F0$	$F1$	$F2$	$F3$
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1

Outputs $F0$ through $F3$ in Table 3.1 are active-high outputs; that is, each decoded input results in *just one* 1 on the outputs. If we change the 1s to 0s and the 0s to 1s for the outputs in Table 3.1, we obtain a 2-to-4 decoder with active-low outputs; that is, each decoded input results in *just one* 0 on the outputs. Larger decoders (3-to-8, 4-to-16, 5-to-32, etc.) have similar truth tables that operate in a similar manner. Decoder circuits are available as discrete off-the-shelf IC devices. Decoders are often used in microprocessor or microcontroller systems as an address decoder that **selects a specific device** in the system such as a **RAM (random-access memory)**, a **ROM (read-only memory)**, or an **I/O (input/output) device** via the outputs of the decoder.

Listing 3.6 shows a complete VHDL design for the 2-to-4 decoder.

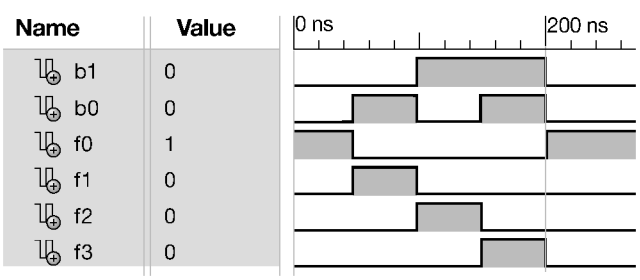
```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb6 is port (
     B1, B0 : in std_logic;
     F0, F1, F2, F3 : out std_logic
     );
end comb6;

architecture Boolean_functions of comb6 is
begin
     F0 <= not B1 and not B0;
     F1 <= not B1 and B0;
     F2 <= B1 and not B0;
     F3 <= B1 and B0;
end Boolean_functions;
```

LISTING 3.6 Complete VHDL design for the 2-to-4 decoder (project: comb6)

Waveform 3.3 shows waveform diagrams for the VHDL design for the 2-to-4 decoder.



WAVEFORM 3.3 Waveform diagrams for the VHDL design for the 2-to-4 decoder

Off-the-shelf decoders are usually equipped with one or more enable inputs—some active high and some active low. A decoder with an enable input is also called a **demultiplexer**. Table 3.2 shows the truth table for a 3-to-8 decoder with an active high enable input G1, an active low enable input G2, and active-low outputs. The 3-to-8 decoder shown in Table 3.2 is very similar to the off-the-shelf Texas Instruments CD74AC138 3-line to 8-line decoder/demultiplexer. As discussed earlier, logic products for Texas Instruments and data sheets are available online at http://ti.com.

TABLE 3.2 Truth table for the 3-to-8 decoder/demultiplexer

Enable inputs		Select inputs			Outputs							
*G*1	*G*2	*B*2	*B*1	*B*0	*F*0	*F*1	*F*2	*F*3	*F*4	*F*5	*F*6	*F*7
0	×	×	×	×	1	1	1	1	1	1	1	1
×	1	×	×	×	1	1	1	1	1	1	1	1
1	0	0	0	0	0	1	1	1	1	1	1	1
1	0	0	0	1	1	0	1	1	1	1	1	1
1	0	0	1	0	1	1	0	1	1	1	1	1
1	0	0	1	1	1	1	1	0	1	1	1	1
1	0	1	0	0	1	1	1	1	0	1	1	1
1	0	1	0	1	1	1	1	1	1	0	1	1
1	0	1	1	0	1	1	1	1	1	1	0	1
1	0	1	1	1	1	1	1	1	1	1	1	0

Using only the 0 in the column of the output function $F0$, we can write the Boolean function for $F0$ as $\overline{F0} = G1 \cdot \overline{G2} \cdot \overline{B2} \cdot \overline{B1} \cdot \overline{B0}$ or as $F0 = \overline{G1 \cdot \overline{G2} \cdot \overline{B2} \cdot \overline{B1} \cdot \overline{B0}}$. In VHDL, the latter form is written as

F0 <= **not** (G1 **and not** G2 **and not** B2 **and not** B1 **and not** B0).

Listing 3.7 shows a complete VHDL design for the 3-to-8 decoder/demultiplexer in Table 3.2.

LISTING 3.7 Complete VHDL design for the 3-to-8 decoder (project: comb7)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb7 is port (
    G1, G2, B2, B1, B0 : in std_logic;
    F0, F1, F2, F3, F4, F5, F6, F7 : out std_logic
    );
end comb7;

architecture Boolean_functions of comb7 is
begin
    F0 <= not (G1 and not G2 and not B2 and not B1 and not B0);
    F1 <= not (G1 and not G2 and not B2 and not B1 and B0);
    F2 <= not (G1 and not G2 and not B2 and B1 and not B0);
    F3 <= not (G1 and not G2 and not B2 and B1 and B0);
    F4 <= not (G1 and not G2 and B2 and not B1 and not B0);
    F5 <= not (G1 and not G2 and B2 and not B1 and B0);
    F6 <= not (G1 and not G2 and B2 and B1 and not B0);
    F7 <= not (G1 and not G2 and B2 and B1 and B0);
end Boolean_functions;
```

3.7.1 Designing Logic Circuits with Decoders and Single Gates

It is rather easy to manually design a logic circuit using a decoder and a single gate (AND gate, OR gate, NAND gate, or NOR gate). The design technique utilizes the fact that a decoder generates all possible minterms for the input variables. ORing the required minterms for the 1s of the function is the job of an OR gate when designing with a decoder that has active high outputs. If a decoder has active low outputs, then ORing the minterms is the job of a NAND gate drawn as an OR form—that is, its DeMorgan equivalent gate symbol.

ORing the required minterms for the 0s of the function is the job of a NOR gate when designing with a decoder that has active high outputs. If a decoder has active low outputs, then ORing the minterms is the job of an AND gate drawn as an OR form—that is, its DeMorgan equivalent gate symbol.

When designing with a decoder and a gate, the function does not have to be reduced. Table 3.3 shows the truth table for the function $F1(A,B,C) = \Sigma m(2,3,5,7)$.

TABLE 3.3 Truth table for function *F1*

A	B	C	$F1$
0	0	0	0
0	0	1	0
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 3.18a shows a design for the function $F1(A,B,C) = \Sigma m(2,3,5,7)$ in Table 3.3 using a 3-to-8 decoder with an OR gate.

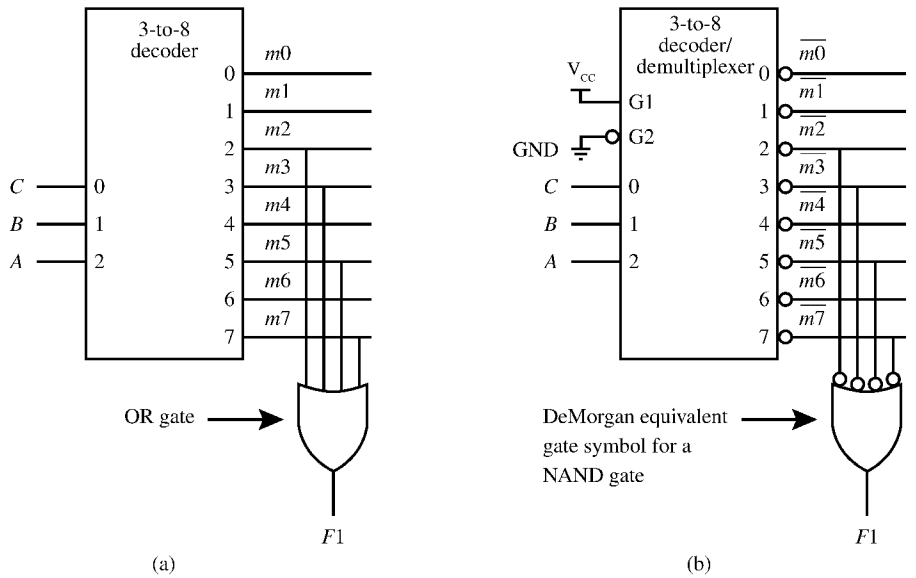


FIGURE 3.18 Design of the function $F1(A,B,C) = \Sigma m(2,3,5,7)$ (a) using a 3-to-8 decoder with active high outputs for the 1s of the function $F1$ and (b) using a 3-to-8 decoder/demultiplexer with active low outputs for the 1s of the function $F1$

Figure 3.18b shows an equivalent design for the function $F1(A,B,C) = \Sigma m(2,3,5,7)$ in Table 3.3 using a 3-to-8 decoder/demultiplexer with a NAND gate. Notice that both designs use the function expressed as the minterms of the 1s of the function $F1$.

Listing 3.8 shows a complete VHDL design for the function $F1(A,B,C) = \Sigma m(2,3,5,7)$.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb8 is port (
    A, B, C : in std_logic;
    F1 : out std_logic
    );
end comb8;

architecture Boolean_function of comb8 is
begin
    F1 <= (not A and B and not C) or (not A and B and C) or
                    --minterms 2 and 3
          (A and not B and C) or (A and B and C);
                    --minterms 5 and 7
end Boolean_function;
```

LISTING 3.8 Complete VHDL design for the function $F1(A,B,C) = \Sigma m(2,3,5,7)$ (project: comb8)

The function $F1$ expressed as the minterms of the 0s of the function is written as $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$. Figure 3.19a shows a design for the function $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$ using a 3-to-8 decoder with a NOR gate, which ORs the minterms and complements the result.

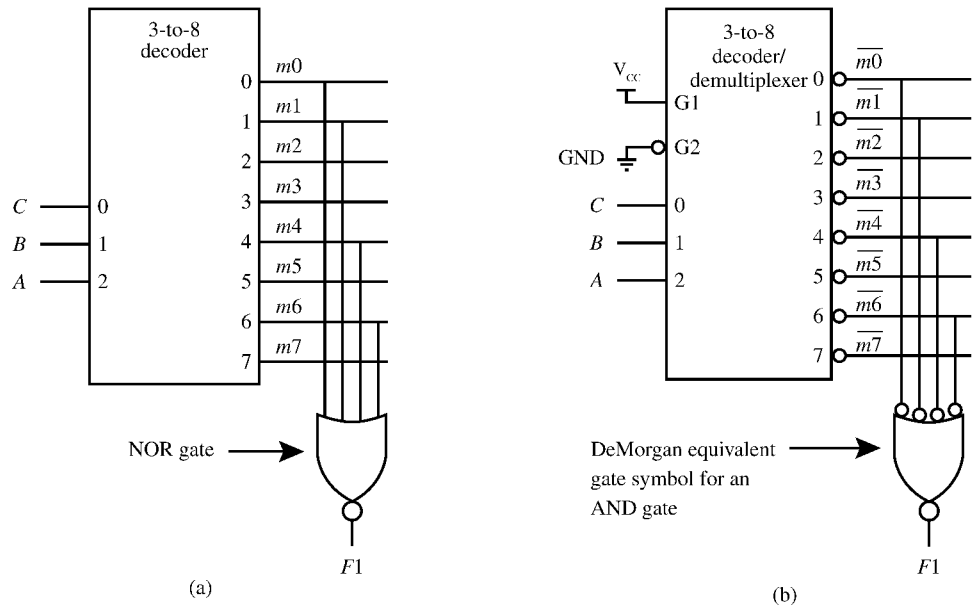


FIGURE 3.19 Design for the function $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$: (a) using a 3-to-8 decoder with active high outputs for the 0s of the function $F1$; (b) using a 3-to-8 decoder/demultiplexer with active low outputs for the 0s of the function $F1$

Figure 3.19b shows an equivalent design for the function $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$ using the 3-to-8 decoder/demultiplexer with an AND gate. If you are confused with the names associated with the DeMorgan equivalent gate symbols, now is a good time to review the DeMorgan equivalent gate symbols back in Chapter 1, Section 1.5.1.

When implementing a function with a decoder, it is best to use the fewest number of 1s or 0s to make up the function so that the fan-in of the gate is as small as possible. If there are fewer 1s in the function, use the compact minterm form for the 1s of the function to obtain the design; however, if there are fewer 0s in the function, use the compact minterm form for the 0s of the function. Additional gates can also be added to provide for additional outputs, thus allowing more than a single function to be implemented with a decoder.

Listing 3.9 shows a complete VHDL design for the function $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$.

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb9 is port (
    A, B, C : in std_logic;
    F1 : out std_logic
    );
end comb9;

architecture Boolean_function of comb9 is
begin
    F1 <= not ((not A and not B and not C) or (not A and not B and C) or
                    --minterms 0 and 1
               (A and not B and not C) or (A and B and not C));
                    --minterms 4 and 6
end Boolean_function;
```

LISTING 3.9 Complete VHDL Design for the function $\overline{F1}(A,B,C) = \Sigma m(0,1,4,6)$ (project: comb9)

3.8 MULTIPLEXERS

Figure 3.20a shows a very versatile circuit called a **multiplexer** or **MUX** that utilizes a NOT gate, a couple of AND gates, and an OR gate shown in AND/OR form. A MUX is a circuit that is used to direct one of 2^n data inputs to a single output. Because n select lines are used to select each of the 2^n data input signals and direct it to the output, a MUX is also called a **data selector**. Figure 3.20a shows the circuit diagram for a 2-line to 1-line MUX, which we will just call a 2-to-1 MUX.

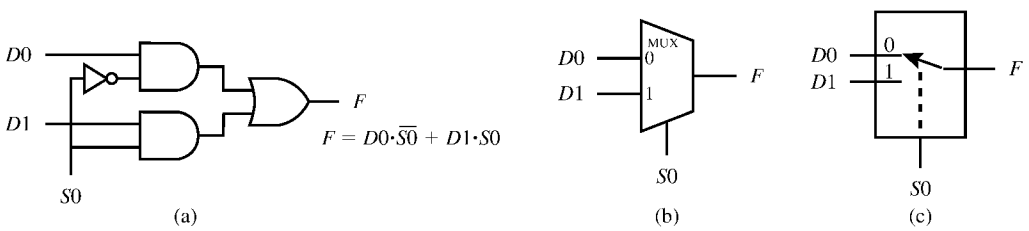


FIGURE 3.20
Multiplexer: (a) gate-level circuit diagram; (b) logic symbol; (c) switch representation

To observe how the MUX works, look at the switch representation in Figure 3.20c. When the select input $S0$ is 0, output F is $D0$, and when $S0$ is 1, output F is $D1$. The equation of the MUX in Figure 3.20a provides the same result when $S0$ is 0 and when $S0$ is 1 as shown as follows:

$F = D0 \cdot \overline{S0} + D1 \cdot S0 = D0$ when $S0 = 0$

$F = D0 \cdot \overline{S0} + D1 \cdot S0 = D1$ when $S0 = 1$

The logic symbol for the MUX in Figure 3.20b implies that $F = D0$ when $S0 = 0$ and $F = D1$ when $S0 = 1$. The truth table for the 2-to-1 MUX is shown in Table 3.4.

Inputs			Output
$S0$	$D1$	$D0$	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

TABLE 3.4 Truth table for the 2-to-1 MUX in Figure 3.20

Notice in the truth table that output F follows (is the same as) input $D0$ when $S0$ is 0, but output F follows (is the same as) input $D1$ when $S0$ is 1. Based on this observation, we can write a compact or compressed form of the truth table for the 2-to-1 MUX as shown in Table 3.5.

$S0$	F
0	$D0$
1	$D1$

TABLE 3.5 Compressed truth table for the 2-to-1 MUX in Figure 3.20

|导读|

本节介绍另一种常见的组合逻辑电路—多路选择器，并给出用多路选择器设计逻辑电路的方法。

多路选择器：
multiplexer，用来将2^n个数据输入中的一个传到一个单输出上的电路。

Listing 3.10 shows a complete VHDL design for the function $F = D0 \cdot \overline{S0} + D1 \cdot S0$ for the 2-to-1 MUX.

LISTING 3.10
Complete VHDL design for the function $F = D0 \cdot \overline{S0} + D1 \cdot S0$ for the 2-to-1 MUX (project: comb10)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb10 is port (
    D1, D0, S0 : in std_logic;
    F : out std_logic
    );
end comb10;

architecture Boolean_function of comb10 is
begin
    F <= (D0 and not S0) or (D1 and S0);
end Boolean_function;
```

Larger MUXs or data selectors (4-to-1, 8-to-1, 16-to-1, etc.) have similar truth tables and operate in a similar manner with more inputs. Table 3.6 shows the compressed truth table for a 2-to-1 MUX with an active low strobe input G. When the strobe input is 1 the output is 0, and when the strobe input is 0 the output is selected from one of the two data inputs and is routed to the output. The 2-to-1 MUX shown in Table 3.6 performs the same as one-fourth of an off-the-shelf Texas Instruments CD74AC157 Quadruple 2-line to 1-line data selector/multiplexer. CD stands for compliant device and is a lead-free device. When discarded, these devices do not pollute the world by contributing to lead contamination because they are lead free.

TABLE 3.6 Compressed truth table for the 2-to-1 MUX with an active low strobe input *G*

$G0$	$S0$	F
1	×	0
0	0	$D0$
0	1	$D1$

Figure 3.21a shows the logic symbol and output function for the 2-to-1 MUX in Table 3.6, and Figure 3.21b shows the logic symbol and output function for a 4-to-1 MUX without a strobe input.

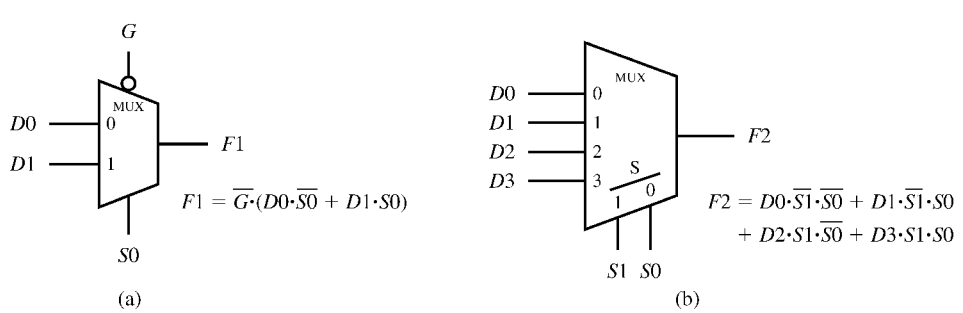


FIGURE 3.21 (a) Logic symbol and output function for the 2-to-1 MUX with an active low strobe input G; (b) Logic symbol and output function for a 4-to-1 MUX without a strobe input

MUXs are used to *implement designs for logic functions* and to *provide data-flow paths* between circuits by using MUXs as steering or routing circuits. We discuss the implementation of logic functions with MUXs in the following section.

Listing 3.11 shows a complete VHDL design for the 2-to-1 MUX and the 4-to-1 MUX in Figure 3.21.

LISTING 3.11
Complete VHDL design for the 2-to-1 MUX and the 4-to-1 MUX in Figure 3.21 (project: comb11)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb11 is port (
     D3, D2, D1, D0, S1, S0, G : in std_logic;
     F1, F2 : out std_logic
     );
end comb11;

architecture Boolean_functions of comb11 is
begin
     F1 <=  not G and ((D0 and not S0) or (D1 and S0));
     F2 <=  (D0 and not S1 and not S0) or (D1 and not S1 and S0) or
            (D2 and S1 and not S0) or (D3 and S1 and S0);
end Boolean_functions;
```

3.8.1 Designing Logic Circuits with MUXs

To obtain a MUX design for a logic function, we use the compact minterm form of the function. To implement the function, connect the data inputs of the MUX to the function values (V_{CC} for 1 and GND for 0). Connect the select lines of the MUX to the input variables. To generate the design for a function, simply write the compact minterm form for the function; that is, the function does not have to be reduced.

Figure 3.22a shows a MUX design for the function $F1(X,Y,Z) = \Sigma m(1,2,5,7)$ or the function $\overline{F1}(X,Y,Z) = \Sigma m(0, 3, 4, 6)$. Simply connect the data inputs of the MUX to the values of the function $F1$. If you wish, you may write the truth table for the function and then use the function values in the truth table, or you can simply make the connections for the 1s and by default the rest of the connections are for the 0s, or vice versa. The select inputs $S3$, $S2$, and $S1$ are then connected to the input X, Y, and Z respectively. The MUX design shown in Figure 3.22a uses an 8-to-1 MUX, such as a Texas Instruments CD74AC151, with an active low strobe input. The MUX design in Figure 3.22b does not have a strobe input.

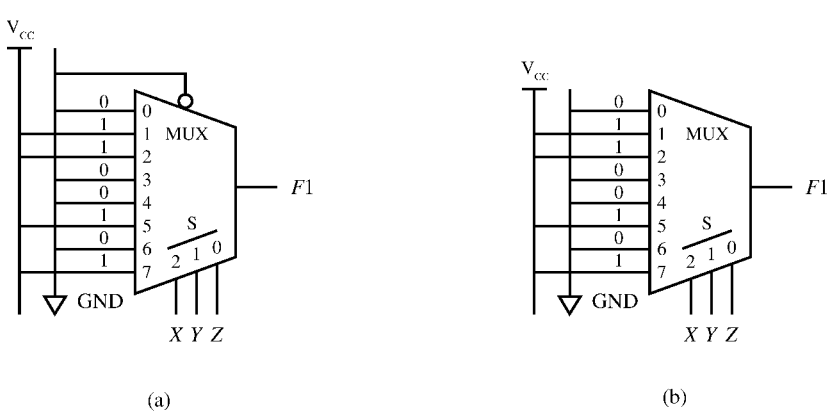


FIGURE 3.22 MUX design: (a) with a strobe input; (b) without a strobe input

With this manual procedure, you can obtain a MUX design for any 2-variable function using a 4-to-1 MUX, or you can obtain a MUX design for any 3-variable function using an 8-to-1 MUX. For a MUX design you need a 2^n-to-1 MUX for any n-variable function. Notice that you do not have to reduce a function to obtain a MUX design, but you must obtain the truth table of the function or obtain the function in compact minterm form for its 1s or 0s.

Listing 3.12 shows a complete VHDL design for the function $F1(X,Y,Z) = \Sigma m(1,2,5,7)$.

LISTING 3.12
Complete VHDL design for the function $F1(X,Y,Z) = \Sigma m(1,2,5,7)$ (project: comb12)

```
library IEEE;
use IEEE.STD_LOGIC_1164.ALL;

entity comb12 is port (
    X, Y, Z : in std_logic;
    F1 : out std_logic
    );
end comb12;

architecture Boolean_function of comb12 is
begin
    F1 <= (not X and not Y and Z) or (not X and Y and not Z) or
                --minterms 1 and 2
          (X and not Y and Z) or (X and Y and Z);
                --minterms 5 and 7
end Boolean_function;
```

As you can see, the manual MUX design methods that we presented are rather easy to use and understand, but you still must obtain a detailed logic diagram for the circuit so that the circuit can be wired up on a breadboard or on a PC board. The modern way is to write VHDL code for the design and then download the bit pattern into a CPLD or an FPGA. The software automatically wires up the circuit on the CPLD or FPGA chip. You must remember to assign the external package pins for each of the signals that are placed in the entity.

3.9 HAZARDS

|导读|

本节介绍会引起电路输出出现毛刺的险象问题，包括功能险象和逻辑险象。

Hazards are classified as either **function hazards** or **logic hazards**. In this section, we present a brief introduction to these two different types of hazards. A hazard can cause a logic **glitch**, where a glitch is an undesired momentary pulse that occurs at the output of a circuit. In some cases, glitches in a circuit can cause a circuit to fail.

3.9.1 Function Hazards

A hazard that can cause a glitch in the output signal of a combinational logic function implemented with gates, when two or more input signals are changed at the same time in the circuit due to the way the function is defined, is called a **function hazard**. A function hazard can be spotted by plotting the function in a K-map. If two or more input signals are changed in the function to produce the output, a logic glitch may occur. In a combinational logic circuit, the designer has no control over function hazards. Figure 3.23 illustrates the occurrence of static and dynamic function hazards and their corresponding glitches for the function $F1$.

The function in Figure 3.23 is a 3-input XOR gate. All hardware implementations of this function will contain the function hazards, so showing the circuit is not necessary. The directed lines in the K-map show the transitions that cause each of the function hazards and their corresponding glitches—that is, a static 1 function hazard, a static 0 function hazard, a dynamic 1-to-0 function hazard, and a dynamic 0-to-1 function hazard. Notice in Figure 3.23 that a **runt pulse** can also occur. A runt pulse is a pulse with small amplitude.

Function hazards cannot be eliminated; however, the output signals from circuits that contain function hazards may be used by simply waiting until the function hazards settle (die out). After the output signals become stable or the function hazards settle, the signals may be used. This concept is the basis of synchronous circuits that are introduced in Chapters 6 and 9, where settling occurs between clock ticks.

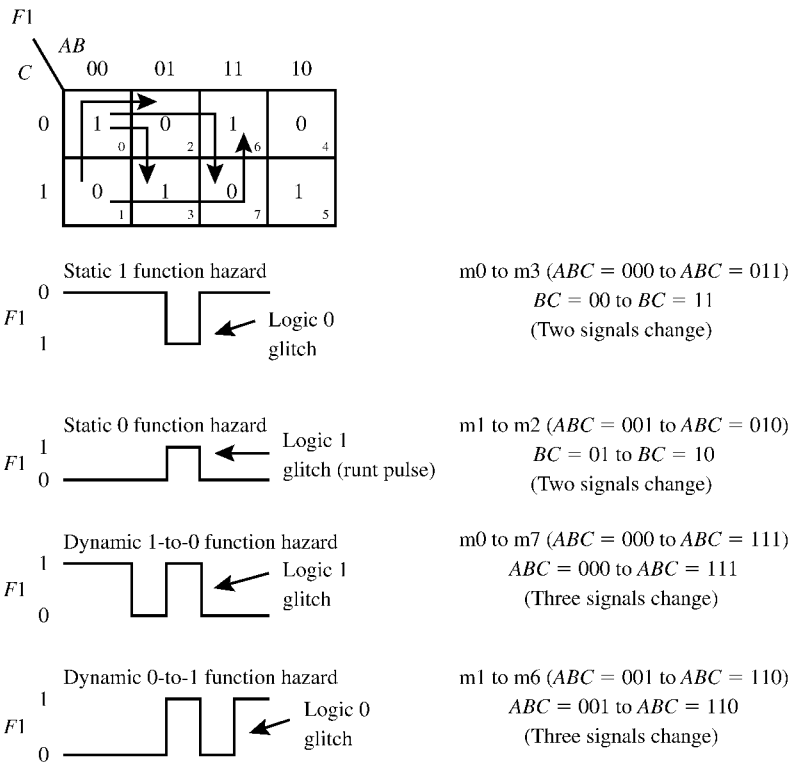


FIGURE 3.23 Function hazards—hazards that result from two or more input signals changing at the same time

功能险象：
function hazard，当电路中两个或者更多个输入信号在同一时刻发生变化，在用门实现的组合逻辑函数的输出信号引起一个毛刺的险象。

3.9.2 Logic Hazards

A hazard that can cause a glitch in the output signal of a combinational logic function implemented with gates, when only one input signal is changed due to delays in the particular circuit used to implement the function, is called a **logic hazard**. Both static and dynamic logic hazards can be eliminated by adding additional product terms in the Boolean equation implemented by the circuit (this requires adding more gates in the implementation). To eliminate a logic hazard, a designer must recognize that a logic hazard may occur and add the necessary circuitry to prevent the logic hazard. Some functions do not contain logic hazards. Figure 3.24 shows a design specification with a single static 1 logic hazard. The single static 1 logic hazard represented in Figure 3.24a exists for any realization of the minimized logic function. Figure 3.24b shows a circuit for the function and a plausible explanation for the single static 1 logic hazard to be present.

Figure 3.25a shows how the single static 1 logic hazard can be eliminated, and Figure 3.25b shows a circuit for the function that eliminates the single static 1 logic hazard.

Things you should know about eliminating static and dynamic logic hazards:

- Logic hazards *may occur* for a minimized function implemented for the 1s or 0s of the function. If the product terms for the function are linked to each other, as shown for the K-map for the 1s of the function in Figure 3.25a, then the function contains no logic hazards.
- For the minimized 1s or 0s of a function, one can add logic hazard cover terms *if required*, which are consensus terms, to eliminate static and dynamic logic hazards.
- Cover terms are nonessential product terms that are used to link each product term in the minimized form in the K-map for the function.

- In some cases, product terms for a minimized form of the function are linked to each other and require no additional cover terms.
- By chain linking all the minimized product terms in a K-map for the function, you will obtain the required cover terms to add to the minimized form of the function to eliminate all the static and dynamic logic hazards for the function.
- A function that has all of its products terms for the 1s or 0s of the function linked to each other does not have static or dynamic logic hazards and is called a **logic hazard-free function.**

逻辑险象：
logic hazard,当电路中只有一个输入信号发生变化,由于实现函数所用的特定电路存在延时,在用门实现的组合逻辑函数的输出信号引起一个毛刺的险象。

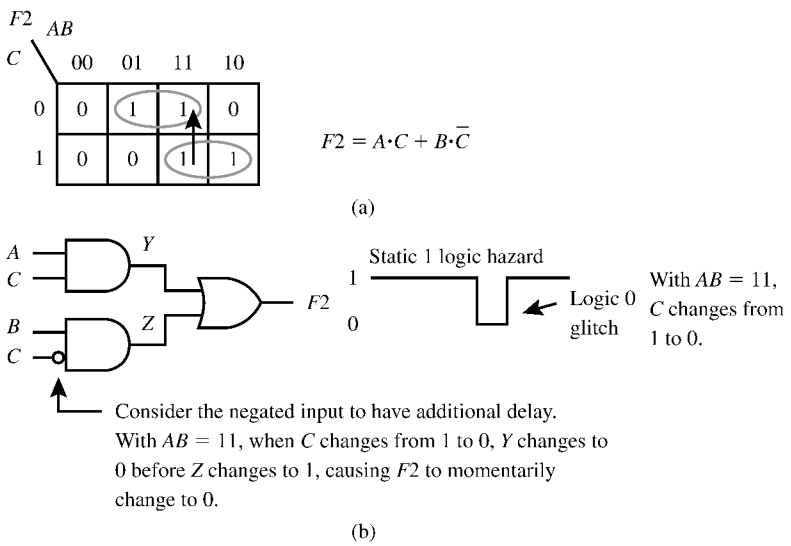


FIGURE 3.24 (a) Design specification with a single static 1 logic hazard; (b) circuit for the function and a plausible explanation for the single static 1 logic hazard to be present

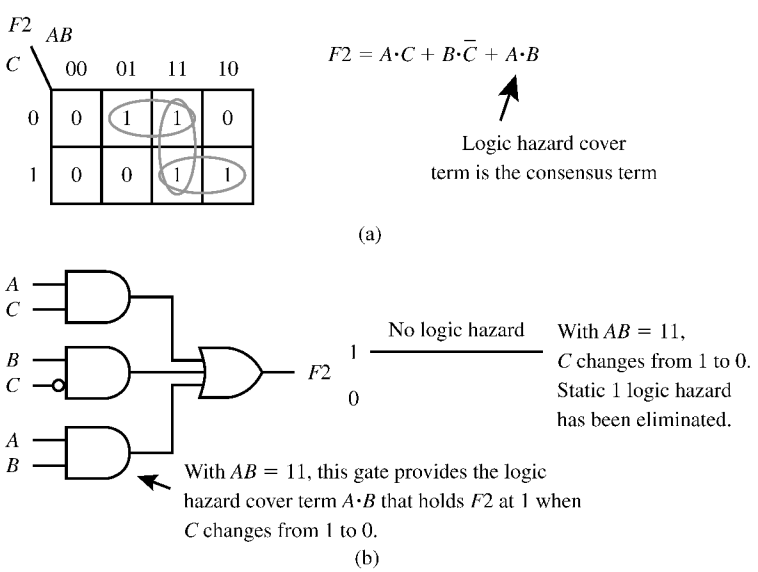


FIGURE 3.25 (a) How a single static 1 logic hazard is eliminated; (b) circuit for the function that eliminates the single static 1 logic hazard

Logic hazards may be eliminated; however, the output signals from circuits that contain logic hazards may also be used by simply waiting until the logic hazards settle (die out). After the output signals become stable or the logic hazards settle, the signals may be used. This concept is the basis of synchronous circuits that are introduced in Chapters 6 and 9, where settling occurs between clock ticks.

PROBLEMS

Section 3.2 Integrated Circuit Devices

3.1 What can physical hardware devices called integrated circuits (ICs) do?

3.2 What type of wire is used to connect together the die and the package leads or pins inside an IC?

3.3 What does an IC with a hermetical seal provide?

3.4 Name a few different types of integrated circuit packages.

3.5 Which IC package type has balls of solder on its pins that are soldered directly to a PC board?

Section 3.3 Analyzing and Designing Logic Circuits

3.6 What is the process of circuit analysis?

3.7 What is the process of circuit design or synthesis?

3.8 Show a common symbol for normally closed relay contacts.

3.9 Show a common symbol for normally open relay contacts.

3.10 Relays or switches connected in series provide what logic operation?

3.11 Relays or switches connected in parallel provide what logic operation?

3.12 What is the name that is used for logic switching circuit in power applications?

3.13 Analyze the logic switching circuit shown in Figure P3.13 to obtain its function *F*.

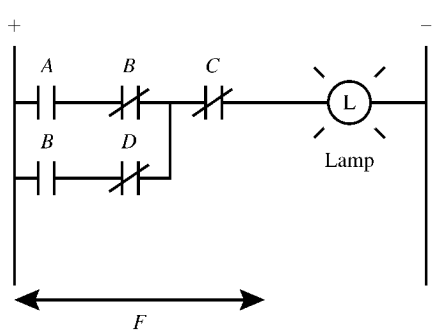


FIGURE P3.13

3.14 Show the design for an OR gate ladder logic circuit driving a lamp.

3.15 Show the design for an AND gate ladder logic circuit driving a control relay (CR).

3.16 Show the design for an NAND gate ladder logic circuit driving a lamp.

3.17 Analyze the logic circuit shown in Figure P3.17 to obtain its function *F* in SOP form and its truth table.

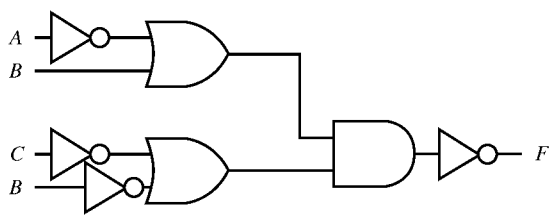


FIGURE P3.17

3.18 Analyze the logic circuit shown in Figure P3.18 to obtain its function *F* in SOP form and its truth table.

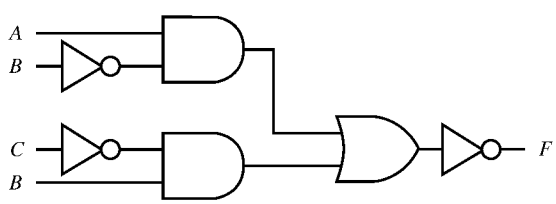


FIGURE P3.18

3.19 Analyze each of the following circuits in Figure P3.19 to obtain their Boolean function in SOP form and their truth table.

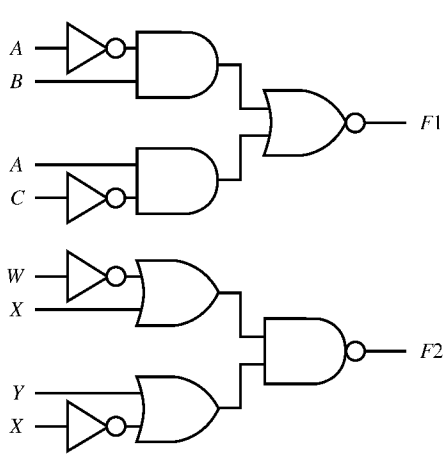


FIGURE P3.19

3.20 Design a circuit that provides the 1's complement at its output for each 4-bit binary number applied at its input. Use the input signals *IN*3, *IN*2, *IN*1, and *IN*0 and the

corresponding output signals *OUT*3, *OUT*2, *OUT*1, and *OUT*0. Hint: The 1's complement of a binary number is simply the complement of each individual bit.

3.21 Show a complete VHDL design for a circuit that provides the 1's complement at its output for each 4-bit binary number applied at its input. Use the input signals *IN*3, *IN*2, *IN*1, and *IN*0 and the corresponding output signals *OUT*3, *OUT*2, *OUT*1, and *OUT*0. Hint: The 1's complement of a binary number is simply the complement of each individual bit.

3.22 Design a circuit that provides the 2's complement at its output for each 3-bit binary number applied at its input. Use the input signals *X*, *Y*, and *Z* and the corresponding output signals *F*1, *F*2, and *F*3, and only AND, OR, and NOT gates. Hint: The 1's complement of a binary number is simply the complement of each individual bit. The 2's complement of a binary number is the 1's complement of the binary number + 1 (i.e., the 1 is added to the least significant bit).

3.23 Show a complete VHDL design for a circuit that provides the 2's complement at its output for each 3-bit binary number applied at its input. Use the input signals *X*, *Y*, and *Z* and the corresponding output signals *F*1, *F*2, and *F*3. Hint: The 1's complement of a binary number is simply the complement of each individual bit. The 2's complement of a binary number is the 1's complement of the binary number + 1 (i.e., the 1 is added to the least significant bit).

3.24 Design a majority of 1s circuit such that the output signal *F* is 1 when a majority of the input signals *X*, *Y*, and *Z* are 1. Only use OR, AND, and NOT gates.

3.25 Use VHDL to design a majority of 1s circuit such that the output signal *F* is 1 when a majority of the input signals *X*, *Y*, and *Z* are 1.

Section 3.4 Generating Detailed Schematics

3.26 What is the difference between a functional logic or schematic diagram and a detailed logic or schematic diagram?

3.27 List a good online source for obtaining datasheets for IC devices as mentioned in the text.

3.28 How many gates are contained in the IC device IC1 in Figure 3.10a in the text? Name the gates.

3.29 Which pin must be connected to V_{CC} and which pin must be connected to GND for IC device IC1 in Figure 3.10a in the text?

3.30 How many gates are contained in the IC device IC2 in Figure 3.10a in the text? Name the gates.

3.31 Which pin must be connected to V_{CC} and which pin must be connected to GND for IC device IC2 in Figure 3.10a in the text?

3.32 How many gates are contained in the IC device IC1 in Figure 3.10b in the text? Name the gates.

3.33 Which pin must be connected to V_{CC} and which pin must be connected to GND for IC device IC1 in Figure 3.10b in the text?

3.34 List the important items that are necessary when drawing a detailed schematic.

3.35 Why do you need to know how to provide detailed schematics of your designs?

3.36 Show a complete VHDL design for the functions $F1 = A \cdot B + \overline{A} \cdot \overline{B}$ and $\overline{F2} = \overline{A} \cdot B$.

3.37 What is the purpose of running a simulation on a VHDL design?

3.38 To obtain a circuit on a system board, why is a detailed schematic not required when using a hardware description language such as VHDL?

Section 3.5 Designing Circuits in NAND/NAND and NOR/NOR Form

3.39 Design a two-1s-out-of-four event detector for input signals *A*, *B*, *C*, *D* and output signal *F*. Use a vertical-input scheme and fan-in reduction if possible using NOT gates (six in a package), 4-input NAND gates (two in a package), and an 8-input NAND gate (one in a package). Use the graphical design method.

3.40 Show a complete VHDL design for a two-1s-out-of-four event detector for input signals *A*, *B*, *C*, *D* and output signal *F*.

3.41 Design a circuit using the 1s of the function $F(X,Y,Z) = \Sigma m(1,2,4)$. If possible, reduce the function. Draw the circuit for the function using just NAND gates in NAND/NAND form and NOT gates. Use the graphical design method.

3.42 Show a complete VHDL design for a circuit using the 1s of the function $F(X,Y,Z) = \Sigma m(1,2,4)$.

3.43 Design a circuit using the 1s of the function $F(X,Y,Z) = \Sigma m(1,2,4)$. If possible, reduce the function. Draw the circuit for the function using just NOR gates in NOR/NOR form and NOT gates. Use the graphical design method.

3.44 Design a circuit using the 1s of the function $F(W,X,Y,Z) = \Sigma m(0,2,5,7,8,10) + \Sigma md(12, 13)$. If possible, reduce the function. Draw the circuit for the function using just NAND gates in NAND/NAND form and NOT gates. Use the graphical design method.

3.45 Show a complete VHDL design for a circuit using the 1s of the function $F(W,X,Y,Z) = \Sigma m(0,2,5,7,8,10) + \Sigma md(12,13)$. Hint: Choose 0s for the don't cares, to reduce the number of minterms in the VHDL expression for *F*.

3.46 Design a circuit using the 1s of the function $F(W,X,Y,Z) = \Sigma m(0,2,5,7,8,10) + \Sigma md(12,13)$. If possible, reduce the function. Draw the circuit for the function using just NOR gates in NOR/NOR form and NOT gates.

Section 3.6 Propagation Delay Time

3.47 Which metal has the best conductivity—copper, gold, or silver? Which has the second best conductivity? Which has the third best conductivity?

3.48 Provide an equation for t_p in terms of t_{PHL} and t_{PLH}.

3.49 Describe what is meant by the term *worst-case delay time through a circuit.*

3.50 Draw a circuit for a function *FDELAY* = *A* made up of four cascaded NOT gates. If each NOT gate has a delay

of t_p, what is the output delay of the circuit from its input to its output?

3.51 Draw a circuit for the function $\overline{FD1} = A \cdot \overline{B}$ implemented with an AND gate and NOT gates. Determine the worst-case output delay for the circuit, assuming each gate has a delay of t_p.

3.52 Draw a circuit for the function $FD2 = \overline{\overline{X} + Y}$ implemented with an OR gate and NOT gates. Determine the worst-case output delay for the circuit, assuming each gate has a delay of t_p.

Section 3.7 Decoders

3.53 Draw and label a gate level circuit for a 3-to-8 decoder with active low outputs. Also draw and label a logic symbol for the 3-to-8 decoder.

3.54 Design a circuit for the function $F(X,Y,Z) = \Sigma m(0,2,7)$ with a 3-to-8 decoder with active high outputs. Use an appropriate gate to provide the smallest possible fan-in to implement the function.

3.55 Show a complete VHDL design for the function $F(X,Y,Z) = \Sigma m(0,2,7)$.

3.56 Design a circuit for the function $F(X,Y,Z) = \Sigma m(0,1,5,6,7)$ with a 3-to-8 decoder/demultiplexer that has an active low enable input and active low outputs. Use an appropriate gate to provide the smallest possible fan-in to implement the function.

3.57 Show a complete VHDL design for the function $F(X,Y,Z) = \Sigma m(0,1,5,6,7)$. To simplify the function in VHDL, rewrite the function with a minimum number of minterms.

Section 3.8 Multiplexers

3.58 Write the equation for a 4-to-1 MUX (data selector); then draw and label the circuit. Use $D3$ down to $D0$ as the data inputs, $S1$ down to $S0$ as the select inputs, and F as the output. Draw a logic symbol for the circuit, and then show its truth table in compressed form.

3.59 Show a complete VHDL design for a 4-to-1 MUX (data selector). Use $D3$ down to $D0$ as the data inputs, $S1$ down to $S0$ as the select inputs, and F as the output.

3.60 Obtain a MUX design for the AND function $F(A,B) = A \cdot B$ using an off-the-shelf MUX without a strobe input.

3.61 Show a complete VHDL design for the AND function $F(A, B) = A \cdot B$.

3.62 Obtain a MUX design for the NOR function $F(A,B) = \overline{A + B}$ using an off-the-shelf MUX without a strobe input.

3.63 Show a complete VHDL design for the NOR function $F(A,B) = \overline{A + B}$.

3.64 Obtain a MUX design for the XOR function $F(X,Y) = X \cdot \overline{Y} + \overline{X} \cdot Y$ using an off-the-shelf MUX with an active low strobe input.

3.65 Show a complete VHDL design for the XOR function $F(X,Y) = X \cdot \overline{Y} + \overline{X} \cdot Y$.

3.66 Obtain a MUX design for the function $F(X,Y,Z) = \Sigma m(0,1,2,3,5,7)$ using an off-the-shelf MUX with an active low strobe input.

3.67 Show a complete VHDL design for the function $F(X,Y,Z) = \Sigma m(0,1,2,3,5,7)$. To simplify the function in VHDL, rewrite the function with a minimum number of minterms.

Section 3.9 Hazards

3.68 Name the two classifications of hazards.

3.69 What can a hazard cause at the output of a circuit?

3.70 List the reason a glitch can occur in a combinational logic circuit as a result of a function hazard.

3.71 List the four types of function hazards covered in the book.

3.72 List the two types of glitches covered in the book.

3.73 What is a runt pulse?

3.74 Can function hazards be eliminated? Describe how the outputs of circuits that contain function hazards can be used.

3.75 List the reason a glitch can occur in a combinational logic circuit as a result of a logic hazard.

3.76 Can logic hazards be eliminated? If so, describe how.

3.77 Eliminate all logic hazards that can result for the 1s of the Boolean function $F1(A,B,C) = \Sigma m(1,3,4,5)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.78 Eliminate all logic hazards that can result for the 0s of the Boolean function $F1(A,B,C) = \Sigma m(1,3,4,5)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.79 Eliminate all logic hazards that can result for the 1s of the Boolean function $F2(A,B,C) = \Sigma(1,2,3,6)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.80 Eliminate all logic hazards that can result for the 0s of the Boolean function $F2(A,B,C) = \Sigma(1,2,3,6)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.81 Eliminate all logic hazards that can result for the 1s of the Boolean function $F3(A,B,C) = \Sigma(0,2,3,4,6)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.82 Eliminate all logic hazards that can result for the 0s of the Boolean function $F3(A,B,C) = \Sigma(0,2,3,4,6)$. Show the K-maps and the equations for the minimum Boolean function and the logic hazard-free Boolean function.

3.83 Determine the number of logic hazards that the circuit shown in Figure P3.83 could contain. What product terms are necessary to eliminate these logic hazards? Write a logic hazard-free function for the circuits.

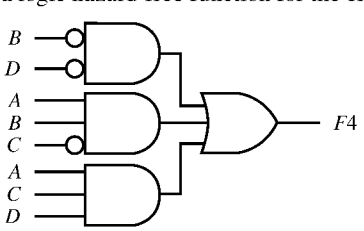


FIGURE P3.83