# 13

# LOGIC ELEMENTS

## CALCULATORS TO COMPUTERS

Ways of thinking about and dealing with numbers reflect the cultures and thinking patterns of different civilizations. Early number systems used mechanical calculators such as the abacus, a frame of wires on which *calculi* (pebbles) are slid to perform number combinations. About A.D. 800, the Arab al-Khowarizmi wrote a book using the decimal notation, based on the number of our *digits* (fingers, in Latin).

In the active commerce of the Middle Ages, buying and selling made it necessary to count and calculate quickly, and various counting systems developed. The new methods were neither more rapid nor more accurate than the abacus they replaced, but they are important in understanding the development of mathematics.

The first big step toward the computer was taken in 1642 by a 19-year-old Frenchman, Blaise Pascal, a mathematical genius with an inventive mind. He devised an adding machine consisting of ten numbered wheels linked by gears—the first simple digital calculating machine.

Gottfried Wilhelm Leibnitz, a 25-year-old German mathematician, invented the stepped wheel in 1671 and expanded on Pascal's machine. Two hundred years later the Swedish engineer Odhner invented his pinwheel mechanism and developed the fast and dependable mechanical disk calculator still widely used throughout the world.

In 1812, Charles Babbage, an English mathematician, then 20 years old, conceived a mechanical Difference Engine to perform routine calculations. Although Babbage's designs were practicable, the technology of his day did not permit the construction of the precise mechanisms he devised and his machine was never completed.

In 1833, Babbage invented his Analytical Engine, the first universal digital computer. In detailed designs he described a machine that would accept data on punched cards and, by combining the processes of arithmetic and logic, perform any calculation.

Engineers had grown accustomed to creating circuits with a minimum of active components because transistors and diodes were relatively more expensive than passive resistors and capacitors. However, the invention of the integrated circuit changed their way of thinking; active devices were both smaller and simpler to put on an IC chip than passive components. Digital computations, employing the base 2, required many more components than decimal devices. Digital computers became practical after 1970, when the low cost and high density of integrated circuits made it possible to design with a large number of components.



# 13

# LOGIC ELEMENTS



## 13-1 INTRODUCTION

The music on a phonograph record is preserved as a groove that guides a needle whose motion changes in amplitude and frequency as the record turns; these *continuous* or *analog* signals can take any value in a wide range of values. By contrast, the music on a compact disk is preserved as a pattern of flat areas and holes that either reflect light or do not; these *discrete* or *digital* signals are processed by circuits made up of electronic switches that are either on or off. A major virtue of electronic circuits is the ease and speed with which digital signals can be processed, and the use of such signals in control, computation, and communication is the most rapidly developing aspect of electronic engineering.

To process information in digital form we need special circuits, and for the efficient design of digital circuits we use a special numbering system and a special algebra. The circuits must provide for storing instructions and data, receiving new data, performing calculations, making decisions, and communicating the results. For example, an automatic airline reservation system must receive and store information from the airline regarding the number of seats available on flights all over the world, respond to inquiries from travel agents across the country, subtract the number of seats requested from the number available or add the number of seats canceled, handle 50 or so requests per minute, and keep no one waiting more than a minute.

Information processing is an important component of all branches of engineering and science. Aeronautical and chemical engineers may be designing automatic control systems. Civil and industrial engineers may be concerned with data on traffic flow or product flow. Mechanical engineers may be designing "smart" tools or products. Chemists and medical doctors may be interested in automated laboratory analysis. Biologists and physicists may need remote control of precisely timed events. Everyone engaged in experimental work or in management can benefit from the new data processing techniques.

In this chapter we begin our study of digital systems by learning the basic functions performed by decision-making elements and seeing how we can use semiconductor devices in practical circuits. Then we look at basic memory elements and see how we can construct them from bipolar and MOS transistors. Finally we examine more sophisticated memory elements with desirable operating features.

When you have assimilated the material in this chapter, you will be able to

identify the symbol of each of the basic logic elements, know the function it performs, and predict the output for given inputs. You will know how switches, decision-making, and memory devices operate in terms of the electrical behavior of their electronic components. You will be able to analyze the logic behavior of complex combinations of digital components. You will know how electronic logic devices are classified and how the various types compare in their electrical performance. You will be able to design circuits and specify components to perform simple digital operations. In the next chapter you will learn to design the data processing circuits and devices that are used in digital computers.

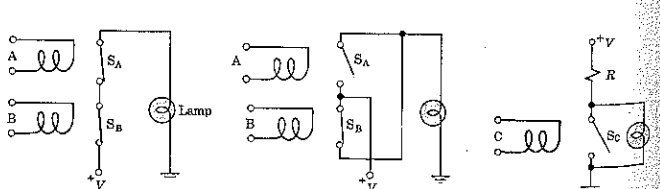
## 13-2 SWITCHING LOGIC

Digital computers, automatic process controls, and instrumentation systems have the ability to take action in response to input stimuli and in accordance with instructions. In performing such functions, an information processing system follows a certain *logic;* the elementary logic operations are described as **AND, OR,** and **NOT.** Tiny electronic circuits consuming very little power can perform such operations dependably, rapidly, and efficiently.

### LOGIC GATES

A *logic gate* is a device that controls the flow of information, usually in the form of pulses. First we consider gates employing magnetically operated switches called *relays.* If the switches are normally open, they close when input signals in the form of currents are applied to the relay coils.

In Fig. 13.1a, the lamp is turned on if switch A *and* switch B are closed; it is called an **AND** circuit or **AND** gate. In Fig. 13.1b, the lamp is turned on if switch A *or* switch B is closed *or* if both are closed; it is called an **OR** circuit. In Fig. 13.1c, the switch is connected *across* the lamp. For an input at C, the switch is closed, shorting out the lamp, and there is *not* an output; the input has been *inverted* by the **NOT** gate.




(a) **AND** gate (b) **OR** gate (c) **NOT** gate

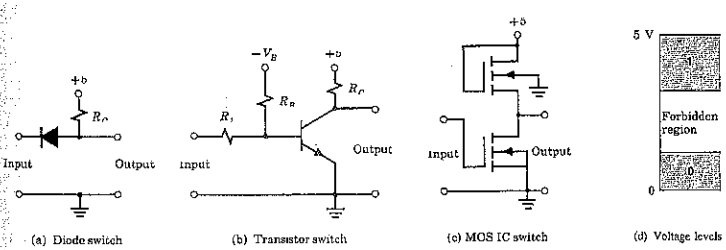
Figure 13.1 Gate circuits using relays.



In general, there may be several inputs and the output may be fed to several other logic elements. In a typical digital computer there are millions of such logic elements. Although the first practical digital computer (the Harvard Mark I, 1944) employed relays, the success of the modern computer is due to our taking advantage of the switching capability of semiconductor devices.

### ELECTRONIC SWITCHING

In the gates of Fig. 13.1, "data" are represented by **ON** and **OFF** switch positions or by the corresponding presence or absence of voltages. How can diodes and transistors be used as switches? If the input to the circuit in Fig. 13.2a is zero (i.e., if the input terminals are shorted), the diode is forward biased, current flows easily, and the output is a few tenths of a volt or approximately zero. If the input is 5 V or more, the diode is not forward biased, no diode current flows, and the output is +5 V. The **OFF–ON** positions of this diode switch correspond to output voltage levels near zero and +5 V. Intermediate output voltages, such as 2.5 V, would be ambiguous and circuits are designed so that intermediate voltages do not appear.




(a) Diode switch (b) Transistor switch (c) MOS IC switch (d) Voltage levels

Figure 13.2 Elementary semiconductor switching circuits.

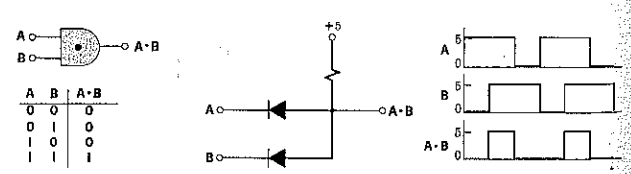
In the elementary transistor switch (Fig. 13.2b) with a zero input, the base–emitter junction is reverse biased by $-V_B$ across the voltage divider, the collector current is very small, and the output is approximately +5 V. An input of +5 V forward biases the base–emitter junction, a large collector current flows, and the output is approximately zero. This circuit is analogous to Fig. 13.1c; the input signal is inverted.

The MOS transistor can operate as a binary switch controlled by changes in gate voltage. Because an MOS transistor can also function as a resistor, the arrangement of Fig. 13.2c is convenient in integrated circuits. With the gate of the upper or "load" transistor connected to the drain terminal, enhancement is high and the resistance of the load is determined by the channel dimensions. With no input signal, the lower or "driver" transistor is **OFF,** there is no $IR$ drop across the load, and the output is approximately +5 V. A positive input signal turns the driver **ON** and the output falls to nearly zero.

## BASIC LOGIC OPERATIONS

To make our work with logic circuits easier, we use the conventional symbols and nomenclature developed by circuit designers and computer architects. Each basic logic operation is indicated by a *symbol*, and its function is defined by a *truth table* that shows all possible input combinations and the corresponding outputs. Logic operations and variables are in boldface type. Electronic circuits are drawn as single-line diagrams with the ground terminal omitted; all voltages are with respect to ground so that "no input" means an input terminal shorted to ground. Two distinct voltage levels, separated by a forbidden region (Fig. 13.2d), electronically represent the *binary* numbers **1** and **0** corresponding to the **TRUE** and **FALSE** of logic.

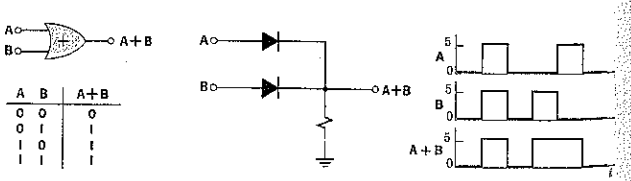
**AND** GATE. The symbol for an **AND** gate is shown in Fig. 13.3a, where **A · B** is read "**A AND B.**" As indicated in the truth table, an output appears only when there are inputs at **A AND B.** If the inputs in Fig. 13.3b are in the form of positive voltage pulses (with respect to ground), inputs at A and B turn off both diodes, no current flows through the resistance, and there is a positive output (**1**). In general, there may be several input terminals. If any one of the inputs is zero (**0**), current flows through that forward-biased diode and the output is nearly zero (**0**). For two inputs varying with time, a typical response is shown in Fig. 13.3c.




(a) Symbol and truth table (b) Diode circuit (c) Typical response

Figure 13.3 A two-input **AND** gate.

**OR** GATE. The symbol for an **OR** gate is shown in Fig. 13.4a, where **A + B** is read "**A OR B.**" As indicated in the truth table, the output is **1** if input **A OR** input **B** is 1. For no input (zero voltage) in Fig. 13.4b, no current flows, and the



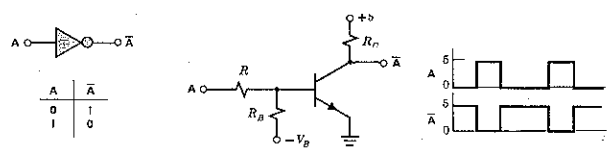

(a) Symbol and truth table (b) Diode circuit (c) Typical response

Figure 13.4 A two-input **OR** gate.



output is zero (**0**). An input of +5 V (**1**) at either terminal A or B or both (or at *any* terminal in the general case) forward biases the corresponding diode, current flows through the resistance, and the output voltage rises to nearly 5 V (**1**). For two inputs varying with time, the response is as shown.

**NOT** GATE. The inversion inherent in a transistor circuit corresponds to a logic **NOT** represented by the symbol in Fig. 13.5a, where $\overline{\mathbf{A}}$ is read "**NOT A.**" As indicated in the truth table, the **NOT** element is an *inverter;* the output is the *complement* of the single input. With no input (**0**), the transistor switch is held open by the negative bias voltage and the output is +5 V (**1**). A positive input voltage (**1**) forward biases the base–emitter junction, collector current flows, and the output voltage drops to a few tenths of a volt (**0**). For a changing input, the output response is as shown.

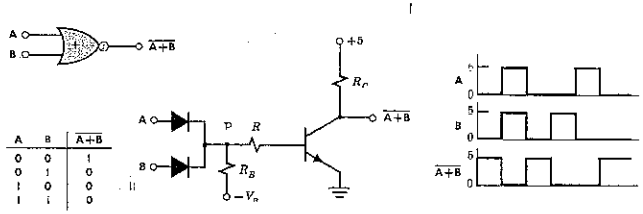



(a) Symbol and truth table (b) Transistor circuit (c) Typical response

Figure 13.5 A transistor **NOT** gate.

**NOR** GATE. In the diode **OR** gate of Fig. 13.4b, an input of +5 V at **A OR B** produces a voltage across $R$ and a positive output voltage. But this output is less than the input (by the diode voltage drop), and after a few cascaded operations the signal would decrease below a dependable level. A transistor supplied with +5 V can be used to restore the level as in Fig. 13.6b; however, the inherent inversion results in a **NOT OR** or **NOR** operation. The small circle on the **NOR** element symbol and the bar in the $\overline{\mathbf{A + B}}$ output indicate the inversion process.

In the **NOR** circuit with no input, the transistor switch is held **OFF** by the negative bias voltage $V_B$ and the output is +5 V (**1**). An input of +5 V at terminal A or B raises the base potential, forward biases the base–emitter junction, turns




(a) Symbol and truth table (b) Electronic circuit (c) Typical response

Figure 13.6 A diode–transistor **NOR** gate.

the transistor switch **ON**, and drops the output to nearly zero (**0**). As we shall see (p. 401), in addition to restoring the signal level, the transistor provides a relatively low output resistance so that this **NOR** element can supply inputs to many other gates. Another advantage is that all the basic logic operations can be achieved by using only **NOR** gates.

Exercise 13-1

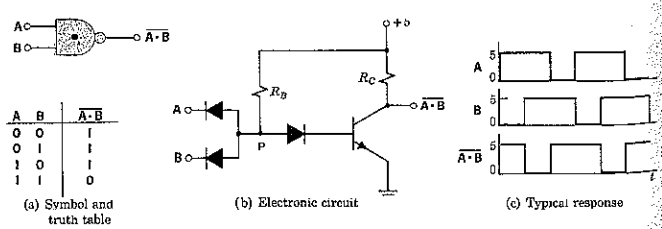
(a) Form the truth table for a **NOR** gate with the two input terminals tied together so that **A = B**. What logic function is performed by this gate?

(b) A logic circuit consists of a **NOR** gate with **NOT** gates inserted in each input line. Draw the circuit, form the truth table, and state the logic function performed.

*Answers:* (a) **NOT**; (b) **AND**.

**NAND** GATE. Diodes and a transistor can be combined to perform an inverted **AND** function. Such a **NAND** gate has all the advantages of a **NOR** gate and is very easy to fabricate, particularly in IC form. In a complex logic system, it is convenient to use just one type of gate, even when simpler types would be satisfactory, so that gate characteristics are the same throughout the system.

The **NAND** gate function is defined by the truth table in Fig. 13.7. The small circle on the **NAND** element symbol and the bar on the $\overline{\mathbf{A \cdot B}}$ output indicate the inversion process. With positive inputs at A and B (**1 AND 1**), the input diodes are reverse biased, and no input current flows; the positive base current supplied through $R_B$ causes heavy collector current and the output is approximately zero (**0**). If either A or B has a zero (**0**) input, at least one input diode conducts to ground, the voltage at point P is too low to supply base current to the transistor, hence no collector current flows, and the output is +5 V (**1**). For better separation of voltage levels (see Fig. 13.2d), a second diode may be placed in series with the base of the transistor.




(a) Symbol and truth table (b) Electronic circuit (c) Typical response

Figure 13.7 A diode–transistor **NAND** gate.

In Example 1 on p. 399, we see that three **NAND** gates can be used to replace an **OR** gate. The combination of **NAND** gates is equivalent to an **OR** gate in that it performs the same logic operation. In digital nomenclature, the function **f** is defined by

$$\mathbf{f = A + B = \overline{\overline{A} \cdot \overline{B}}}$$

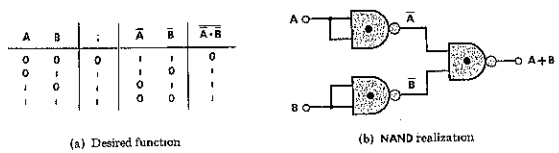


Example 1

Use **NAND** gates to form a two-input **OR** gate.

The desired function is defined by the truth table of Fig. 13.8a. Comparing this with Fig. 13.7a, we see that if each input were inverted (replaced by its complement), the **NAND** gate would produce the desired result as indicated in the truth table.

To provide simple inversion, we tie both terminals of a **NAND** gate together (the first and last rows of Fig. 13.7a). The desired logic circuit is shown in Fig. 13.8b.




(a) Desired function (b) **NAND** realization

Figure 13.8 Using **NAND** gates to form an **OR** gate.

We arrived at this relation by considering the desired and available truth tables. A "digital algebra" for the direct manipulation of such expressions is presented in Chapter 14. First, however, let us see how practical semiconductor logic elements function.

## 13-3 ELECTRONIC SWITCHES

Binary digital elements must respond to two-valued input signals and produce two-valued output signals. In practice the "values" are actually ranges of values separated by a *forbidden region*. The two discrete *states* may be provided electrically by switches, diodes, or transistors. Magnetic cores in which the two states are represented by opposite directions of magnetization were used extensively in early computers. Purely fluid switches are sometimes used in hydraulic control applications. In applications requiring high speed and flexibility, however, electronic switches predominate.

### CLASSIFICATION OF ELECTRONIC LOGIC

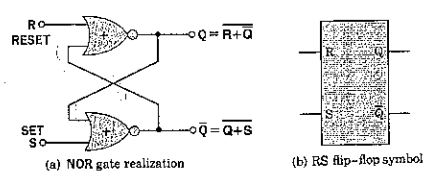
Electronic logic circuits are classified in terms of the components employed. Basic operations can be performed by diode logic (DL), resistor–transistor logic (RTL), or diode–transistor logic (DTL). Currently popular are transistor–transistor logic (TTL), metal–oxide–semiconductor (MOS) and complementary MOS (CMOS), and emitter-coupled logic (ECL). In specifying a logic system, the designer selects the type of logic whose characteristics match the requirements.

the decision-making process. For this reason, along with these *decision* components we need *memory* components to store instructions and results; the outputs of such *sequential* circuits are affected by past inputs as well as present.

What characteristics are essential in a memory unit? A binary storage device must have two distinct states, and it must remain in one state until instructed to change. It must change rapidly from one state to the other, and the state value (0 or **1**) must be clearly evident. The *bistable multivibrator* or *flip-flop*, a simple device that meets these requirements inexpensively and reliably, is used in all types of digital data processing systems.

## A LOGIC GATE MEMORY UNIT

First let us analyze a basic memory unit consisting of familiar logic gates. In the **NOR** gate flip-flop of Fig. 13.21, the output of each **NOR** gate is fed back into the input of the other gate. The operation is summarized in Table 13-1 where we assume to start that $\mathbf{Q}_0$, the "present" value of the output **Q**, at the time of the "Action," is **0** and inputs to the *set* terminal S and the *reset* terminal R are both **0**.




(a) NOR gate realization (b) RS flip-flop symbol

Figure 13.21 The flip-flop, a basic memory unit.

Note that $\overline{\mathbf{Q}}$ is normally meant to mean **NOT Q**, but in the discussion of RS flip-flops $\overline{\mathbf{Q}}$ is an output independent of **Q**. It is named $\overline{\mathbf{Q}}$ for the RS flip-flop because under most circumstances its value is the same as **NOT Q**. For other memory elements, $\overline{\mathbf{Q}}$ always implies **NOT Q**.

To **SET** the flip-flop, a **1** is applied to S only. For $\mathbf{Q}_0 = \mathbf{0}$, $\overline{\mathbf{Q}} = \overline{\mathbf{Q}_0 + \mathbf{S}} = \mathbf{0}$, and $\mathbf{Q} = \mathbf{R} + \overline{\mathbf{Q}} = \mathbf{1}$, the present state of the output is inconsistent with the input, the system is *unstable*, and **Q** must *flip*. After **Q** changes, $\mathbf{Q}_0$ (the present state of **Q**) changes to **1**, and $\overline{\mathbf{Q}}$ becomes $\mathbf{1} + \overline{\mathbf{1}} = \mathbf{0}$; hence $\mathbf{Q} = \overline{\mathbf{0} + \mathbf{0}} = \mathbf{1}$, a *stable* state. (In this analysis, keep in mind that *if either input to a* **NOR** *gate is* **1**, *the output is* **0**.) Removing the input from S causes no change. We conclude that $\mathbf{Q} = \mathbf{1}$ and $\overline{\mathbf{Q}} = \mathbf{0}$ is the stable state after being **SET**. Applying another input to S produces no change.

To **RESET** the flip-flop, a **1** is applied to R only. This results in an unstable system and **Q** must *flop* to **0**. (You should perform the detailed analysis.) A change in $\mathbf{Q}_0$ to **0** produces a stable output $\mathbf{Q} = \mathbf{0}$. Removing the input to R or applying another input to R produces no change. We conclude that $\mathbf{Q} = \mathbf{0}$ and $\overline{\mathbf{Q}} = \mathbf{1}$ is the stable state after being **RESET**.

Note that only a momentary input is required to produce a complete transition; this means that very short pulses can be used for triggering. Attempting to



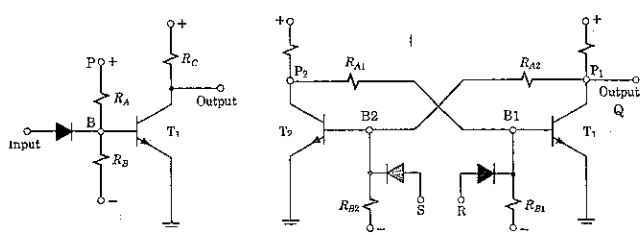
Table 13-1 Analysis of a Memory Unit

| Step | Action | Value of **Q** at Time of Action $\mathbf{Q}_0$ | Input S | Input R | Output Values after Action $\overline{\mathbf{Q}}$ | Output Values after Action **Q** | Conclusion |
|---|---|---|---|---|---|---|---|
| | Assume | **0** | **0** | **0** | i | **0** | This is a stable state |
| 1 | Apply **1** to S | **0** | **1** | **0** | **0** | i | Unstable state; **Q** changes |
| | $\mathbf{Q}_0$ becomes **1** | i | i | **0** | **0** | **1** | Stable |
| 2 | Remove **1** from S | i | **0** | **0** | **0** | i | Stable state after **SET** |
| 3 | Apply **1** to S again | **1** | **1** | **0** | **0** | i | No change in **Q** |
| 4 | Remove **1** from S | **1** | **0** | **0** | **0** | i | Stable state after **SET** |
| 5 | Apply **1** to R | i | **0** | **1** | **0** | **0** | Unstable state; **Q** changes |
| | $\mathbf{Q}_0$ becomes **0** | **0** | **0** | **1** | i | **0** | Stable |
| 6 | Remove **1** from R | **0** | **0** | **0** | i | **0** | Stable state after **RESET** |
| 7 | Apply **1** to both S and R | **0** | i | **1** | **0** | **0** | Unacceptable; $\overline{\mathbf{Q}}$ cannot equal **Q** |

**SET** and **RESET** simultaneously would create an ambiguous state with both **Q** and $\overline{\mathbf{Q}} = \mathbf{0}$. This state ambiguity would be unacceptable in a bistable unit and actual circuits are designed to avoid this condition. (See Fig. 13.24b.)

## A TRANSISTOR FLIP-FLOP

The operation of an electronic flip-flop is based on the switching properties of a transistor. With no input to the switch in Fig. 13.22a, the voltage divider $R_A$-$R_B$ reverse biases the base–emitter junction and the transistor is in cutoff or the switch is **OPEN**; because $I_C R_C = 0$, a positive voltage appears at the output terminal. If a positive signal is applied to the input, $V_B$ rises, the base–emitter junction is forward biased, the switch is **CLOSED**, and the output voltage drops to zero (nearly).




(a) Transistor switch (b) Elementary flip-flop

Figure 13.22 A transistor RS flip-flop.

An input signal applied to point P, making the + terminal of $R_A$ more positive, could also forward bias the base–emitter junction and drop the output voltage to zero. In other words, a **1** input at either of two terminals produces a **0** at the output. This switch is a form of **NOR** gate; a flip-flop can be created from two such switches.

To follow the operation of the flip-flop (Fig. 13.22b), assume that $T_1$ is conducting (**CLOSED**) and $T_2$ is cut off (**OPEN**). With $T_1$ conducting, the potential of point $P_1$ is nearly zero and, in combination with the negative voltage applied to $R_{B2}$, this ensures that $T_2$ is cut off. With $T_2$ cut off, the potential of point $P_2$ is large and positive, and this supplies the bias current through $R_{A1}$ that ensures that $T_1$ is conducting and $V_{P1}$ is low. In logic terms, **Q = 0**; this is a stable state that we may designate as the **0** state of this binary memory element.

A positive pulse applied to **RESET** terminal R that would raise $V_{B1}$ has no effect since $T_1$ is already conducting. However, a positive pulse at **SET** terminal S causes $T_2$ to begin conducting, the potential of $P_2$ drops, the forward bias on $T_1$ is reduced, the potential of $P_1$ rises, the forward bias on $T_2$ increases, the potential of $P_2$ drops further, $T_2$ goes into saturation, and $T_1$ is cut off. The output voltage is high, **Q = 1**, and this indicates another stable state that we may designate as the **1** state. If a flip-flop in the **1** state receives a positive pulse at R, transition proceeds in the opposite direction (since the device is symmetric) and the device is **RESET** to the **0** state. In a well-designed flip-flop, these changes in state take place in a few nanoseconds, and we see that this simple device satisfies all the requirements of a binary storage element.

Exercise 13-5

For the transistors of Fig. 13.22, $V_{CE(sat)} = 0.3$ V and $V_{CC} = 5$ V. Draw up a table showing the voltages at $P_1$ and $P_2$ for the following conditions: (a) $T_1$ initially conducting, (b) a negative pulse applied to S, (c) a positive pulse applied to S, (d) a negative pulse applied to R, (e) a positive pulse applied to R.

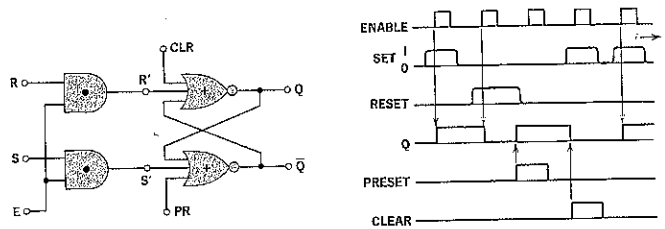
*Answers:* (a) 0.3, 5; (b) 0.3, 5; (c) 5, 0.3; (d) 5, 0.3; (e) 0.3, 5.

## TIMING WAVEFORMS

In the RS flip-flop of Fig. 13.21, a **1** input at the S input will **SET** the output **Q** to **1**. To **RESET** the flip-flop, a **1** is applied to input R. The duration of the input signal (as long as it exceeds a certain minimum time) and the time at which an input signal is applied are not significant. Such a flip-flop responds to *asynchronous* inputs.

A more sophisticated flip-flop incorporating two **AND** gates is shown in Fig. 13.23. Here an input is effective only when *enabled* by a **1** input at terminal E. In a digital system composed of many elements, it is usually necessary for the outputs of all elements to be synchronized. The synchronizing signal may come from a *clock*, and the enabling terminal is frequently designated **CLOCK (CK)**. In a clocked system, transitions cannot run wild through a circuit; instead, changes






(a) Block diagram (b) Timing waveforms

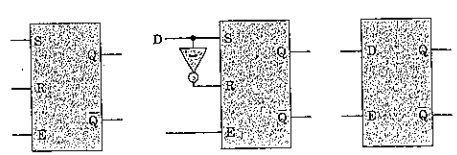
Figure 13.23 A more sophisticated RS flip-flop.

occur in an orderly, one-step-at-a-time fashion. In addition to the synchronous inputs **R** and **S**, there may be asynchronous inputs to *clear* or *preset* the flip-flop.

The operation of a *clocked* RS flip-flop is illustrated by the typical waveforms of Fig. 13.23b. Initially, output **Q = 0**. If a **1** appears at **SET**, when **ENABLE** goes to **1** the flip-flop is set with **Q = 1**. At the next clock pulse, the presence of a **1** at **RESET** forces the output to **0**. At any time, a **1** at **PRESET** forces the output to **1**; a **1** input at the **CLEAR** terminal overrides other inputs and forces **Q** to **0**.

## THE DATA LATCH

The functional symbol for a simple RS flip-flop (without **PRESET** and **CLEAR**) is shown in Fig. 13.24a. One way to avoid the ambiguous state where **R = 1** and **S = 1** simultaneously is the circuit modification shown in Fig. 13.24b. By connecting an inverter between the R and S terminals and using only one input signal, the ambiguity is avoided and the number of terminals is reduced (an advantage in IC packages). When the **ENABLE** line is **HIGH**, the output **Q** follows the input **D**. In other words, when this *flip-flop* is enabled, the input data is transferred to the output line. After the **ENABLE** line goes **LOW**, no change in **Q** is possible, and the output is "latched" at the previous data value. This *data latch* is widely used as an element in digital systems; for example, a set of eight such latches could "remember" the eight digits representing a number or an instruction. (See Example 5 on p. 416.)




(a) RS flip-flop (b) Modified RS flip-flop (c) Data latch

Figure 13.24 Deriving a data latch from an RS flip-flop.

Example 5

The enable and data inputs to a data latch are shown below. Predict the waveform of the output.

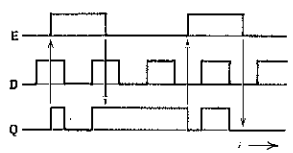


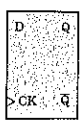

Figure 13.25 Typical data latch waveforms.

In the data latch, the output **Q** follows input **D** whenever enabled (**E** = **1**). When **E** goes to **0**, the output remains *latched* in the previous condition.

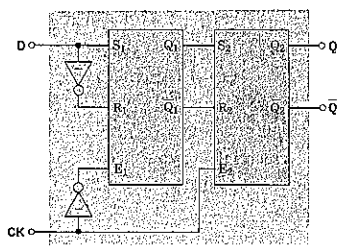
In Fig. 13.25, when **E** first goes **HIGH**, **D** = **1**; therefore, **Q** follows **D** and becomes **1**. As long as **E** is **HIGH**, **Q** follows any changes in **D**. When **E** goes **LOW**, **Q** = **D** = **1** and remains so. The output waveform is as shown.

## THE D FLIP-FLOP

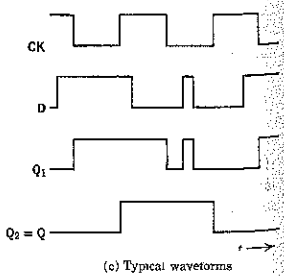
In digital systems it is sometimes desirable to delay the transfer of data from input to output. For example, we may wish to maintain a present state at **Q** while we read in a new state that will be transferred to the output at the appropriate time. The D (for *delay*) flip-flop† shown in Fig. 13.26 is a refinement of the data latch incorporating a second RS flip-flop. Here the data latch is enabled when the clock signal goes **LOW**, but the following RS flip-flop is enabled when **CLOCK** goes **HIGH**. In other words, $\mathbf{Q}_1$ follows **D** whenever **CK** is **LOW**, but any change in the output of the combination $\mathbf{Q} = \mathbf{Q}_2$ is delayed until the next upward transition of **CK**. This is an *edge-triggered* flip-flop; $\mathbf{Q}_1$ follows **D** while **CK** is **LOW**, then, on




(a) Symbol




(b) Block diagram




(c) Typical waveforms

Figure 13.26 The D flip-flop.

† "Flip-flop" is a general term applied to the basic two-transistor element and to sophisticated, multigate IC devices.

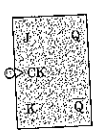


the leading edge of the clock pulse, the value of **D** is transferred to output **Q**. On the logic symbol, the small triangle indicates an edge-triggered device.

Because the output can change only at the instant that the clock goes **HIGH**, the output can be synchronized with the outputs of other elements. Furthermore, a sudden spurious change in **D** similar to that shown in Fig. 13.26c will not affect the output. For proper operation of a practical device, the data input must be stable for a few nanoseconds before the device is clocked (the *set-up time*), and it must remain stable for a few nanoseconds after the clocking is initiated (the *hold time*).

## THE JK FLIP-FLOP

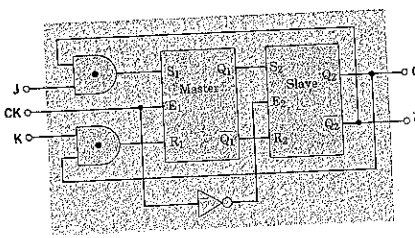
A widely used memory element is the JK flip-flop shown in Fig. 13.27. In its most common IC form, the output changes state on downward transitions of the clock pulse. The small circle on the symbol identifies this as a *trailing-edge-triggered* flip-flop. The operation of this logic element is improved by employing a *master* flip-flop that is enabled on the upward transition of the clock pulse while the *slave* flip-flop is inactive. Then the slave is enabled on the downward transition and follows its master; that is, it takes on the state of the immobilized master.




(a) Symbol

| J | K | Q |
|---|---|---|
| 0 | 0 | $Q_0$ |
| 0 | 1 | 0 |
| 1 | 0 | 1 |
| 1 | 1 | $\bar{Q}_0$ |

(b) Truth table




(c) Block diagram

Figure 13.27 The JK master–slave flip-flop.

In addition to avoiding the ambiguity referred to previously, this versatile device provides three different modes of response. Because of the feedback connections from output to input, the output of a JK flip-flop depends on the states of the inputs and the outputs at the instant the clock goes **LOW**. As indicated in the truth table, with **0** inputs at J and K, the clock has no effect, and the flip-flop remains in its present state $\mathbf{Q}_0$. With unequal inputs, the unit behaves like an RS flip-flop. For **J** = **1** and **K** = **0**, the clock **SETS** the flip-flop to **Q** = **1**; for **K** = **1** and **J** = **0**, the clock **RESETS** the flip-flop to **Q** = **0**. (In other words, with **J** ≠ **K**, **Q** = **J** or **Q** follows **J**.) With **1** inputs at both J and K, the flip-flop toggles; that is, the output changes each time the clock goes **LOW**. The operation of the JK flip-flop is revealed by a complete truth table constructed for the eight possible combinations of **J**, **K**, and $\mathbf{Q}_0$. (See Problem 13-34.)

Exercise 14-2

(a) Perform 38 − 24 and 24 − 38 by using signed 2's complement notation.
(b) Express decimal 541 as a binary-coded digit in the 8421 code.

*Answer:* (b) **0101 0100 0001.**

## 14-3 BOOLEAN ALGEBRA

We see that binary arithmetic and decimal arithmetic are similar in many respects. In working with logic relations in digital form, we need a set of rules for symbolic manipulation that will enable us to simplify complex expressions and solve for unknowns; in other words, we need a "digital algebra." Nearly 100 years before the first digital computer, George Boole, an English mathematician (1815–1864), formulated a basic set of rules governing the true–false statements of logic. Eighty-five years later (1938), Claude Shannon (at that time a graduate student at MIT) pointed out the usefulness of *Boolean algebra* in solving telephone switching problems and established the analysis of such problems on a firm mathematical basis. From our standpoint, Boolean algebra is valuable in manipulating binary variables in **OR**, **AND**, or **NOT** relations and in the analysis and design of all types of digital systems.

### BOOLE'S THEOREMS

The basic postulates are displayed in Tables 14-2 and 14-3. At first glance, some of the relations in the tables are startling. Properly read and interpreted, however, their validity is obvious. For example, "**1 OR 1**" (1 + 1) is just equivalent to **1**, and "**0 AND 1**" (**0 · 1**) is effectively **0**. In other words, for an **OR** gate with inputs of **1** at both terminals, the output is **1**, and for an **AND** gate with inputs of **0** and **1**, the output is **0**.

Table 14-2 Boolean Postulates in 0 and 1

| OR | AND | NOT |
|---|---|---|
| $0 + 0 = 0$ | $0 \cdot 0 = 0$ | $\overline{0} = 1$ |
| $0 + 1 = 1$ | $0 \cdot 1 = 0$ | $\overline{1} = 0$ |
| $1 + 0 = 1$ | $1 \cdot 0 = 0$ | |
| $1 + 1 = 1$ | $1 \cdot 1 = 1$ | |

Table 14-3 Boolean Theorems in One Variable

| OR | AND | NOT |
|---|---|---|
| $A + 0 = A$ | $A \cdot 0 = 0$ | $\overline{\overline{A}} = A$ |
| $A + 1 = 1$ | $A \cdot 1 = A$ | |
| $A + A = A$ | $A \cdot A = A$ | |
| $A + \overline{A} = 1$ | $A \cdot \overline{A} = 0$ | |

In general, the inputs and outputs of logic circuits are *variables;* that is, the signal may be present or absent (the statement is true or false) corresponding to the binary numbers **1** and **0**. The validity of the theorems in Table 14-3 may be reasoned out in terms of the corresponding logic circuit or demonstrated in a truth table showing all possible combinations of the input variables.



Example 5

Demonstrate the theorems

$A + \overline{A} = 1$ and $A \cdot 1 = A$

by constructing truth tables.

Using the postulates, the truth tables are

| A | $\overline{A}$ | $A + \overline{A}$ |
|---|---|---|
| 0 | 1 | 1 |
| 1 | 0 | 1 |

| A | $A \cdot 1$ |
|---|---|
| 0 | 0 |
| 1 | 1 |

Table 14-4 Boolean Theorems in More Than One Variable

| Commutation rules: | Association rules: | DeMorgan's theorems: |
|---|---|---|
| $A + B = B + A$ | $A + (B + C) = (A + B) + C$ | $\overline{A + B} = \overline{A} \cdot \overline{B}$ |
| $A \cdot B = B \cdot A$ | $A \cdot (B \cdot C) = (A \cdot B) \cdot C$ | $\overline{A \cdot B} = \overline{A} + \overline{B}$ |
| Absorption rules: | Distribution rules: | |
| $A + (A \cdot B) = A$ | $A \cdot (B + C) = (A \cdot B) + (A \cdot C)$ | |
| $A \cdot (A + B) = A$ | $A + (B \cdot C) = (A + B) \cdot (A + C)$ | |

Some of the more useful theorems in more than one variable are displayed in Table 14-4. The *commutation* rules indicate that the order of the variables in performing **OR** and **AND** operations is unimportant, just as in ordinary algebra. The *association* rules indicate that the order of the **OR** and **AND** operations is unimportant, just as in ordinary algebra. The second *distribution* rule indicates that variables or combinations of variables may be distributed in multiplication (not permitted in ordinary algebra) as well as in addition. The *absorption* rules are new and permit the elimination of redundant terms.

Example 6

Derive the absorption rule

$A + (A \cdot B) = A$

using other basic theorems.

| A | $A \cdot B$ | $A + (A \cdot B)$ |
|---|---|---|
| 0 | 0 | 0 |
| 1 | B | 1 |

Factoring by the second distribution rule,

$$A + (A \cdot B) = (A + A) \cdot (A + B) = A \cdot (A + B)$$

we see that the two absorption forms are equivalent.
Substituting $A \cdot 1$ for $A \cdot A$ (Table 14-3),

$$A \cdot (A + B) = A \cdot 1 + A \cdot B = A \cdot (1 + B) = A \cdot 1 = A$$

The truth table is shown at the left.

### DEMORGAN'S THEOREMS

Augustus DeMorgan, a contemporary of George Boole, contributed two interesting and very useful theorems. These concepts are easily interpreted in terms of logic circuits. The first says that a **NOR** gate ($\overline{A + B}$) is equivalent to an **AND** gate with **NOT** circuits in the inputs ($\overline{A} \cdot \overline{B}$). The second says that a **NAND** gate ($\overline{A \cdot B}$)

is equivalent to an **OR** gate with **NOT** circuits in the inputs ($\overline{A} + \overline{B}$). As generalized by Shannon, DeMorgan's theorems say:

> *To obtain the inverse of any Boolean function, invert all variables and replace all* **OR***s by* **AND***s and all* **AND***s by* **OR***s.*

Application of this rule is illustrated in Example 7.

Example 7

Use DeMorgan's theorems to derive a combination of **NAND** gates equivalent to a two-input **OR** gate.

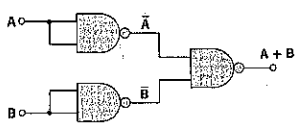



Figure 14.1 **OR** gate from **NAND** gates.

The desired function is $f = A + B$. Applying the general form of DeMorgan's theorems,

$$f = A + B = \overline{\overline{A} \cdot \overline{B}}$$

suggesting a **NAND** gate with **NOT** inputs.

Because $\overline{A \cdot A} = \overline{A}$, a **NAND** gate with the inputs tied together performs the **NOT** operation. The logic circuit is shown in Fig. 14.1.

Exercise 14-3

Use truth tables to prove Boole's theorem $A + (B \cdot C) = (A + B) \cdot (A + C)$ and DeMorgan's theorem $\overline{A + B} = \overline{A} \cdot \overline{B}$.

## LOGIC CIRCUIT ANALYSIS

By comparing Example 7 with Example 1 in Chapter 13, we see that the theorems of Boolean algebra permit us to manipulate logic statements or functions directly, without setting up the truth tables. Furthermore, the use of Boolean algebra can lead to simpler logic statements that are easier to implement. This result is important when it is necessary to design a circuit to perform a specified logic function using the available gates—only **NAND** gates, for example. DeMorgan's theorems are particularly helpful in finding **NAND** operations that are equivalent to other operations.

Up to this point we have been careful to retain the specific **AND** sign in $A \cdot B$ to focus attention on the *logic* interpretation. Henceforth, we shall use the simpler equivalent forms **AB** and **A(B)** whenever convenient. Also note that henceforth we shall follow convention and rely on the distinctive shapes of the logic symbols to identify their functions.

The *analysis* of a logic circuit consists in writing a logic statement expressing the overall operation performed in the circuit. This can be done in a straightforward manner, by starting at the input and tracing through the circuit, noting the function realized at each output. The resulting expression can be simplified or written in an alternative form using Boolean algebra. A truth table can then be constructed.



Example 8

Analyze the logic circuit of Fig. 14.2.

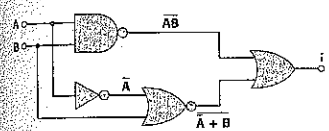



Figure 14.2 Logic circuit analysis.

Construct the truth table to demonstrate that this circuit could be replaced by a single **NAND** gate.

The suboutputs are noted on the diagram. The overall function can be simplified as follows:

$$f = \overline{AB} + \overline{\overline{A} + B}$$
$$= (\overline{A} + \overline{B}) + A\overline{B} \quad \text{(DeMorgan's rule)}$$
$$= \overline{A} + \overline{B}(1 + A) \quad \text{(Distribution)}$$
$$= \overline{A} + \overline{B} \quad (1 + A = 1)$$
$$= \overline{A\overline{B}} \quad \text{(DeMorgan's rule)}$$

| A | B | $\overline{AB}$ | $\overline{\overline{A} + B}$ | f |
|---|---|---|---|---|
| 0 | 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |

## 14-4 LOGIC CIRCUIT SYNTHESIS

One of the fascinating aspects of digital electronics is the construction of circuits that can perform simple mental processes at superhuman speeds. A typical digital computer can perform thousands of additions of 10-place numbers per second. The logic designer starts with a logic statement or truth table, converts the logic function into a convenient form, and then realizes the desired function by means of standard or special logic elements.

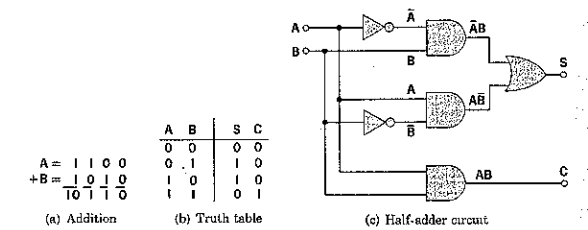
### THE HALF-ADDER

As an illustration, consider the process of addition. In adding two binary digits, the possible sums are as shown in Fig. 14.3a. Note that when **A** = 1 and **B** = 1, the *sum* in the first column is **0** and there is a *carry* of **1** to the next higher column. As indicated in the truth table, the half-adder must perform as follows: "**S** is **1** if **A** is **0 AND B** is **1, OR** if **A** is **1 AND B** is **0; C** is **1** if **A AND B** are **1**." In logic nomenclature, this becomes

$$S = \overline{A}B + A\overline{B} \quad \text{and} \quad C = AB \qquad (14\text{-}2)$$

(A *full-adder* is capable of accepting the carry from the adjacent column.)

To *synthesize* a half-adder circuit, start with the outputs and work backward. Equation 14-2 indicates that the sum **S** is the output of an **OR** gate; the inputs are




(a) Addition (b) Truth table (c) Half-adder circuit

Figure 14.3 Addition of two binary numbers.

obtained from **AND** gates; inversion of **A** and **B** is necessary. Equation 14-2 also indicates that the carry **C** is simply the output of an **AND** gate. The corresponding logic circuit is shown in Fig. 14.3c.

There may be several different Boolean expressions for any given logic statement, and some will lead to better circuit realizations than others. Algebraic manipulation of Eq. 14-2a yields

$$\begin{aligned}
\overline{A}B + A\overline{B} &= \overline{(A + B)(\overline{A} + \overline{B})} && \text{(DeMorgan's theorems)}\\
&= \overline{A\overline{A} + AB + \overline{A}\overline{B} + B\overline{B}} && \text{(Multiplication)}\\
&= \overline{\overline{A}\overline{B} + AB} && (A\overline{A} = 0 \text{ and } B\overline{B} = 0)\\
&= (A + B)(\overline{A} + \overline{B}) && \text{(DeMorgan's theorems)}\\
&= (A + B)\overline{AB} && \text{(DeMorgan's theorems)}
\end{aligned}$$

Looking again at the truth table (Fig. 14.3b), we see that another interpretation is: "**S** is **1** if (**A OR B**) is **1 AND (A AND B)** is **NOT 1**." Therefore binary addition can be expressed as

$$S = (A + B)\overline{AB} \quad \text{and} \quad C = AB \tag{14-3}$$

The synthesis of this circuit, working backward from the output, is shown in Fig. 14.4. This circuit is better than that of Fig. 14.3c in that fewer logic elements are used and the longest path from input to output passes through fewer *levels*. The ability to optimize logic statements is an essential skill for the logic designer.

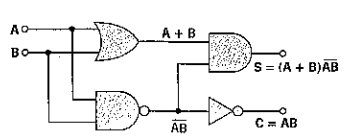



Figure 14.4 Another half-adder circuit.



## THE EXCLUSIVE–OR GATE

The function $(A + B)\overline{AB}$ in Eq. 14-3 is called the **Exclusive–OR** operation. As indicated by the truth table, it can be expressed as: "**A OR B** but **NOT (A AND B)**." The alternative form (Eq. 14-2a) $\overline{A}B + A\overline{B}$ is called an "inequality comparator" because it provides an output of **1** if **A** and **B** are not equal.

**Example 9**

Show that the inverse of the inequality comparator is an "equality comparator" and synthesize a suitable circuit.

Algebraic manipulation of the inverse function by using DeMorgan's theorem and Boole's theorem that $A \cdot \overline{A} = 0$ yields

$$\overline{\overline{A}B + A\overline{B}} = (A + \overline{B})(\overline{A} + B) = AB + \overline{A}\overline{B} \tag{14-4}$$

This is an equality comparator in that the output is **1** if **A** and **B** are equal. This function is highly useful in digital computer operation. Straightforward synthesis results in the circuit of Fig. 14.5.

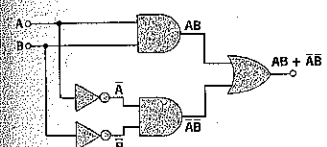
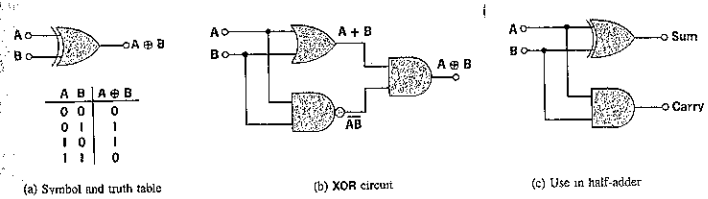



Figure 14.5 An equality comparator.

The **Exclusive–OR** gate is used so frequently that it is represented by the special symbol $\oplus$ defined by

$$A \text{ XOR } B = A \oplus B = (A + B)\overline{AB} \tag{14-5}$$

One realization of the **Exclusive–OR** gate is shown in Fig. 14.6b. Assuming that this gate is available as a logic element, the half-adder takes on the simple form of Fig. 14.6c. As a further simplification, we can treat the half-adder as a discrete logic element and represent it by a rectangular block labeled **HA**.

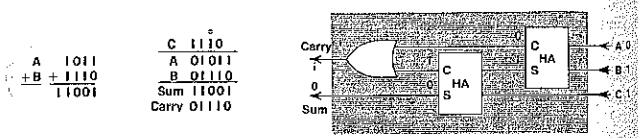



(a) Symbol and truth table (b) XOR circuit (c) Use in half-adder

Figure 14.6 An **Exclusive-OR** circuit and its application.

## THE FULL-ADDER

In adding two binary digits or *bits*, the half-adder performs the most elementary part of what may be highly sophisticated computation. To perform a complete addition, we need a *full-adder* capable of handling the carry input as well. The addition process is illustrated in Fig. 14.7a, where **C** is the carry from the preceding column. Each carry of **1** must be added to the two digits in the next column, so we need a logic circuit capable of combining three inputs. This operation can be realized using two half-adders and an **OR** gate. The values shown in Fig. 14.7b correspond to the third column of the addition. The operation of the full-adder can be seen more clearly by construction of a truth table. (See Review Question 12.)




(a) Addition process (b) Circuit realization

Figure 14.7 Design of a full-adder.

To perform the addition of "4-bit words," that is, numbers consisting of four binary digits, we need a half-adder for the first column and a full-adder for each additional column. In the *parallel binary adder*, the input to the half-adder consists of digits $A_1$ and $B_1$ from the first column. The input to the next unit, a full-adder, consists of carry $C_1$ from the first unit and digits $A_2$ and $B_2$ from the second column. The sum of the two numbers is represented by $C_4S_4S_3S_2S_1$. In subtraction, **NOT** gates provide the complement of the subtrahend (see Example 4, p. 435) and the process is reduced to addition.

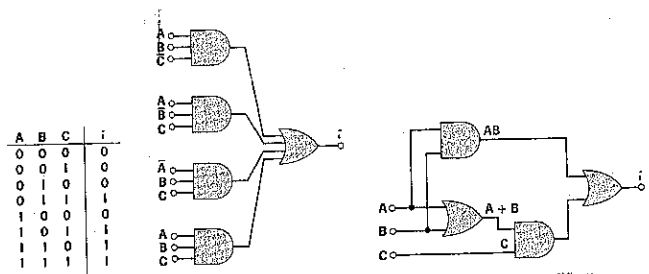
## A DESIGN PROCEDURE

A common problem in logic circuit design is to create a combination of gates to realize a desired function. The basic approach is to proceed from a statement of the function to a truth table and then to a Boolean expression of the function. The experienced logic designer then manipulates the Boolean expression into the simplest form. The realization of the final expression in terms of **AND**, **OR**, and **NOT** gates is straightforward.

For increased reliability on a spacecraft, triple sensing systems are used; no action is taken unless at least two of the three systems call for action. The truth table of the required *vote taker* is shown in Fig. 14.8a. Because the function is **YES(1)** only when a majority of the inputs are **YES**, the Boolean expression must contain a product term for each row of the truth table in which the function is 1. Because the function is **YES** for any one *or* more of these rows, the Boolean expression is

$$\mathbf{f = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC} \qquad (14\text{-}6)$$






(a) Truth table (b) Circuit realization (c) After Boolean simplification

Figure 14.8 Design of a vote taker.

Assuming that the complement of each variable is available, as is true in most computers, the straightforward realization is a combination of four **AND** gates feeding an **OR** gate. (See Fig. 14.8b.)

If the complements were not available, eight logic elements would be required and simplification of the circuit would be desirable. First, the Boolean function of Equation 14-6 can be expanded into

$$\mathbf{f = AB\bar{C} + A\bar{B}C + \bar{A}BC + ABC + ABC}$$

because $\mathbf{ABC + ABC = ABC}$. Factoring by the distribution rule yields

$$\mathbf{f = AB(\bar{C} + C) + C(AB + A\bar{B} + \bar{A}B)}$$

Because $\mathbf{\bar{C} + C = 1}$ and $\mathbf{AB + A\bar{B} + \bar{A}B = A + B}$, the function becomes

$$\mathbf{f = AB + C(A + B)}$$

This function requires only four logic elements (Fig. 14.8c).

Exercise 14-4

Given the logic function $\mathbf{f = \bar{A}B + A\bar{B}}$, apply DeMorgan's theorem to convert function **f** into a "**NAND**ed product of **NAND**s," and design a circuit consisting of **NAND** gates only, assuming complements are available.

*Answer:* $\mathbf{f = \overline{\bar{A}B} \cdot \overline{A\bar{B}}}$.

## 14-5 MINIMIZATION BY MAPPING

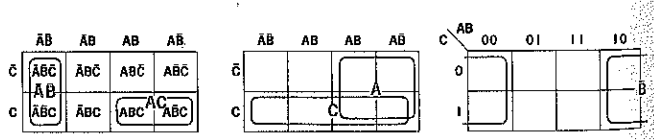
In creating a logic circuit to realize a desired function, the designer seeks the optimum form. The criterion may be maximum speed (fewest logic levels) or minimum cost (fewest gate leads because the number of leads determines the cost

of manufacture and the cost of assembly) or minimum design time (if only a few circuits are required). We have seen how Boolean algebra can be used to derive simpler logic expressions. If the truth table is available or if the logic function is expressed as a "sum of products," the designer can go directly to the minimal expression by the mapping technique suggested by Maurice Karnaugh.

## KARNAUGH MAPS

The *map* of the general logic function of three variables is shown in Fig. 14.9a. Each square in the map corresponds to one of the eight possible combinations of the three variables. The order of the columns is such that *combinations in adjacent squares differ only in the value of one variable*. Therefore, 2-square groups are independent of one variable; that is, $f_1 = \bar{A}B\bar{C} + \bar{A}BC = \bar{A}B$ and $f_2 = AB\bar{C} + ABC = AB$. These relations are easy to derive using Boolean algebra, but they are *obvious by inspection* of the Karnaugh map.

The concept can be extended to groupings of four adjacent squares as shown in Fig. 14.9b, where the labels are omitted from the squares. The 4-square cluster outlined in color is independent of both **B** and **C** and the 4-square in-line group is independent of both **A** and **B**; that is, $f_3 = \bar{A}\bar{B}C + \bar{A}BC + ABC + A\bar{B}C = C$. Enlarging groups by overlapping simplifies the terms. Note that the map is "continuous" in that the last column on the right is "adjacent" to the first column on the left. The 4-square group in Fig. 14.9c is just equal to $\bar{B}$.




(a) 2-square terms (b) 4-square terms (c) Another 4-square term

Figure 14.9 Mapping a three-variable logic function.

The standard labeling scheme for Karnaugh maps (Fig. 14.9c) is convenient for mapping from a truth table. Each square in the map corresponds to a row in the truth table. A specific logic function is *mapped* by placing a **1** in each square for which the function is 1. When this has been done, possible simplifications are easily recognized. This approach is illustrated in Example 10.

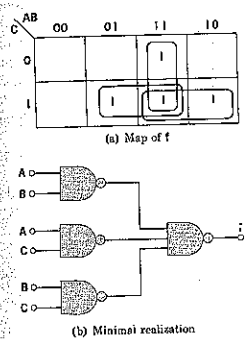
## MAPPING IN FOUR VARIABLES

The Karnaugh technique is even more valuable in simplifying functions in four variables. (For more than four variables, other techniques are usually more convenient.) As indicated in Fig. 14.11, 2-square groups are independent of one variable, 4-square groups are independent of two variables, and 8-square groups are independent of three variables. Note that the standard labeling scheme provides that adjacent rows differ by only one complement bar and the bottom row is *adjacent* to the top row. The four corner squares form a combination that is a little difficult to visualize.



Example 10

Map the vote-taker function and simplify the circuit realization, if possible.




(a) Map of f

(b) Minimal realization

From the truth table of Fig. 14.8a, 1s are placed in the squares corresponding to rows in the truth table for which the function is 1, representing

$$f = \bar{A}BC + A\bar{B}C + AB\bar{C} + ABC$$

All the 1s can be included in three overlapping 2-square groups; ∴ the complete function can be represented by

$$f = AB + AC + BC$$

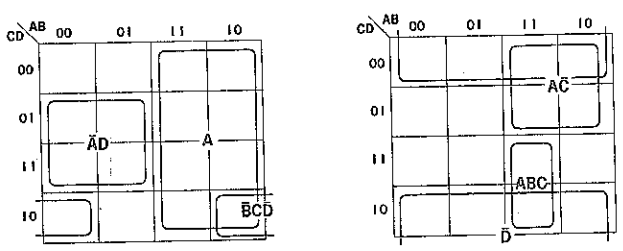
There is no simpler expression for this function.

By using DeMorgan's theorem, any "sum of products" can be converted to a "**NAND**ed product of **NAND**s." Here

$$f = \overline{\overline{AB} \cdot \overline{AC} \cdot \overline{BC}}$$

which can be synthesized using **NAND** gates only. Note that in the circuit realization of Fig. 14.10b, the number of gate leads has been reduced to 9 input + 4 output = 13 from the 21 in Fig. 14.8b. This vote taker could be realized in a single SSI chip.

Figure 14.10 Simplification by Karnaugh mapping.




(a) (b)

Figure 14.11 Examples of grouping on four-variable maps.

There are detailed rules† for finding the minimal expression from a Karnaugh map, but some general guidelines will suffice for our purposes:

*Include all* **1***s in groups of eight, four, two, or one.*
*Groups may overlap; larger groups result in simpler terms.*
*Of the possible combination of terms, select the simplest.*

The technique is illustrated in Example 11 on p. 446.

† See the references at the end of the chapter.

Example 11

Map the function

$f = \overline{AB}(\overline{C + D}) + \overline{A}C\overline{D} + AB\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}D$

and obtain a minimal sum-of-products expression.

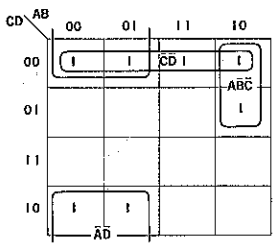



Figure 14.12 Four-variable simplification.

As an alternative to forming the truth table, let us map the function by considering the terms individually. The map of the function will be the "sum" of the maps of the individual terms.

$\overline{AB}$ limits the first term to the **00**, **01**, and **10** columns and $(\overline{C + D})$ corresponds to the first row, implying **1**s in the first, second, and fourth squares of the first row as shown in Fig. 14.12.

The $\overline{A}C\overline{D}$ term is independent of **B**, implying a 2-square group in the lower left-hand corner.

The four-variable terms imply **1**s in the **1100** and **1001** squares.

All the **1**s can be included in the two 4-square and one 2-square groups encircled. Therefore,

$$f = \overline{A}\,\overline{D} + \overline{C}\,\overline{D} + A\overline{B}\,\overline{C}$$

Other expressions are possible, but none will include fewer, simpler terms.

Two additional comments should be made. In some circuits, certain combinations of inputs never occur; such *don't care* combinations may be mapped as **X**s and considered as either **0**s or **1**s, whichever provides the greatest simplification. In other circuits, the simplest realization results from implementing $\overline{f}$ as a sum of products and then inverting to obtain **f**.

Exercise 14-5

Map the function $f = A\overline{B}CD + \overline{A}BC + \overline{A}\,\overline{B}C + BCD$ and find the minimal sum-of-products form.

*Answer:* $f = \overline{A}C + CD$.

The emphasis in this section is on synthesizing logic circuits from optimum arrangements of basic gates. This is the proper approach in designing complex circuits for mass production or in designing custom circuits requiring only a few gates. As we shall see, other approaches using standard IC packages are better where the cost of design time is an important factor.

## 14-6 REGISTERS

In addition to the logic circuits that process data, digital systems must include memory devices to store data and results. We know that a flip-flop can store or "remember" one digit of a binary number, one bit. A *register* is an array of flip-flops that can temporarily store data or information in digital form. For example, the 16-bit registers in a microprocessor, composed of sixteen flip-flops in parallel, can handle 2-byte instructions or 16-digit numbers. A great variety of registers is available in IC form.



### SHIFT REGISTERS

The sophistication of the response of the JK flip-flop makes it useful in computer applications. The *serial shift register* of Fig. 13.13 consists of four trailing-edge-triggered JK flip-flops connected so that $J \neq K$. (See Fig. 13.27.) At the trailing edge of each clock pulse, **Q** follows **J** in each flip-flop of the 4-bit register. The data are entered serially, that is, one bit at a time, and shifted right through the register at each clock pulse.

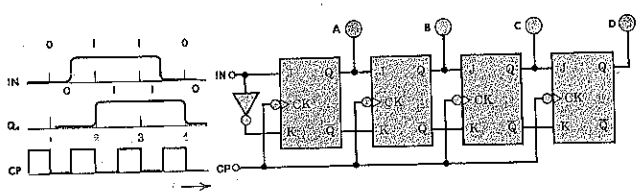



Figure 14.13 Entering **0110** into a 4-bit serial shift register.

Table 14-5 shows how **0110** would be placed in the register. We begin with the least significant bit. With a **0** at $IN = J_A$ and $K \neq J$, at the trailing edge of the first clock pulse (CP1), $Q_A$ follows $J_A$ and the LSB is transferred to the output of flip-flop A. During the next clock cycle, $J_B = Q_A = 0$ and the second bit, a **1**, is applied to $IN = J_A$. At CP2, the **0** is transferred to $Q_B$ (i.e., shifted one position to the right) and the **1** is transferred to $Q_A$. After four clock pulses, the 4-bit number is stored in the register and **0110** is available at parallel outputs **ABCD**. One application of such a register is as a *serial-to-parallel converter*, changing serial data to parallel form for processing all bits simultaneously. There is a single input line, but four lines are required for the parallel data output.

Table 14-5 Serial Shift Register

| CP | IN | $Q_A$ | $Q_B$ | $Q_C$ | $Q_D$ |
|---|---|---|---|---|---|
| 1 | 0 | 0 | | | |
| 2 | i | 1 | 0 | | |
| 3 | 1 | i | 1 | 0 | |
| 4 | 0 | 0 | 1 | 1 | 0 |

The shift register of Fig. 14.14 consists of D flip-flops with **CLEAR** and **PRESET** capabilities. It is similar to MSI TTL units available commercially. The symbols indicate that the flip-flops are cleared to **0** if **CLR** goes **LOW** while **PR** is

- A register is an array of flip-flops that can store binary numbers. In general, data may be entered and extracted in serial or parallel form. Practical IC registers may provide **CLEAR**, **PRESET**, and **ENABLE** capability.
- An array of flip-flops may be connected to function as a digital counter. Synchronous counters are faster but more expensive than ripple counters. Practical IC counters provide versatile operation at low cost.
- In read-and-write memory (RAM), the computer can store data at any location and retrieve it at any subsequent time; $k$ lines can address $2^k$ words. RAM units consist of a matrix of memory cells and an address decoder. MOS devices offer high density at low power; bipolar devices are faster.
- In read-only memory (ROM), data are permanently stored as cell states. ROMs can act as addressable memory, or code converter, or decision logic. PROMs are field-programmable ROMS; EPROMs are erasable PROMs.
- In a bus-organized system, a multiplexer selects a source and puts its data on the bus; a demultiplexer transfers the data to a selected destination.
- A digital computer is a complex array of logic and memory elements organized to perform binary calculations at high speed. Computers include input, memory, control, processing, and output sections.

## TERMS AND CONCEPTS

**adder** Logic circuit that outputs the sum of two binary digits.

**binary** Two-valued; using the base-2 numbering system.

**bit** Binary digit; a one or a zero.

**Boolean algebra** Set of algebraic rules for describing and manipulating binary variables.

**bus** Common path for information transfer.

**byte** An 8-bit sequence of binary digits.

**computer** Machine designed to accept data, perform operations according to instructions, and present the results.

**counter** Sequential circuit that keeps count of input pulses.

**DeMorgan's theorems** Set of algebraic rules useful in simplifying logic circuits.

**digital computer** Computer that operates on discrete data by performing binary arithmetic and logic processes.

**EPROM** Erasable programmable read-only memory that can be programmed and erased repeatedly.

**Karnaugh map** Graphic display of a logic function that leads directly to the minimal expression.

**memory** Organized array of addressable elements, each capable of storing one bit of information.

**program** Complete set of instructions to a computer.

**PROM** Read-only memory that can be programmed irreversibly after manufacture.

**RAM** Read-and-write memory into which data can be written and from which data can be read.

**ROM** Read-only memory containing fixed data that can be read but not changed.



## REFERENCES

J. A. Aseltine et al., *Introduction to Computer Systems*, Wiley, New York, 1989.

T. C. Bartee, *Digital Computer Fundamentals*, 7th Edition, McGraw–Hill, New York, 1990.

D. J. Comer, *Digital Logic and State Machine Design*, Holt, Rinehart & Winston, New York, 1990.

D. V. Hall, *Digital Circuits and Systems*, McGraw–Hill, New York, 1989.

J. Watson, *Analog and Switching Circuit Design*, Wiley, New York, 1989.

## REVIEW QUESTIONS

1. Why is the binary system useful in electronic data processing?
2. How can decimal numbers be converted to binary? Binary to decimal? To octal?
3. Explain "2's complement" notation. What are its advantages?
4. What is a "binary-coded decimal"? What is decimal 476 as a BCD in the 8421 code?
5. What is Boolean algebra and why is it useful in digital computers?
6. Which of the Boolean theorems in **0** and **1** contradict ordinary algebra?
7. Explain the association and distribution rules for Boolean algebra.
8. State DeMorgan's theorems in words. When are they useful?
9. How can a logic circuit be analyzed to obtain a logic statement?
10. How can a logic statement be synthesized to create a logic circuit?
11. What function is performed by a half-adder? An **Exclusive–OR** gate?
12. Explain how binary numbers are added in a parallel adder.
13. Explain how we can design a **NAND** circuit from a truth table.
14. Explain the principle behind the numbering of columns in a Karnaugh map.
15. How can a shift register serve as a serial-to-parallel converter?
16. What is meant by a "bus-organized" system?
17. Explain the operation of a binary ripple counter.
18. Distinguish between RAM and ROM and PROM.
19. How many address lines are required to address a 1-kilobyte memory?
20. Explain the **WRITE** operation in a MOS RAM.
21. Explain the **READ** operation in a bipolar ROM.
22. Differentiate between a multiplexer and a decoder.

## PROBLEMS

**P14-1.** (a) Write in binary the following decimals: 4, 12, 23, 31.
(b) Write in decimal the following binaries: **00011, 00101, 01010, 10101.**

**P14-2.** (a) Write in binary the following decimals: 10, 14, 21, 39, 75.
(b) Write in decimal the following binaries: **00111, 01110, 11010, 101011.**
(c) Convert decimals 0.875 and 0.65 to binaries.

**P14-3.** Given four numbers: (a) $165_{10}$, (b) **11001101B**, (c) **316Q**, and (d) $\mathbf{D8}_{16}$. Convert each to equivalent numbers in three different notations.

**P14-4.** Repeat Problem 14-3 for: (a) $85_{10}$, (b) **11100101B**, (c) **62Q**, and (d) $\mathbf{6F}_{16}$.

**P14-5.** Repeat Problem 14-3 for: (a) $103_{10}$, (b) **01101110B**, (c) **207Q**, and (d) $\mathbf{A5}_{16}$.

**P14-6.** (a) Convert the number **161Q** to binary, decimal, hex, and BCD.
(b) Convert the number $\mathbf{5F}_{16}$ to binary, decimal, octal, and BCD.

**P14-7.** Convert the BCD **10010011** to decimal, binary, and octal.

**P14-8.** (a) Using signed 2's complement notation, express as 8-bit words the decimal numbers 25, 121, −17, and −96.
(b) Show in binary notation the arithmetic operations: 121 − 25, 25 + (−17), and 25 − 96.
(c) Write a brief explanation of how subtraction is accomplished using 2's complement notation.

**P14-9.** (a) Using signed 2's complement notation, express as 8-bit words the decimal numbers 37, 92, −30, and −48.
(b) Show in binary notation the arithmetic operations: 92 − 37, 37 + (−30), and 37 − 48.
(c) Write a brief explanation of how subtraction is accomplished using 2's complement notation.

**P14-10.** In general, what is the 2's complement of the 2's complement of binary number **A**?

**P14-11.** Use truth tables to prove the following Boolean theorems:
(a) $\mathbf{A + 1 = 1}$
(b) $\mathbf{A + B = B + A}$
(c) $\mathbf{BC + AC = (A + B)C}$
(d) $\mathbf{A(BC) = (AB)C}$

**P14-12.** Use Boolean theorems to prove the following identities:
(a) $\mathbf{A + \overline{A}B = A + B}$
(b) $\mathbf{ABC + AB\overline{C} = AB}$
(c) $\mathbf{(A + \overline{B})B = AB}$
(d) $\mathbf{(A + B)(\overline{A} + C) = AC + \overline{A}B}$
(e) $\mathbf{(A + C)(A + D)(B + C)(B + D) = AB + CD}$

**P14-13.** When writing equations for "programmed array logic" circuits, complicated expressions must be broken down into simple "sums-of-products" (like Eq. 14-6).
(a) Write the following expression as a sum-of-products. Show the Boolean theorems used during each step of simplification.

$$[(\mathbf{A \cdot B \cdot \overline{C}}) \cdot \overline{(\mathbf{A \cdot \overline{B} \cdot C})} + \mathbf{\overline{A}\overline{B}}] \cdot \mathbf{\overline{D}}$$

(b) Invert the result from part (a), and factor it into a sum-of-products, showing theorems used.

**P14-14.** The function $\mathbf{f = A + B}$ is to be realized using only **NAND** gates. Use DeMorgan's theorems to express **f** in terms of $\mathbf{\overline{C} \cdot \overline{D}}$ where **C** and **D** can be expressed in terms of **A** and **B**. Draw the necessary logic circuit and check by constructing the truth table.

**P14-15.** Analyze the logic circuit of Fig. P14.15 and determine **f** in terms of **A** and **B**. Simplify using Boolean algebra and check your result with a truth table.

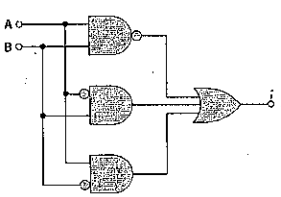



Figure P14.15

**P14-16.** (a) What single gate is equivalent to the circuit of Fig. P14.16a? Check your answer using Boolean algebra.
(b) Repeat part (a) for the circuit of Fig. P14.16b.
(c) What theorem is illustrated in parts (a) and (b)?

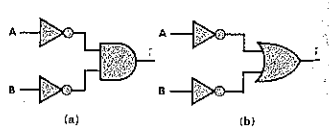



Figure P14.16

**P14-17.** The 8212 I/O (input/output) port consists of 8 data latches with noninverting three-state buffers as shown in Fig. P14.17. When this port (Device) is to be used (Selected) in the output Mode, the microprocessor places 8 bits of data on the **DI** lines and sends device select signal **DS = 1** and control signals **M = 1** and **S** (Strobe) = **1**. Assuming **S = 1** at all times:
(a) Construct a truth table showing **M**, **DS**, **CK**, and **E**.



(b) Explain the operation of this "output latch" when **M = 1**.
(c) Explain the operation of this "gated buffer" when **M = 0**.

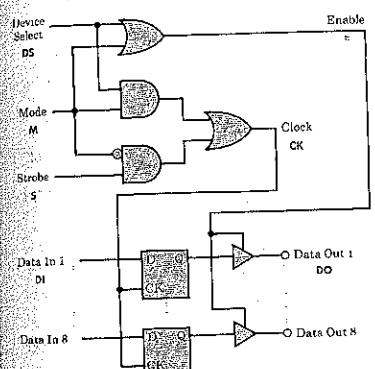



Figure P14.17

**P14-18.** To provide a comparison of the most common logic operations, construct a table with inputs **A** and **B** and outputs **AND**, **NAND**, **OR**, **NOR**, **Exclusive–OR**, and **Equality Comparator.**

**P14-19.** Synthesize logic circuits to realize the following functions as written (i.e., do not perform Boolean simplification).
(a) $\mathbf{\overline{\overline{A} + \overline{B} + A \cdot B + \overline{A} + \overline{B}}}$
(b) $\mathbf{\overline{(\overline{A \cdot B} + A \cdot B) \cdot \overline{A} + \overline{B}}}$
(c) $\mathbf{\overline{(A + B) + \overline{A}\overline{B} \cdot \overline{A}\overline{B}}}$
(d) $\mathbf{\overline{A + \overline{B} \cdot \overline{A}\overline{B}} + AB}$

**P14-20.** Given the logic function

$$\mathbf{f = \overline{\overline{AB} + \overline{A}\overline{B}} + \overline{A}B}$$

(a) Assuming the complements are available, simplify the function using DeMorgan's theorem and synthesize it using the basic gates.
(b) Assuming the complements are not available, simplify the function and synthesize it from five **NAND** gates.

*Answer:* (a) $\mathbf{f = AB + BA}$.

**P14-21.** Using DeMorgan's theorem, convert the vote-taker expression (Eq. 14-6) to a **NAND** expression and synthesize it using **NAND** gates only.

**P14-22.** Map the following functions and find the minimal sum-of-products form:
(a) $\mathbf{ABC + AB\overline{C} + A\overline{B}\overline{C}}$
(b) $\mathbf{A\overline{B} + ABC + \overline{A}\overline{B}}$
(c) $\mathbf{(A + \overline{C})(\overline{B} + \overline{C})}$
(d) $\mathbf{ABC + \overline{B}C + A\overline{B}\overline{C}}$

**P14-23.** Evaluate the combination of the four corner squares of a four-variable Karnaugh map.

**P14-24.** Map the following functions and find the minimal sum-of-products form:
(a) $\mathbf{AB\overline{C}\overline{D} + A\overline{B}C + \overline{B}\overline{C}}$
(b) $\mathbf{AB + \overline{A}\overline{B}CD + A\overline{B}C}$
(c) $\mathbf{\overline{A}(C + D) + A\overline{B}C + A\overline{B}\overline{C}\overline{D}}$
(d) $\mathbf{A\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}\overline{D} + \overline{B}\overline{C}D + \overline{B}C}$

**P14-25.** Decimal numbers coded in binary are to drive the display of Fig. P14.25. (*Note:* The anodes of all LED segments are connected to +5 V; a segment is lit when its cathode is taken **LOW**, that is, logic **0**.) You are to design a "BCD-to-seven segment" code converter.
(a) Draw up a truth table showing decimal numbers, their binary codes, and the state of each segment.
(b) How many decoder input and output lines are required?
(c) Segment *a* should be lit for certain input numbers. Write the logic expression for segment *a* in terms of input variables **DCBA** (**A** is LSB).
(d) Simplify the logic expression and synthesize it using basic logic gates.

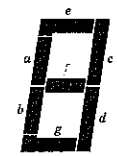



Figure P14.25

**P14-26.** The array of delay flip-flops in Fig. P14.26 serves as a three-bit register.
(a) For **LOAD LOW**, what are the D values?
(b) For **LOAD HIGH**, what are the D values?
(c) Describe the operation of this register.
(d) For the input waveforms shown, draw the waveform of $\mathbf{Q}_A$, the first bit of output.