# Digital Principles

# 1

OBJECTIVES

- **Understand the difference between analog and digital signals, recognize binary equivalents of decimal numbers 0 to 15, and be familiar with basic terminology related to digital waveforms.**
- **Based on input conditions, determine the output of a buffer, a tri-state buffer, an inverter, a tri-state inverter, an AND gate, and an OR gate.**
- **Discuss several ways of how digital information (bits) can be stored and transferred and describe some of the operations of an ALU.**
- **Recognize digital logic symbols and identify fundamental difference, in operation and logic levels between major IC families.**

In the modern world of electronics, the term *digital* is probably most often associated with a *computer*. It certainly is difficult to think of an area of life today that is not influenced in one way or another by a digital computer. Checking and savings accounts at a bank, automobile insurance, credit card accounts, federal and state income taxes, airline tickets—the list of functions controlled by large computer systems seems almost endless! In addition to these large systems, the hand calculator, the IBM or IBM clone personal computer (PC), the Apple family of computers, and a host of other desktop computer systems are readily available at a reasonable cost to virtually anyone. The availability of such computational power can be traced directly to the development of the digital integrated circuit (IC).

The semiconductor industry provided the first commercially available families of digital ICs in the early 1960s. These devices were used to develop smaller, faster, more economical, and more powerful digital computers. They were also used in a great many other applications. Today, digital circuits and systems can be found in almost every field of electronics. In communications, the principles of digital electronics are found in satellites, telephone switching and transmission networks, and navigation systems. Digital circuits in the area of consumer electronics are found in compact discs, VCRs, and television. Process controls in industrial applications, and electronic systems used in medicine have benefited greatly from advances in digital electronics. The list will no doubt continue to expand.

An introduction to the field of digital electronics cannot cover all possible applications, but a collection of basic principles can be identified. These *digital principles* are the basis for this text, and they along with a number of *applications* are intended to provide the background for you to succeed in the modern world of digital electronics.

This chapter presents some distinctive features of the world of digital electronics. It defines digital signals and terminologies associated with digital waveform; discusses digital logic and basic operations on digital data; introduces the concept of digital IC, its signal levels and noise margin.

## 1.1 DEFINITIONS FOR DIGITAL SIGNALS

### Analog versus Digital

Electronic circuits and systems can be conveniently divided into two broad categories generally referred to as *analog* and *digital*. Analog circuits, designed for use with small signals, can be made to work in a linear fashion. An operational amplifier (op amp) connected as an amplifier with a voltage gain of 10 is an analog circuit. The output voltage for this circuit will be a faithfully amplified version of any signal presented at its input till saturation is reached. This is *linear operation*. Digital circuits are generally used with *large signals* and are considered nonlinear. One example is a remote control circuit that switches the lights in a parking area on after sunset and turns them off at sunrise. In this case, the input signal might be a voltage representing the time of day or it might be a current taken from a light-sensing circuit. The output signal is simply *on* or *off*, which is clearly not an amplified version of the input signal. This is *nonlinear operation*.

Any quantity that changes with time either can be represented as an analog signal or it can be treated as a digital signal. For example, place a container of water at room temperature on a stove and apply heat. The measurable quantity of interest here is the change in water temperature. There are two ways to record the water temperature over a period of time. In Fig. 1.1a, the temperature is recorded *continuously*, and it changes smoothly from 20°C to 80°C. While being heated, the water temperature passes through every possible

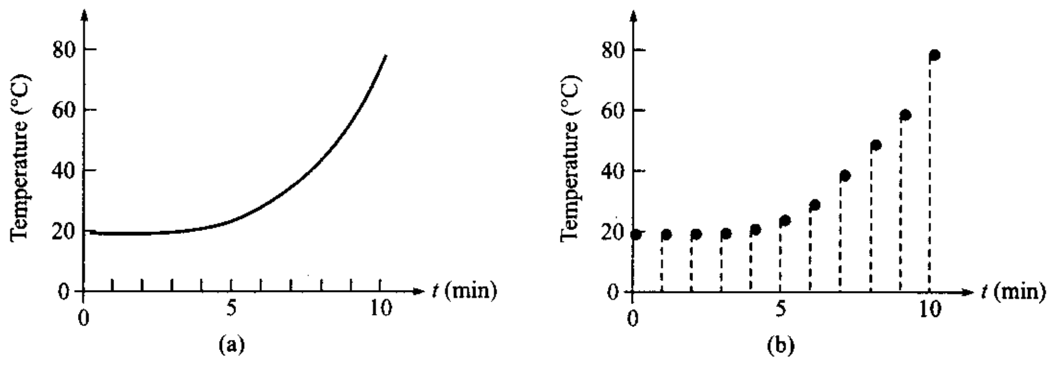


**Fig. 1.1** (a) An analog (continuous) signal, (b) A digital (discrete) signal

value between 20°C and 80°C. This is an example of an analog signal. *Analog signals* are continuous and all possible values are represented.

If the water temperature is measured and recorded only once every minute, the temperature is recorded as in Fig. 1.1b. In this case, the recorded temperature is *not* continuous. Rather, it *jumps* from point to point, and there are only a finite number of values between 20°C and 80°C say, at an increment of 1°C like 20°C, 21°C, 22°C, and so on. There are exactly 11 values in this case. When a quantity is recorded as a series of distinct (discrete) points, it is said to be *sampled*. This is an example of a digital signal. *Digital signals* represent only a finite number of discrete values.

Virtually all naturally occurring physical phenomena are analog signals. Temperature, pressure, velocity, and sound, for instance, are signals that take on all possible values between given limits. These signals can be conditioned and operated on with the use of analog electronic circuits. For example, an analog power amplifier is used to amplify a music signal and drive a set of stereo speakers. Digital circuits and systems can be used to process both analog signals and digital signals. As an example, both music and speech are readily translated into digital signals for use in a digital stereo system. At the same time, digital circuits can be used to perform functions unrealizable with analog circuits—counting, for instance.

## Binary System

Digital electronics today involves circuits that have exactly two possible states. A system having only two states is said to be *bi*nary (*bi* means "two"). The *binary number system* has exactly two symbols—0 and 1. As you might expect, the binary number system is widely used in uigital electronics. We will consider the binary number system in much greater detail later, but for immediate reference, the first 16 binary numbers and their decimal equivalents are shown in Table 1.1.

Table 1.1

| *Decimal* | *Binary* | *Decimal* | *Binary* |
|---|---|---|---|
| 0 | 0000 | 8 | 1000 |
| 1 | 0001 | 9 | 1001 |
| 2 | 0010 | 10 | 1010 |
| 3 | 0011 | 11 | 1011 |
| 4 | 0100 | 12 | 1100 |
| 5 | 0101 | 13 | 1101 |
| 6 | 0110 | 14 | 1110 |
| 7 | 0111 | 15 | 1111 |

The operation of an electronic circuit can be described in terms of its voltage levels. In the case of a digital circuit, there are only two. Clearly one voltage is more positive than the other. The more positive voltage is the *high (H)* level, and the other is the *low (L)* level. This is immediately related to the binary number system by assigning $L = 0$ and $H = 1$. Many functions performed by digital circuits are *logical operations*, and thus the terms true ($T$) and false ($F$) are often used. Choosing $H = 1 = T$ and $L = 0 = F$ is called *positive logic*. The majority of digital systems utilize positive logic. Note that it is also possible to construct a *negative logic* system by choosing $H = 0 = F$ and $L = 1 = T$.

Today the majority of digital circuit families utilize a single +5 Vdc power supply, and the two voltage levels used are +5 Vdc and 0 Vdc. Here is a summary of the two binary states (levels) in this positive logic system.

$$+5\text{ Vdc} = H = 1 = T$$
$$0\text{ Vdc} = L = 0 = F$$

You can no doubt see how to extend these definitions to include terms such as *on–off go–no go, yes–no*, and so on. A lamp or a *light-emitting* diode (LED) is frequently used to indicate a digital signal. *On* (illuminated) represents 1, and *off* (extinguished) represents 0. As an example, the four LEDs in Fig. 1.2 are indicating the binary number 0101, which is equivalent to decimal 5.

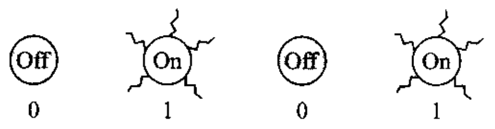


Fig. 1.2 The binary number 0101 (decimal 5)

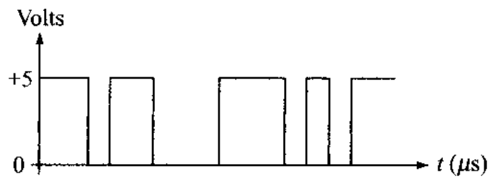


Fig. 1.3 An ideal digital signal

### Ideal Digital Signals

The voltage levels in an *ideal* digital circuit will have values of either +5 Vdc or 0 Vdc. Furthermore, when the voltages change (switch) between values, they do so in zero time!

These concepts are illustrated in Fig. 1.3, which represents an arbitrary digital signal whose level changes with time. As we shall see in Sec. 1.2, *actual* digital signals depart somewhat from this ideal.

**SELF-TEST**

1. Analog signals are (continuous, discrete).
2. The operation of a digital circuit is generally considered to be nonlinear. (T or F)
3. Write the binary number for the decimal number 7.
4. A certain digital circuit is designed to operate with voltage levels of −0.2 Vdc and −3.0 Vdc. If $H = 1 = -0.2$ Vdc and $L = 0 = -3.0$ Vdc, is this positive logic or negative logic?

## 1.2 DIGITAL WAVEFORMS

The ideal digital signal represented in Fig. 1.3 has two precise voltage levels—+5 Vdc and 0 Vdc. Furthermore, the signal switches from one level to the other in zero time. In reality, modern digital circuits can produce signals that approach, but do not quite attain, this ideal behavior.

### Voltage Levels

First of all, the output voltage level of any digital circuit depends somewhat on its load, as illustrated in Fig. 1.4a below. When $V_0$ is high, the voltage should be +5 Vdc. In this case, the digital circuit must act as a *current source* to deliver the current $I_o$ to the load. However, the circuit may not be capable of delivering the necessary current $I_o$ while maintaining +5 Vdc. To account for this, it is agreed that any output voltage close to +5 Vdc within a certain range will be considered high. This is illustrated in Fig. 1.4c, where any output voltage level between +5 Vdc and $V_{\text{OH,min}}$ is defined as $H = 1 = T$. The term $V_{\text{OH,min}}$ stands for the minimum value of the output voltage when high. As we will see, one popular transistor-transistor logic (TTL) family of digital circuits allows $V_{\text{OH,min}} = +3.5$ Vdc. In this case, any voltage level between +5 Vdc and +3.5 Vdc is $H = 1$.

In Fig. 1.4b. $V_o$ is low, and the digital circuit must act as a *current sink*. That is, it must be capable of accepting a current $I_o$ from the load and delivering it to ground. In this situation, $V_o$, should be 0 Vdc, but the

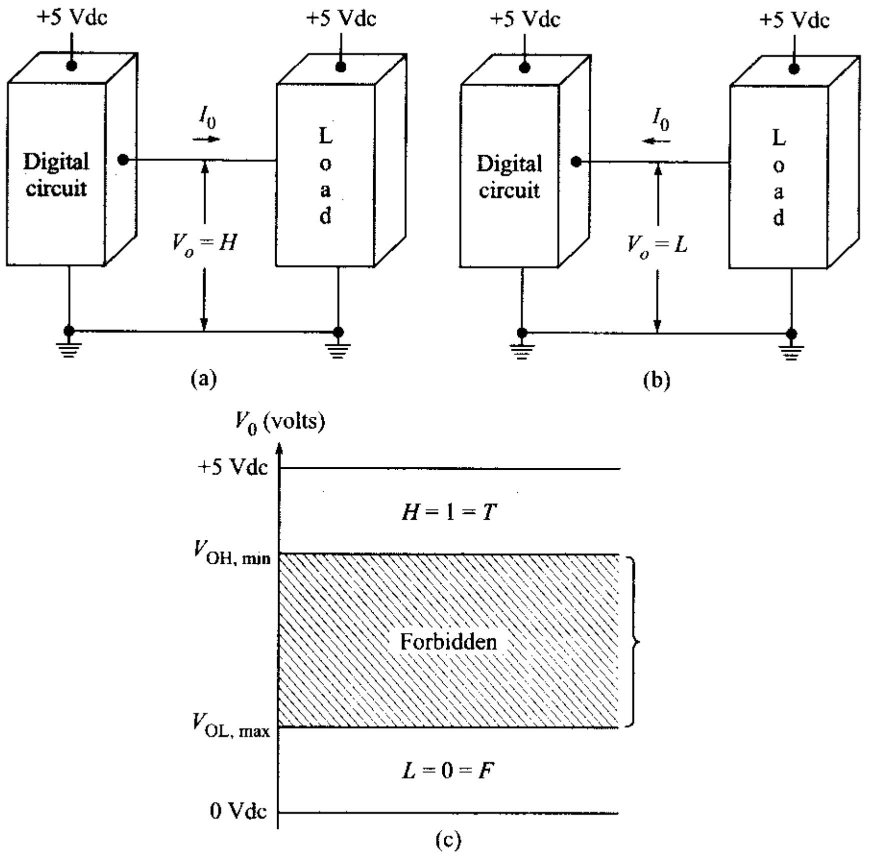


**Fig. 1.4** **Loading of digital circuit**

circuit may not be capable of this. So it is agreed to accept any output voltage that is close to 0 Vdc within certain limits as the low level. This is illustrated in Fig. 1.4c, where any output voltage level between 0 Vdc and $V_{OL,max}$, is defined as $L = 0 = F$. The term $V_{OL,max}$ stands for the *maximum* value of the output voltage when low. Again, the popular TTL family mentioned above allows $V_{OL,max} = +0.1$ Vdc. Thus, any voltage level between +0.1 Vdc and 0 Vdc is $L = 0$.

The diagram in Fig. 1.4c clearly shows that the digital signals being used here require a high-level voltage somewhere in the band labeled $H$. A low-level voltage must be somewhere in the band labeled $L$. Furthermore, *no other voltage levels are permitted!*

## Switching Time

If the digital circuit in Fig. 1.4 were *ideal*, it would change from high to low, or from low to high, in zero time. Thus, the output voltage would never have a value in the forbidden range. In reality, it does, in fact, require a finite amount of time for $V_o$ to make the transition (switch) between levels. As a result, the voltage $V_o$ versus time might appear as in Fig 1.5a. Clearly $V_o$ does take on values in the forbidden range between the high and low band—but only for a very short time, and only while switching! When it is not switching, $V_o$ remains within the high or the low band as required.

The time required for $V_o$ to make the transition from its high level to its low level is defined as *fall time* $t_f$. For ease of measurement, it is customary to measure fall time using $0.9H$ and $1.1L$, as shown in Fig. 1.5a.

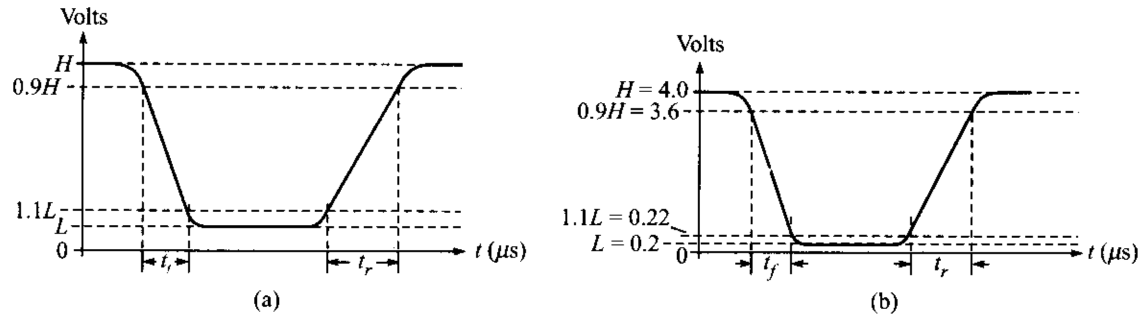


Fig. 1.5 Switching in digital circuit

The time required for $V_0$ to make the transition from its low level to its high level is defined as *rise time* $t_r$. Again, rise time is measured between $1.1L$ and $0.9H$, as illustrated in Fig. 1.5a.

Figure 1.5b illustrates how rise time and fall time are measured. For example, suppose $H = +4.0$ Vdc and $L = +0.2$ Vdc. Then, $0.9H = 0.9 \times 4.0 = +3.6$ Vdc, and $1.1L = 1.1 \times 0.2 = +0.22$ Vdc. The rise and fall times are then measured between these two voltage levels as shown.

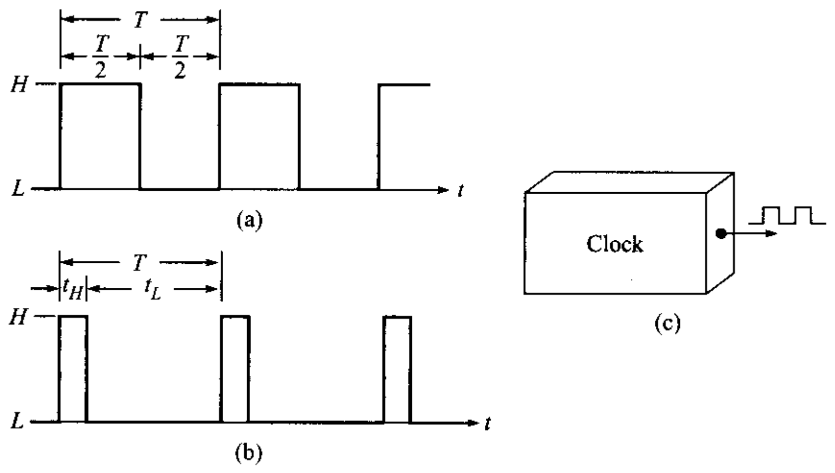


Fig. 1.6 (a) Symmetrical signal with period T, (b) Asymmetrical signal with period T, (c) System clock

## Period and Frequency

There are many occasions where a symmetrical digital signal as in Fig. 1.6a will be used (clock and counter circuits for instance). The period $T$ of this waveform is shown. This is the time over which the signal repeats itself. A rectangular waveform such as this can be produced by adding together an infinite (or at least a large number) of sinusoidal waveforms of different frequencies and amplitudes. Even though this digital signal is not sinusoidal, it is convenient to define the frequency as $f = 1/T$. As an example, if the period of this square wave is 1 μs, then its frequency is found as

$$f = \frac{1}{T} = \frac{1}{1\ \mu s} = \frac{1}{10^{-6}} = 10^6 = 1 \text{ MHz}$$

The digital waveform in Fig. 1.6b is not a square wave; that is, it is not symmetrical. Even so, its frequency is still found as the reciprocal of its period i.e. $f = 1/T$.

A symmetrical signal as illustrated in Fig. 1.6a or Fig. 1.6b is frequently used as the basis for timing all operations in a digital system. As such, it is called the *clock signal.* The electronic circuit used to generate this square wave is referred to as the *system clock,* as illustrated in Fig. 1.6c. A system clock is simply an oscillator circuit having a very precise frequency. Frequency stability is provided by using a crystal as the frequency-determining element.

## Duty Cycle

*Duty cycle* is a convenient measure of how symmetrical or how unsymmetrical a waveform is. For the waveform in Fig. 1.6b, there are two possible definitions for duty cycle.

$$\text{Duty cycle } H = \frac{t_H}{T}$$

$$\text{Duty cycle } L = \frac{t_L}{T}$$

The first definition is the fraction of time the signal is high, and the second is the fraction of time the signal is low. Either definition is acceptable, provided you clearly define which you are using. To express as a percentage, simply multiply by 100.

Note that the duty cycle for a symmetrical wave as in Fig. 1.6a is

$$\text{Duty cycle } H = \text{Duty cycle } L = \frac{T/2}{T} = 0.5 \text{ or } (50\%)$$

**Example 1.1** The waveform in Fig. 1.6b has a frequency of 5 MHz, and the width of the positive pulse is 0.05 $\mu s$. What is the high duty cycle (Duty cycle $H$)?

*Solution* The period of the waveform is found as

$$T = \frac{1}{f} = \frac{1}{(5 \text{ MHz})} = \frac{1}{5 \times 10^6} = 0.2\ \mu s$$

Then

$$\text{Duty cycle } H = \frac{0.05\ \mu s}{0.2\ \mu s} = 0.25 = 25\%$$

**SELF-TEST**

5. Refer to Fig. 1.4c and describe the meaning of the terms $V_{OH,min}$ and $V_{OL,mix}$.
6. Can $V_o$ ever have a value within the *forbidden* band in Fig. 1.4c? Explain.
7. In Fig. 1.5a, $H = +5.0$ Vdc and $L = +1.0$ Vdc. What are the voltage levels between which the rise and fall times are measured?
8. What is the value of Duty cycle $H$ if the waveform in Fig. 1.6b is high for 2 ms and low for 5 ms?

## 1.3 DIGITAL LOGIC

### Generating Logic Levels

The digital voltage levels described in Fig. 1.4 can be produced using switches as illustrated in Fig. 1.7 on the next page. In Fig. 1.7a, the switch is down and $V_o = L = 0 = 0$ Vdc. When the switch is up, as in Fig. 1.7b, $V_o = H = 1 = +5$ Vdc. A switch is easy to use and easy to understand, but it must be operated by hand.

A *relay* is a switch that is actuated by applying a voltage $V_i$ to a coil as shown in Fig. 1.7c. The coil current develops a magnetic field that moves the switch arm from one contact to the other. This is indicated with the dashed line drawn between the coil and the relay arm. For this particular relay, $V_o = L = 0$ Vdc when $V_i = 0$ Vdc. Applying a voltage $V_i$ will actuate the relay, and then $V_o = H = +5$ Vdc. This relay could of course be connected so that its output is low when actuated.

Switches and relays were useful in the construction of early machines used for calculation and/or logic operations. In fact, they are still used to a limited extent in modern computer systems where humans must interact with a system. For instance, on–off power switches, reset, start–stop, and load–unload are functions that might require human initiation.

On the other hand, modern computers are capable of performing billions of switching operations every second! Switches and relays are clearly not capable of this performance, and they have therefore been replaced by transistors (bipolar and/or MOSFET). A *digital integrated circuit (IC)* is constructed using numerous transistors and resistors, and each is designed to perform a given logic operation. On an IC, each transistor is used as an electronic switch. Let's use the simple switch models shown in Fig. 1.7 to define some basic logic circuits. Later on, we'll take the time to look at the actual ICs and discuss circuit operation in detail.

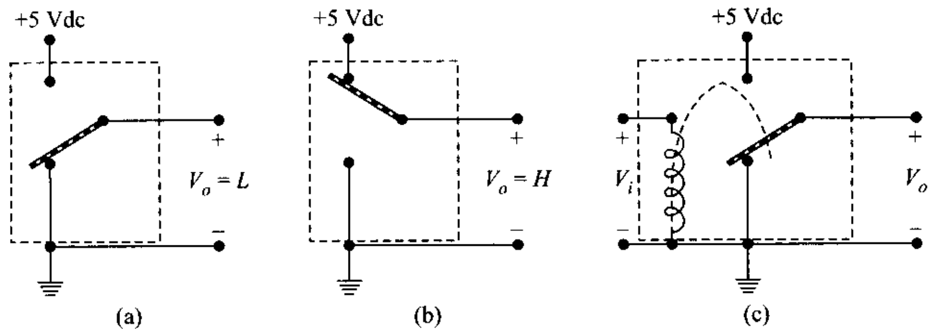
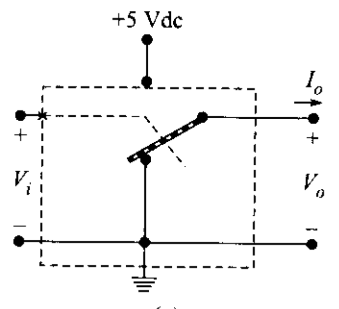


Fig. 1.7 (a) Switch, (b) Switch, (c) Normally low relay

### The Buffer

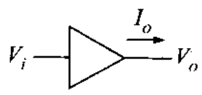
In order to deliver the necessary load current $I_o$ in Fig. 1.4, a digital IC called a *buffer* might be used. A buffer can be thought of as an electronic switch, as shown in Fig. 1.8a. The switch is actuated by the input voltage $V_i$. Its operation is similar to the relay in Fig. 1.7c. When $V_i$ is low, the switch is down, and $V_o$ is low. On the other hand, when $V_i$ is high, the switch moves up and $V_o$ is high. Operation of this IC is summarized by using the truth table, or table of combinations, shown in Fig. 1.8b. There are only two possible input voltage levels ($L$ and $H$), and the truth table shows the value of $V_o$ in each case.



(a)

| $V_i$ | $V_o$ |
|---|---|
| $L$ | $L$ |
| $H$ | $H$ |

(b)



(c)

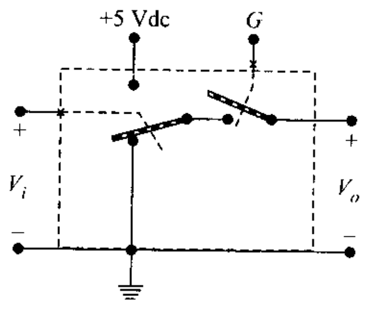
Fig. 1.8 (a) Buffer amplifier model, (b) Truth table, (c) Symbol

Since the buffer is capable of delivering additional current to a load, it is often called a *buffer amplifier.* The traditional amplifier symbol (a triangle) shown in Fig. 1.8c is used on schematic diagrams. If you're interested in an actual IC buffer, look in the standard TTL logic family. The 5407 or 7407 is a 14-pin IC that contains six buffers.

## The Tri-State Buffer

At the input of a digital system, there may be more than one input signal of interest. Generally speaking, however, it will be necessary to connect only one signal at a time, and thus there is a requirement to connect or disconnect (switch) input signals electronically. Similarly, the output of a digital system may need to be directed to more than one destination, one at a time.

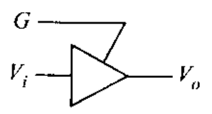
The logic circuit in Fig. 1.9a is a simple buffer with an additional switch controlled by an input labeled $G$. When $G$ is low, this switch is open and the output is "disconnected" from the buffer. When $G$ is high, the switch is closed and the output follows the input. That is, the circuit behaves as an ordinary buffer amplifier. In effect, the control signal $G$ connects the buffer to the load or disconnects the buffer from the load.



(a)

| $V_i$ | $G$ | $V_o$ |
|---|---|---|
| $L$ | $L$ | Open |
| $H$ | $L$ | Open |
| $L$ | $H$ | $L$ |
| $H$ | $H$ | $H$ |

(b)



(c)

Fig. 1.9 A tri-state buffer: (a) Model, (b) Truth table, (c) Symbol

The truth table in Fig. 1.9b summarizes circuit operation. Notice that when $G$ is high, $V_o$ is either high or low (two states). However, when $G$ is low, the output is in effect an *open circuit* (a third state). Since there are *three* possible states for $V_o$, this circuit is called a *tri-state buffer.* (*Tri* stands for "three", and thus the term *three-state buffer* is often used.)

The standard symbol for a tri-state buffer such as this is shown in Fig. 1.9c. It is simply the buffer symbol with an additional input, *G*. Since *G* controls operation of the circuit, it is often referred to as the *enable input.* In the standard TTL logic family, a 54126 or a 74126 is a 14-pin IC that has four of these circuits.

## The Inverter

One of the most basic operations in a digital system is *inversion*, or *negation*. This requires a circuit that will invert a digital level. This logic circuit is called an *inverter*, or sometimes a *NOT circuit*. The switch arrangement in Fig. 1.10a is an inverter. When the input to this circuit is low, the switch remains up and the output is high. When the input is high, the switch moves down and the output is low. The truth table for the inverter is given in Fig. 1.10b Clearly the output is the negative, or the inverse, of the input.

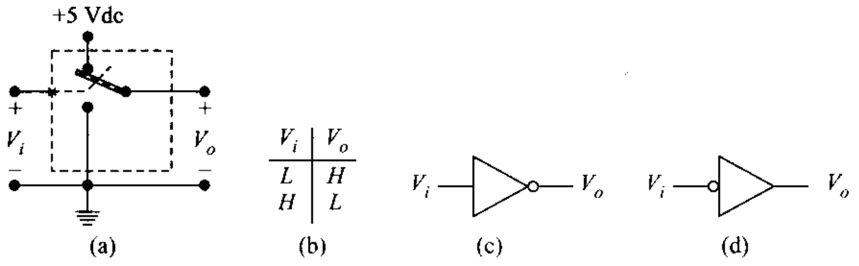


**Fig. 1.10** **Digital inverter: (a) Model, (b) Truth table, (c) Symbol, (d) Another symbol**

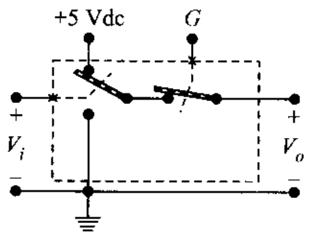
When the inverter is used as a logic circuit, *H* is often defined as the "true" state, while *L* is defined as the "false" state. In this sense, the inverter will always provide at its output a signal that is the inverse, or complement, of the signal at its input. It is thus called a *negation* or NOT circuit. This makes sense, since there are only two possible states, and therefore NOT *H* must be *L* and NOT *L* must be *H*.

The inverse or complement of a signal is shown by writing a bar above the symbol. For instance, the complement of *A* is written as $\bar{A}$ or $A'$ and this is read as "*A* bar" A logic expression for the inverter in Fig. 1.10c is $V_o = \bar{V_i}$. It is read "*V* sub oh is equal to *V* sub eye bar."

The standard symbol for an inverter is given in Fig. 1.10c. Notice the small circle (bubble) at the output. This small circle signifies inversion, and it is used on many other logic symbols. For instance, the symbol in Fig. 1.10d has the small circle on the input side. This is still an inverter, but the circle on the input side has additional significance, which will be considered next. In the standard TTL logic family, a 5404 or 7404 is a 14-pin IC with six inverters.

## The Tri-state Inverter

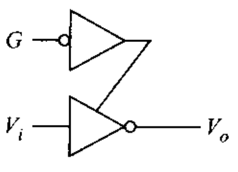
A *tri-state inverter* is easy to construct, as shown in Fig. 1.11a. The truth table in Fig. 1.11b shows that when *G* is low, the inverter is connected to the output. When *G* is high, the enable switch opens, and the output is disconnected from the inverter. The standard logic symbol for this tri-state inverter is given in Fig. 1.11c. The inverting amplifier symbol indicates that $V_o$ is the inverse of $V_i$ (the small circle is at the amplifier output). However, note the small circle at the input of the amplifier used for *G*. From the truth table, you can see that the switch controlled by *G* is closed when *G* is low! Thus, when *G* is low, the circuit is activated and output $V_o$ is the inverse of the input $V_i$. Compare this with the *G* input to the tri-state buffer in Fig. 1.9. In this case, the switch is closed and the circuit is activated when *G* is high. Here, then, is the significance of the circle on the input side: *Placing a circle at the input of a logic circuit means that circuit is activated when the* *signal at that input is low*! The tri-state inverters used on the 74LS386A IC (TTL logic family) are similar to Fig. 1.11.



(a)

| $V_i$ | $G$ | $V_o$ |
|---|---|---|
| $L$ | $L$ | $H$ |
| $H$ | $L$ | $L$ |
| $L$ | $H$ | Open |
| $H$ | $H$ | Open |

(b)



(c)

**Fig. 1.11 Inverting tri-state buffer: (a) Model, (b) Truth table, (c) Symbol**

## The AND Gate

An *AND gate* is a digital circuit having two or more inputs and a single output as indicated in Fig. 1.12. The inputs to this gate are labeled $V_1$, $V_2$, $V_3$, ... $V_n$ (there are $n$ inputs), and the output is labeled $V_o$. The operation of an AND gate can be expressed in a number of different, but equivalent, ways. For instance,

1. If *any* input is low, $V_o$ will be low.
2. $V_o$ will be high only when all inputs are high.
3. $V_o = H$ only if $V_1 = H$, and $V_2 = H$, and $V_3 = H$, . . . and $V_n = H$.

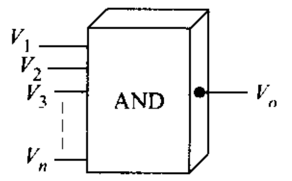
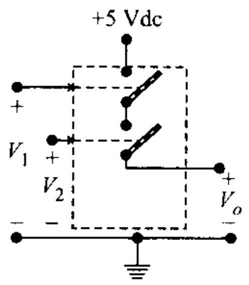


**Fig. 1.12 AND gate**

This last statement leads to the designation *AND gate*, since $V_1$ *and* $V_2$, *and* $V_3$, ... *and* $V_n$ must all be high in order for $V_o$ to be high.

A model for an AND gate having 2 inputs is shown in Fig. 1.13a. This gate can be used to make "logical" decisions; for example, "If $V_1$ and $V_2$, then $V_o$." As a result, it is referred to as a digital logic circuit, as are all AND gates. From the model, it is seen that $V_1 = H$ closes the upper switch, and $V_2 = H$ closes the lower switch. Clearly, $V_o = H$ only when both $V_1$ and $V_2$ are high. This can be expressed in the form of a logic equation written as

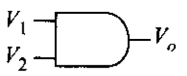
$$V_o = V_1 \text{ AND } V_2$$



(a)

| $V_1$ | $V_2$ | $V_o$ |
|---|---|---|
| $L$ | $L$ | $L$ |
| $H$ | $L$ | $L$ |
| $L$ | $H$ | $L$ |
| $H$ | $H$ | $H$ |

(b)



(c)

**Fig. 1.13 Two-input AND gate: (a) Model, (b) Truth table, (c) Symbol**

The operation is summarized in the truth table in Fig. 1.13b. The symbol for a 2-input AND gate is shown in Fig. 1.13c. Thus, AND is a logic operation which is realized here through a logic gate.

## The OR Gate

An *OR gate* is also a digital circuit having 2 or more inputs and a single output as indicated in Fig. 1.14. The inputs to this gate are labeled $V_1$, $V_2$, $V_3$, ... $V_n$, (there are $n$ inputs), and the output is labeled $V_o$. The operation of an OR gate can be expressed in a number of different ways. For instance,

1. $V_o$ will be low only when all inputs are low.
2. If any input is high, $V_o$ will be high.
3. $V_o = H$ if $V_1$, *or* $V_2$ *or* $V_3$, ... *or* $V_n = H$.

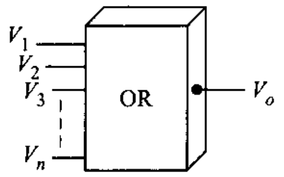


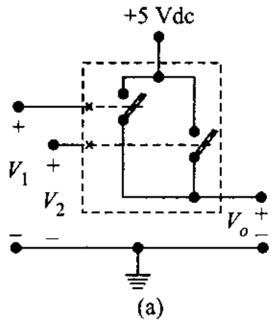
**Fig. 1.14 OR gate**

This last statement leads to the designation OR gate, since $V_o = H$ only if $V_1$ *or* $V_2$, *or* $V_3$, ... $V_n = H$.

A model for an OR gate having 2 inputs is shown in Fig. 1.15a. This gate can be used to make "logical" decisions; for example, "If $V_1$ or $V_2$, then $V_o$." As a result, it is referred to as a digital logic circuit, as are all OR gates. From the model, it is seen that $V_1 = H$ closes the upper switch, and $V_2 = H$ closes the lower switch. Clearly, $V_o = H$ if either $V_1$ or $V_2$ is high. This can be expressed in the form of a "logic" equation written as

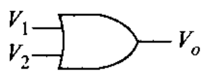
$$V_o = V_1 \text{ OR } V_2$$

The operation is summarized in the truth table in Fig. 1.15b. The symbol for a 2-input OR gate is shown in Fig. 1.15c. Thus, OR is a logic operation which is realized here through a logic gate.



| $V_1$ | $V_2$ | $V_o$ |
|---|---|---|
| $L$ | $L$ | $L$ |
| $H$ | $L$ | $H$ |
| $L$ | $H$ | $H$ |
| $H$ | $H$ | $H$ |

(b)



(c)

**Fig. 1.15 Two-input OR gate: (a) Model, (b) Truth table, (c) Symbol**

**SELF-TEST**

9. Refer only to the tri-state buffer symbol in Fig. 1.9c and determine the State of $V_o$ if both $G$ and $V_i$ are low. Check your response with the truth table in Fig. 1.9b. Repeat if both $G$ and $V_i$ are high.
10. Refer only to the inverting tri-state buffer symbol in Fig. 1.11c and determine the state of $V_o$ if both $G$ and $V_i$ are low. Check your response with the truth table in Fig. 1.11b. Repeat if both $G$ and $V_i$ are high.

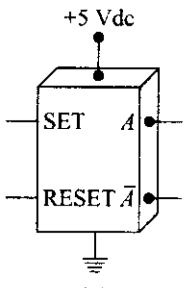
11. For the AND gate in Fig. 1.13c, $V_1 = H$ and $V_2 = L$. What is the state of $V_o$?
12. If the AND gate in Fig. 1.13c had an additional input ($V_3$), and $V_1 = V_2 = H$, and $V_3 = L$, what would be the state of $V_o$? What would it take to produce $V_o = H$?
13. If the OR gate in Fig. 1.15c had an additional, input ($V_3$), and $V_1 = V_2 = H$, and $V_3 = L$, what would be the state of $V_o$? What would it take to produce $V_o = L$?

## 1.4 MOVING AND STORING DIGITAL INFORMATION

### Memory Elements

A *digital memory element* is a device or perhaps a circuit that will maintain a desired logic level at its output till it is changed by changing the input condition. The simplest memory element is the switch shown in Figs. 1.7a and b. The switch in Fig. 1.7a is placed such that its output is low, and it will remain low without any further action. Thus, it will "remember" that $V_o = L$. Since $L = 0 = 0$ Vdc, the switch can be thought of as "holding" or "storing" a logic 0. In Fig. 1.7b. $V_o = H$. and it will remain high without any further action. The switch remembers that $V_o = H$. In this case, the switch is holding or storing a logic 1, since $H = 1 = +5$ Vdc. It is easy to see that this switch can be used to store a digital level, and it will remember the stored level indefinitely.

The simplest electronic circuit used as a memory element is called a *flip-flop*. Since a flip-flop is constructed using transistors, its operation depends upon dc supply voltage(s) as seen in Fig. 1.16a. The flip-flop can be used to store a logic level (high or low), and it will retain a stored level indefinitely provided the dc supply voltage is maintained. An interruption in the dc supply voltage will result in loss of the stored logic level. When power is first applied to a flip-flop (turning the system on first thing in the morning), it will store either a high or a low. This is a "random" result, and it must be accounted for in any digital system. Generally, a signal such as MASTER RESET or power-on reset will be used to *initialize* all storage elements.



(a)

| SET | RESET | $A$ |
|---|---|---|
| $H$ | $L$ | $H$ |
| $L$ | $H$ | $L$ |
| $L$ | $L$ | No change |
| $H$ | $H$ | ???(not allowed) |

(b)

**Fig. 1.16 (a) A flip-flop, (b) Truth table**

The truth table in Fig. 1.16b can be used to explain the operation of this flip-flop. The two inputs are SET and RESET, and the output is $A$. The output labeled $\bar{A}$ is simply the *inverse* of $A$. Here's how it works:

1. When SET = $H$ and RESET = $L$, the flip-flop is *set*, and $A = H$.
2. When SET = $L$ and RESET = $H$, the flip-flop is reset, and $A = L$.

3. Holding SET = $L$ and RESET = $L$ disables the flip-flop and its output remains unchanged.
4. Applying SET = $H$ and RESET = $H$ at the same time is not allowed, since this is a request to *set* and *reset* at the same time—an impossible request!

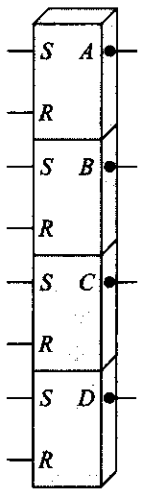
To summarize, when the flip-flop is SET, it stores a high (a logic 1). When it is RESET, it stores a low (a logic 0). A simple flip-flop such as this is often called a *latch*, since its operation is similar to a switch. A 7475 is an IC in the TTL family that contains four similar flip-flops.

## Registers

A group of flip-flops can be connected together to store more than a single logic level. For instance, the four flip-flops in Fig. 1.17 can be used to store four logic levels. As such, they could be used to store any of the ten binary numbers given in Table 1.1. As an example, if $A$ is SET, $B$ is RESET, $C$ is SET, and $D$ is RESET, this will store the binary number $DCBA$ = $LHLH$ = 0101, which is equivalent to decimal 5.

When we speak of decimal numbers, each position in a number is called a *decimal digit*, or simply a digit. For example, the decimal number 847 has three digits. When we speak of binary numbers, each position in the number is called a *binary digit*, or *bit*. (The term "binary digit" has been shortened to "bit.") For example, the binary number 0101 is composed of four bits; it is a 4-bit binary number. The four flip-flops in Fig. 1.17 can be used to store any 4-bit binary number.

A group of flip-flops used to store a binary number is called a *register*, or sometimes a *storage register*. The register in Fig. 1.17 is a 4-bit register. There are eight flip-flops in an 8-bit register, and so on. In the TTL family, the 74198 is an 8-bit register. Clearly a register can be used to store decimal numbers in their binary equivalent form. In general, binary numbers such as this are referred to as *data*. A register is a fundamental building block in a microprocessor or digital computer, and you can now see the beginnings of how these systems are used for computation.



$S$ = SET $R$ = RESET

**Fig. 1.17 A four-bit register**

The register in Fig. 1.18a has 8 inputs, 1 through 8, and 8 outputs, $a$ through $h$. It is constructed using eight flip-flops and some additional electronic circuits. A binary number is stored in this register by applying the appropriate level (high or low) at each input *simultaneously*. Thus one bit is "shifted" into each flip-flop in the register. The binary number is said to be shifted into the register *in parallel*, since all bits are entered at the same time. In this case, the binary number (or data) is entered in one single operation. Once a number is stored in this register, it appears immediately at the 8 outputs, $a$ through $h$. A 74198 is an example of an 8-bit *parallel register*.

The register in Fig. 1.18b has a single input and a single output. It is also constructed using eight flip-flops and some additional electronic circuits. It will store an 8-bit binary number, but the number must be entered into the register one bit at a time at the input. It thus requires eight operations to store an 8-bit number. This is how it is done. The first bit of the binary number is entered in flip-flop $A$ at the input. The second bit is then entered into flip-flop $A$, and at the same time the first bit in flip-flop $A$ is passed along (shifted) to flip-flop

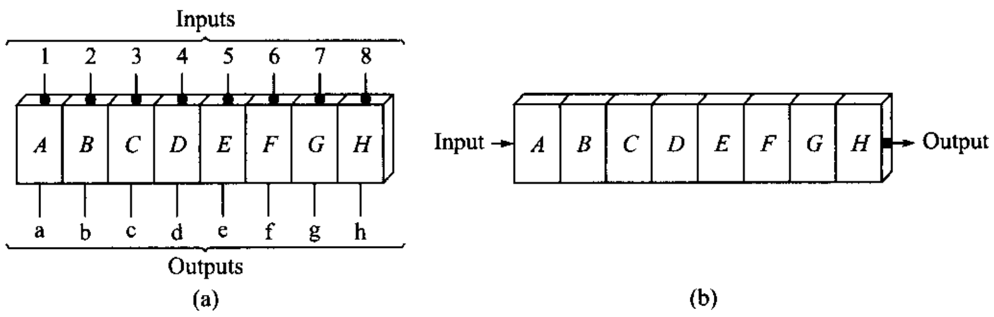


**Fig. 1.18** (a) An 8-bit parallel register, (b) An 8-bit serial register

*B*. When the third bit enters *A*, the bit in *A* goes to *B* and the bit in *B* goes to *C*. This shift right process is repeated, and after eight operations, the 8-bit number will be stored in the register. Since the bits are entered one after the other in a serial fashion, this is called a *serial register*. For a stored number to be extracted from this register, the bits must he shifted through the flip-flops from left to right. The stored number will then appear at the output, one bit at a time. It requires eight operations, or eight right shifts, to extract the stored number. A 74164 TTL is an example of an 8-bit serial register (this particular IC also provides parallel outputs).

## Transferring Digital Data

A register is used to enter data (binary numbers) into a microprocessor or computer. A register is also used to extract data from a computer and direct it to an external destination. Wire cables are generally the means for connecting systems. If a parallel register is used, the data is said to be shifted in parallel. The connector in this case must have one pin for each bit, and the cable must have at least one wire for each bit. An 8-bit register requires a cable having at least 8 wires, a 16-bit register must have at least 16 wires, and so on.

Data are also transferred (shifted) between registers within a digital system. Instead of drawing all 8 (or 16 or 32) wires on a schematic, it is common practice to use an arrow between the registers, as illustrated in Fig. 1.19a. The number 8 in parentheses means that there are eight wires. In this case, there are eight connections used to transfer 8 bits of data in parallel from register *A* to register *B*. The eight wires represented by this arrow are called a *data bus*. The double arrow shown in Fig. 1.19b means 16 bits of data can be shifted in parallel from *A* to *B* or from *B* to *A*. This is a 16-bit *bidirectional data bus*.

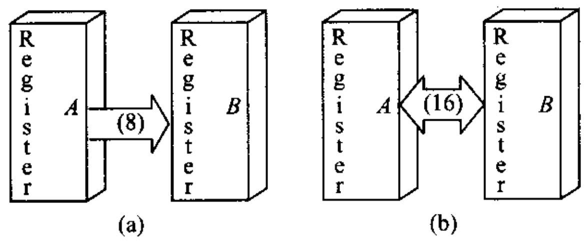


**Fig. 1.19** (a) An 8-bit data bus, (b) A 16-bit, bidirectional data bus

On the other hand, data can be shifted serially into or out of a serial register, and only one wire (connection) is required for the data. Clearly, parallel operation will transfer data into or out of a computer system much more rapidly than serial operation. The price paid for this gain in speed is an increase in complexity, in terms of both the electronic circuits and the increased number of connections (wires in the cable). The computer connector where data is entered or extracted is frequently called a *port*. Nearly all computer systems have available both a serial port and a parallel port.

## Magnetic and Optical Memory

Any memory element must be capable of storing or retaining only two logic levels, and there are numerous devices with the appropriate electronic circuits used for this purpose. One of the most common systems for memory makes use of the fact that a magnetic material can be magnetized with two different orientations. Thus, magnetizing spots on a strip of magnetic tape, or on a hard disk with a magnetic coating, or on a magnetic floppy disk are well known and widely used memory systems. In optical memory data is encoded in binary by making two different kinds of reflecting surface on a spiral track of a circular disk. A special pointed source of light falls on the surface and intensity of reflected light gives information about the data stored. A number of devices that utilize magnetic and optical storage are illustrated in Fig. 1.20.

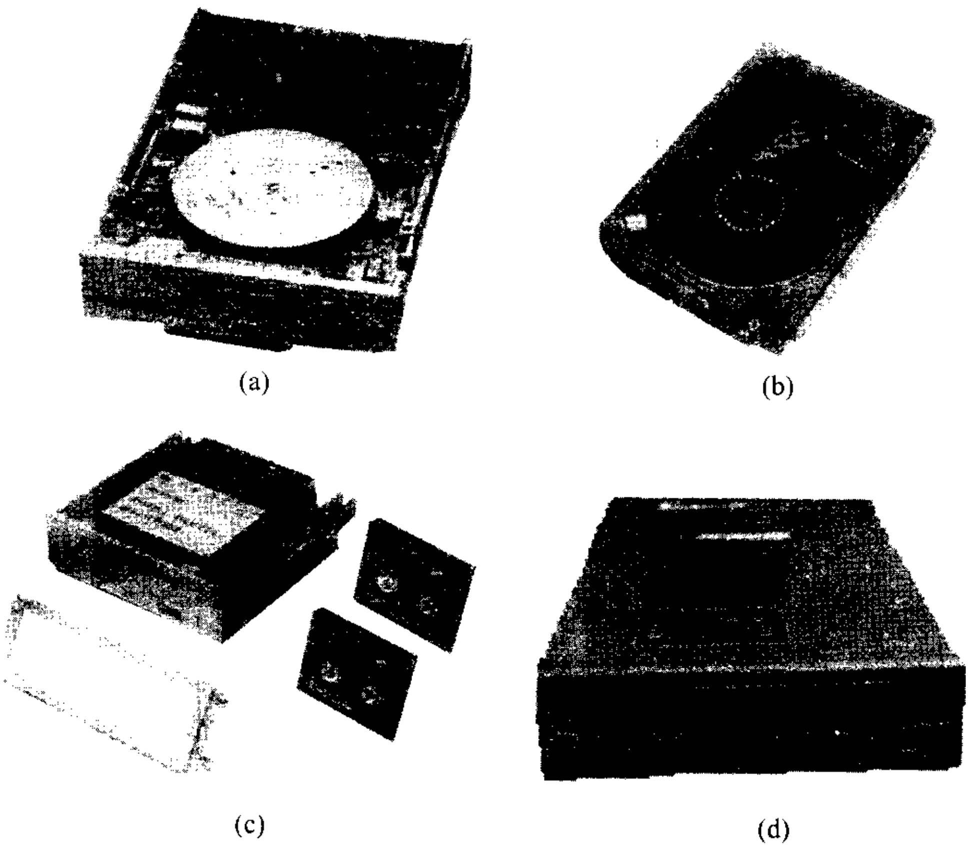


**Fig. 1.20** **(a) A 3½-inch magnetic floppy disk drive, (b) Magnetic hard disk drive, (c) Magnetic tape drive, (d) An optical disk drive**

SELF-TEST

14. Can you think of different ways to construct a digital memory element beginning with "opposite" terms such as on–off, in–out, up–down, right–left, cold–hot, wet–dry, etc.?
15. What logic level will appear at *A* if the flip-flop in Fig. 1.16 has SET = *L* and RESET = *H*?
16. Look at the binary representation of the decimal number 9 in Table 1.1. How many bits are there in this binary number? If it is stored in the register in Fig. 1.17, what are the bit values of *DCBA*?
17. If a shift operation requires a time of 1 $\mu$s to complete, how long would it take to enter an 8-bit number into the parallel register in Fig. 1.18a? How long would it take to enter an 8-bit number into the serial register in Fig. 1.18b?
18. When we speak of a microprocessor, what is meant by the term *port*?

## 1.5 DIGITAL OPERATIONS

### Counters

It was mentioned previously that counting is an operation easily performed by a digital circuit. A digital circuit designed to keep track of a number of events, or to count, is called a *counter.* The counter in Fig. 1.21a is constructed using a number of flip-flops (*n*) and additional electronic circuits. It is similar to a storage register, since it is capable of storing a binary number. The input to this counter is the rectangular waveform labeled *clock.* Each time the clock signal changes state from low to high, the counter will add one (1) to the number stored in its flip-flops. In other words, this counter will count the number of clock transitions from low to high. A clock having a small circle (bubble) in the input side would count clock transitions from high to low. This is the concept of *active low*—that is, an action occurs when the input is low.

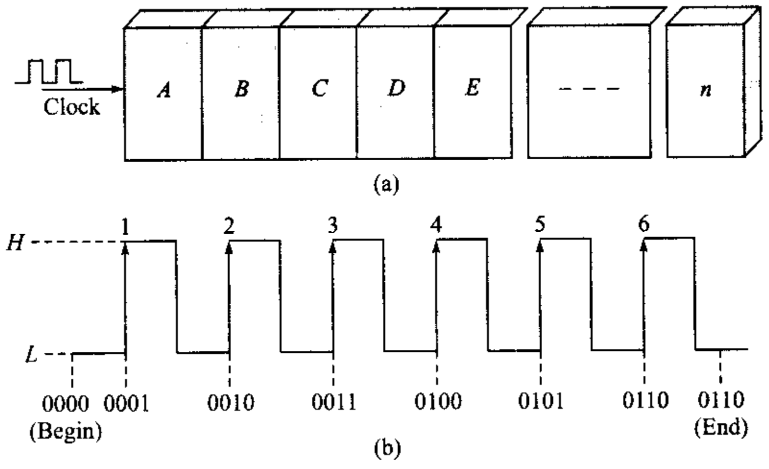


Fig. 1.21 (a) A counter constructed with *n* flip-flops, (b) A count of 6

As an example of how this circuit might be used, suppose that the counter consists of four flip-flops, all of which are RESET. That is, the binary number stored in the counter is 0000. The clock signal is initially held low. Now the clock is allowed to "run" for six clock periods, and then it is held low, as shown in Fig. 1.21b. After the first clock transition from low to high, the counter will advance to 0001. After the second transition, it will advance to 0010, and so on, until it will store the binary number 0110 after the sixth transition. The binary number 0110 is equal to decimal 6, and thus the counter has counted and stored the six clock transitions! The waveform in Fig. 1.21b shows the clock with the counter contents directly beneath each transition.

A four-flip-flop counter can count decimal numbers from 0 to 15. To count higher, it is necessary to add more flip-flops. It is easy to determine the maximum decimal count in terms of the number of flip-flops using the following relation

$$\text{Maximum count} = 2^n - 1 \tag{1.1}$$

where $n$ = number of flip-flops.

The term $2^n$ means 2 raised to the $n$th power, that is, 2 multiplied by itself $n$ times. For example,

$$\begin{aligned}
2^2 &= 2\times 2 = 4\\
2^3 &= 2\times 2\times 2 = 8\\
2^4 &= 2\times 2\times 2\times 2 = 16\\
2^5 &= 2\times 2\times 2\times 2\times 2 = 32\\
2^6 &= 2\times 2\times 2\times 2\times 2\times 2 = 64\\
2^7 &= 128\\
2^8 &= 256\\
2^9 &= 512\\
2^{10} &= 1024
\end{aligned}$$

We'll spend more time on binary and decimal numbers in Chapter 5. For now, this listing of powers of 2 can be used with Eq. (1.1). For example, the four-flip-flop counter has a maximum decimal count of

$$\text{Maximum count} = 2^4 - 1 = 16 - 1 = 15$$

**Example 1.2** How many flip-flops are required to count up to 100?

*Solution* From Eq. (1.1), we find that for $n = 6$, maximum count is $2^6 - 1 = 63$

and for $n = 7$, maximum count is $2^7 - 1 = 127$

Thus, 100 lying in between, the count would require 6 flip-flops.

The other way of solving this problem is to round the number $\log_2(N + 1)$ to next higher integer when a count of $N$ is desired. Since, $\log_2 101 = 6.6582$, the number of flip-flops required is 7.

## Arithmetic Logic Unit

An *arithmetic logic unit* (*ALU*) is a digital circuit capable of performing both *arithmetic* and *logic* operations. The basic arithmetic operations performed by an ALU are addition (+) and subtraction (–). Multiplication (×) and division (÷) of digital numbers are accomplished with other digital circuits. Logic operations will usually include inversion (NOT), AND, and OR. The ALU represented in Fig. 1.22 has two data inputs, the

*A* bus and the *B* bus, and the *F* bus is the resultant output. The digital levels on the *S* bus determine which operation is to be performed. Generally, each of the data buses will have the same number of bits. As an example, in the TTL family, the 74181 is an ALU similar to Fig. 1.22. Both the *A* and *B* inputs are 4-bit buses, and the output at *F* is also a 4-bit bus. For this particular circuit, the control signal at the *S* bus is also four bits. In addition, the 74181 has a number of other inputs and outputs which we will discuss in detail later.

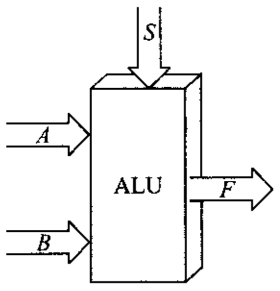


**Fig. 1.22** An arithmetic logic unit (ALU)

**Addition and Subtraction** If the proper digital levels are applied to the inputs of the *S* bus, the ALU in Fig. 1.23 can be used to add two digital numbers. The two numbers to be added are represented by the proper logic levels at *A* and *B*, and the SUM of these two numbers will appear at output *F*. In the event the sum of the two numbers generates a carry, an *H* will appear at the CARRY OUT. To illustrate, suppose we wish to perform the addition 6 + 7 = 13. Here's how we might do it with decimal numbers.

```
(carry)←
          6
          7(+1)
       ------
        1 3
```

The digital levels illustrated in Fig. 1.23 will result in the addition of these two numbers. The equivalent decimal numbers are shown in parentheses. Notice that the CARRY IN allows this ALU to add two numbers, plus a carry.

By changing the control levels at the *S* bus, this ALU will determine the difference *A* – *B* (subtraction). In this case, the digital levels at the *F* bus represent the DIFFERENCE, rather than the SUM.

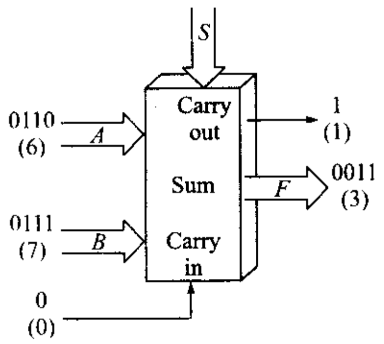


**Fig. 1.23** An ALU used for addition

**Logic Functions** By changing the digital levels at the *S* bus, the ALU in Fig. 1.22 can be used to perform a number of different logic functions relative to the two digital inputs. The desired function appears at the *F* bus. Here are some of the possibilities:

$$F = \overline{A}$$
$$F = \overline{B}$$
$$F = A \text{ AND } B$$
$$F = A \text{ OR } B$$

The operations are carried out "bit by bit." For example,

If $A = 1010$ then $F = \overline{A} = 0101$

If $A$ = 1010 and if $B$ = 0110, then

$$F = A \text{ AND } B = 1010 \text{ AND } 0110 = 0010$$

In this case, the ANDing is done on the corresponding bit of each input. There are tour AND operations. It's easier to see by writing the data as follows:

$$A = 1010$$
$$||||$$
$$B = 0110$$
$$F = 0010$$

The "vertical lines" between $A$ and $B$ show which bits are ANDed.

**Comparison** Comparing the magnitude of two numbers is an important logical operation. The circuit in Fig. 1.24 is a *comparator*. It is capable of comparing two digital numbers and indicating whether the magnitude of one is greater than, less than, or equal to the other. For example, if $A$ = 0110 (decimal 6) and $B$ = 0111 (decimal 7), then the output $A < B$ will be high. The other two outputs will be low. A 7485 in the TTL family is a 4-bit comparator similar to Fig. 1.24. Also, the 74181 ALU can be used with the same results.

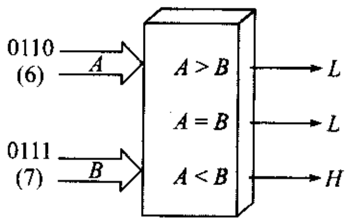


Fig. 1.24 A comparator

## Input/Output

In order for any digital system to be useful, there must be some provision for entering data into the system and also some method of extracting data from the system. In the case of a computer, information is frequently entered by typing on a keyboard or perhaps by using a magnetic floppy disk. Useful information can be obtained from the computer by examining the visual displays on a cathode-ray tube (CRT) or by reading material produced on a printer. Clearly there is a requirement to connect multiple input devices, one at a time, to the system. The digital circuit used for this operation is a multiplexer. Likewise, there is a need to connect the system output to a number of different destinations, one at a time. The digital circuit used for this purpose is a demultiplexer.

The term *multiplex* means "many into one." A *multiplexer* (*MUX*) can be represented as shown in Fig. 1.25a. There are $n$ input lines. Each line is used to shift digital data serially. There is a single output line which is connected to the computer (system) input port. Operation of the circuit can be explained by using the "switch" as a model. Each setting of the digital control levels on the $C$ bus will connect the switch to one of the input lines. Data from that particular input is then entered into the computer. Changing the $C$ bus levels will connect a different input. Thus, data from multiple sources can be connected to a single input port, one at a time. An example of a MUX is the 74150 in the TTL family. It has 16 input lines and a single output line.

The opposite of multiplex is *demultiplex*, which means "one into many." A *demultiplexer* (*DEMUX*) can be represented as in Fig. 1.25b. This digital circuit simply connects the single data input line to one of the $n$ output lines, one at a time, according to the levels on the $C$ bus. Thus serial data from the computer output port can be directed to different destinations, one at a time. An example of a DEMUX is the TTL 74154, which can be used to connect a single input to any one of 16 outputs, one at a time.

Any information entered into a digital system must be in the form of a digital number. A circuit that changes data into the required digital form is called an *encoder*. The encoder shown in Fig. 1.26a on the next

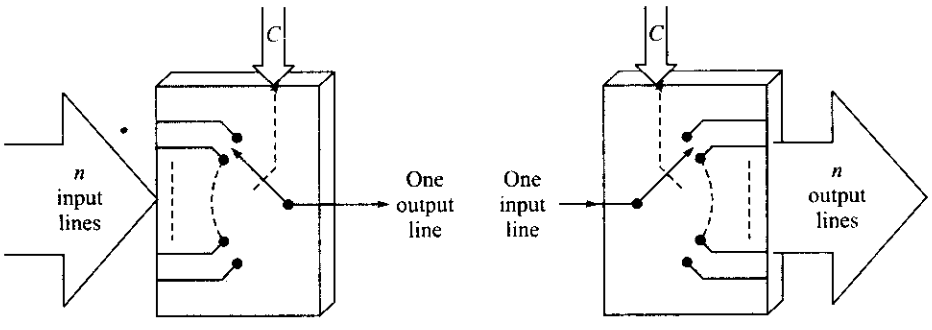


Fig. 1.25 (a) A multiplexer (MUX), (b) A demultiplexer (DEMUX)

page will change a decimal number into its binary equivalent. It may be used with a keyboard. For instance, depressing the number 4 key on a keyboard will cause input line 4 to this encoder to be high (the other inputs are all low). The result will be decimal 4, binary 0100, at the encoder output as shown.

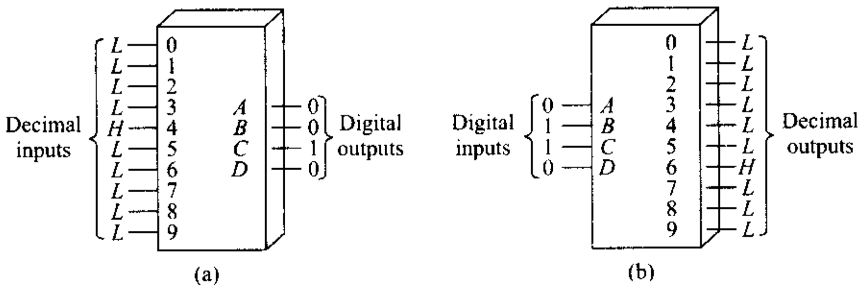


Fig. 1.26 (a) An encoder, (b) A decoder

Taking digital information from the output of a computer and changing it into another form is accomplished with a *decoder*, for example, changing the digital number 0110 (decimal 6) into its decimal form. The decoder in Fig. 1.26b will accept a 4-bit binary number and indicate its decimal equivalent between 0 (zero) and 9. As shown, the binary input 0110 will cause output line 6 to be high, while all other output lines remain low. There are many different types of encoders and decoders. A number of them will be discussed in detail in Chapter 4.

## SELF-TEST

19. What binary number will be stored in the counter in Fig. 1.21a if the clock is allowed to run for seven periods?
20. How many flip-flops are required to construct a digital counter capable of counting 1000 events?
21. State whether or not the ALU in Fig. 1.22 will generate a carry out if the numbers added are: (a) 2 and 3; (b) 5 and 5; (c) 9 and 9.
22. What are the digital output levels of the encoder in Fig. 1.26a if only input line 6 is high?

## 1.6 DIGITAL COMPUTERS

### Terms

An appropriate selection of the previously discussed digital circuits can be interconnected to construct a digital computer. A computer intended to perform a very specific task, constructed with a minimum number of components, might be referred to as a *microcomputer*. Small portable, or desktop, computers are usually in the microcomputer class. Computers with greater capacities, often used in business, are called *minicomputers*. A large mainframe computer system capable of storing and manipulating massive quantities of data, for example, a digital computer system used by a bank or an insurance company, might then be called a *maxicomputer*.

### Uses

What can a digital computer be used for? Numerical computation is surely one possible use. The inclusion of an ALU with additional logic circuits provides arithmetic capabilities (addition, subtraction, multiplication, division). The logic portion of the ALU means the computer can be used to make logical decisions. Beyond these basic functions, a digital computer can be used to process data (balance bank accounts), to rapidly perform otherwise time-consuming tasks (determine payroll amounts and print out paychecks), to precisely monitor and control intricate processes (life support systems in a hospital operating room), to use speech for communication with humans (automatic telephone systems and voice recognition)—the list is almost endless, and is limited only by the ingenuity and resourcefulness of individual users!

### Basic Configurations

A microcomputer designed to control a given machine, process, or system might be represented as in Fig. 1.27. The control signals produced by the computer appear as the output bus and are sent to an output device. Here, the signals are properly conditioned and sent to the mechanism being controlled. The controlled entity must then send signals indicating its present condition back to the computer via an input device and via the input bus. The computer analyzes these present condition signals, determines any necessary action, and sends required correction signals out to the system.

A microcomputer system might be designed to irrigate the lawn area of a park. Watering is to be done only at night, when the soil moisture falls below a given value. The system "sensors" in this case would be (1) a probe to detect soil moisture and (2) a light detector to distinguish between daylight and darkness. The computer would monitor signals from the two sensors, turn on the watering system whenever the soil moisture fell below a set value (but only at night), and turn the system off as the moisture increased above the desired value.

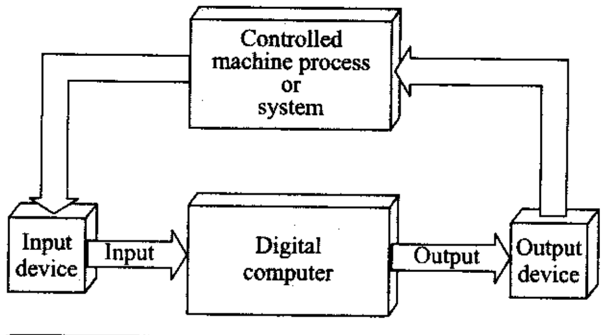


Fig. 1.27 A digital computer based system

The minicomputer illustrated in Fig. 1.28 on the next page is more complicated than the microcomputer described, but it has greatly increased utility. The input bus is serviced with a MUX.

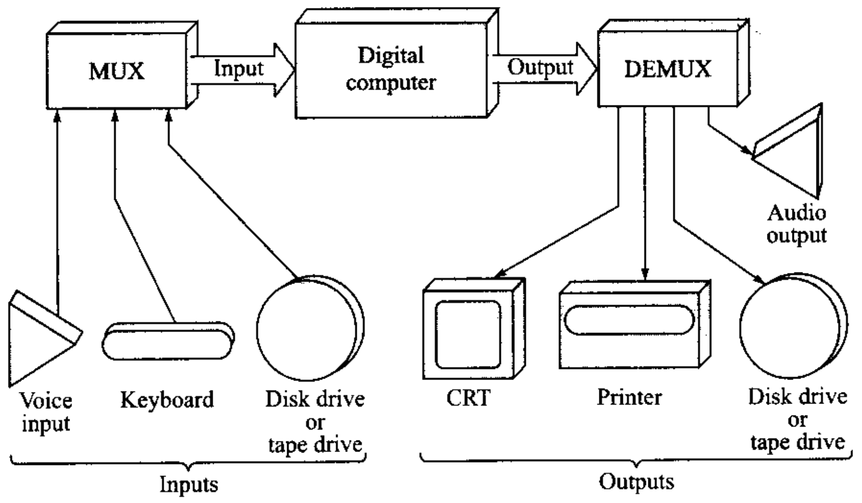


Fig. 1.28 **Block diagram of a computer system**

This allows the connection of a number of different input devices:

A keyboard for typewritten entry of alphanumeric information

A disk drive or tape drive for entering data stored in magnetic form

A microphone for voice input

The DEMUX on the output bus allows numerous possibilities for receiving information from the computer:

The familiar CRT for a visual display

A printer to provide printed material (called *hard copy*)

A disk or tape drive to record data in magnetic form

Perhaps a speaker for audio information

A minicomputer such as this can be used for many different tasks. It can be used as a word processor, for data processing, for communication via telephone (both voice and fax), for training in an educational setting, for computer games, and so on!

The block diagram in Fig. 1.28 forms the basis for larger computer systems. A larger system will have a much greater capacity to store data and will in general utilize numerous input/output units. For instance, a maxicomputer will likely have more than one printer, and perhaps even different types of printers. It will generally have a large number of users, all of whom desire access to the system at the same time. One workstation must then be provided for each user. A keyboard and a CRT are the minimum components required at each workstation. The digital circuits used to construct maxicomputer systems are necessarily more complicated than minicomputer systems, and they may operate at a much faster rate. Let's take a look inside a typical digital computer.

## Basic Computer Architecture

The *central processing unit* (*CPU*) is the brain of a digital computer. It is constructed using an ALU along with a number of registers and counters. The CPU is therefore the primary center for computation and decision making. All the operations within the CPU, and indeed within the computer itself, must be carefully coordinated. A digital signal referred to as the *system clock* is used as a reference to time when specific operations take place. The clock signal is usually a periodic, rectangular waveform as illustrated in Fig. 1.6. Using a crystal in the clock circuit allows the accuracy and stability of the clock frequency, to be controlled with great precision. The clock provides a "heartbeat" for the computer. A block diagram of a digital computer is started by drawing the CPU and clock as shown in Fig. 1.29.

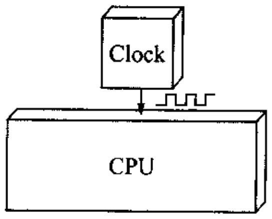


Fig. 1.29 The "heart" and "brain" of a digital computer

The CPU is capable of computation and decision, but it must have specific instructions telling it exactly *what* to do and *when* to do it. This set of instructions is called a *program*. A program is a detailed list of operations written by a human programmer. The programmer decides what the computer is to do and when it should be done, and then writes a list of instructions to be carried out in the proper order. The program is entered into the computer, using perhaps a keyboard, and stored in the computer *memory*. The CPU can then "fetch" from memory one instruction at a time, in the given order. It will execute the instruction and then fetch the next instruction. With this repeated fetch-and-execute cycle, the CPU will accomplish the desired task. A memory block used for program storage has been added in Fig. 1.30.

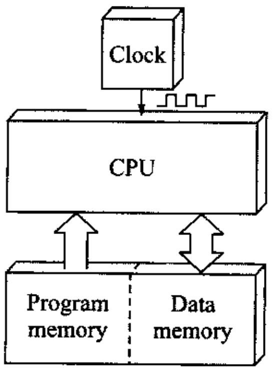


Fig. 1.30 CPU with memory block

A portion of the memory block in Fig. 1.30 is labeled *data*. This is the area where the information being processed by the computer is stored. In addition, this is where the CPU stores the results of computations and/or decisions made. Since the CPU takes ("reads") data from memory, as well as returns ("writes") data into memory, the memory data bus is bidirectional. By contrast, the program data bus is not bidirectional, since information on this bus is always from memory to CPU.

The CPU communicates with the "outside world" by means of the input encoders and the output decoders. The ability to multiplex inputs and demultiplex outputs may also be included in the input/output blocks that have been added in Fig. 1.31a. This configuration is sometimes quite inefficient, since all information entering or exiting the computer must pass through the CPU. The CPU operates at a much faster rate than most external devices, and it must *wait* while data are being entered or exited. A *direct memory access (DMA)* block is generally included to alleviate this problem. As seen in Fig. 1.31b, the DMA allows information to move directly from an input device into memory, or from memory directly to an output device. While information is being transferred via the DMA, the CPU is free to carry on its computational or logical operations. This greatly improves system efficiency as well as speed of operation.

Before data can be entered into the computer, a signal on the input request line asks the computer for "permission" to input information. For instance, depressing the enter key on a keyboard will generate an input request signal. When the CPU is ready, a signal is generated on the acknowledge line, and data will be entered

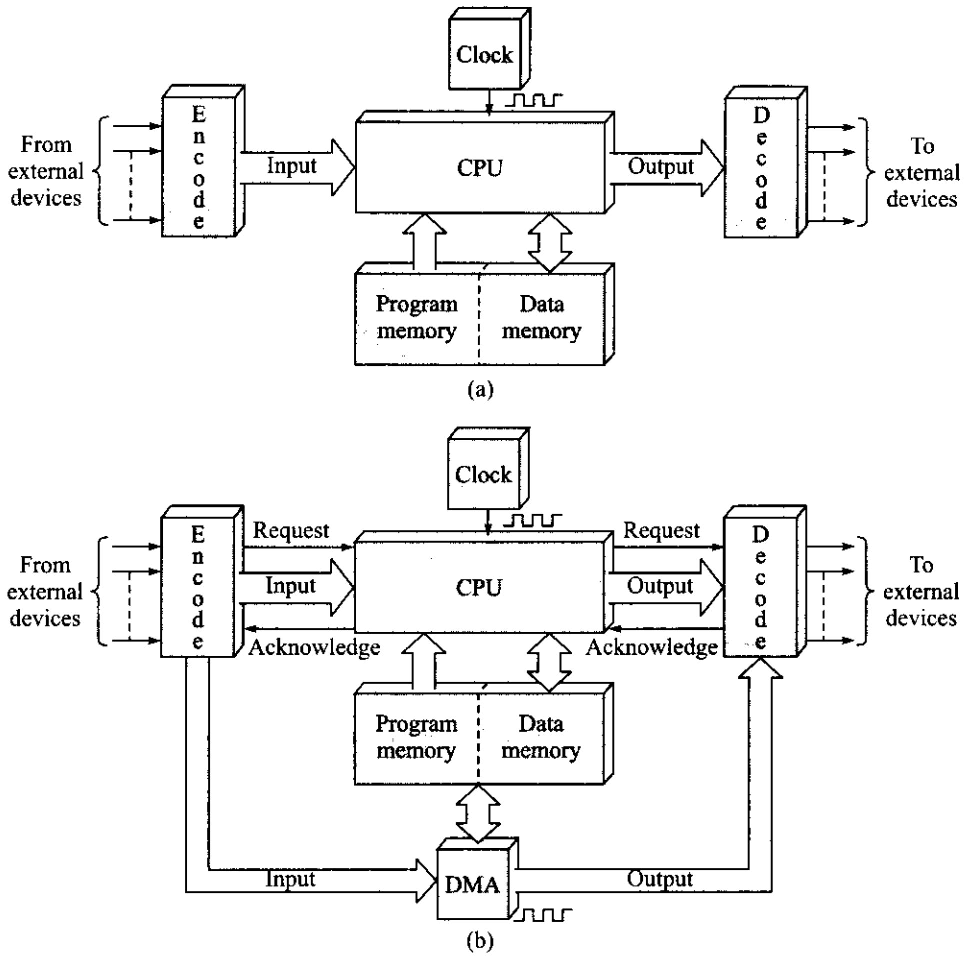


Fig. 1.31 CPU, memory with input/output block

via the DMA into the memory. This request-acknowledge sequence is often called *handshaking*. A similar handshaking must occur when the CPU is ready to deliver data to an external device. However, in this case, the CPU makes an output request, and the external device gives permission.

All of these blocks are operated in synchronism with the clock, but additional direction must be provided. The *controller* is the unit that decides which block "goes first" (establishes priorities), decides the order in which external devices are serviced, routes data along the various buses such that no conflicts occur, and controls the overall operation of the system. As such, the controller communicates individually with each block as illustrated in Fig. 1.32. This block diagram is representative of the architecture of many digital computers.

A *microprocessor* is often used as the basic IC around which a microcomputer or minicomputer is constructed. Numerous computers have been designed, beginning with the 8080 microprocessor. Improvements to this basic IC have led to the development of a family of microprocessor units including the 8085 and the 8086.

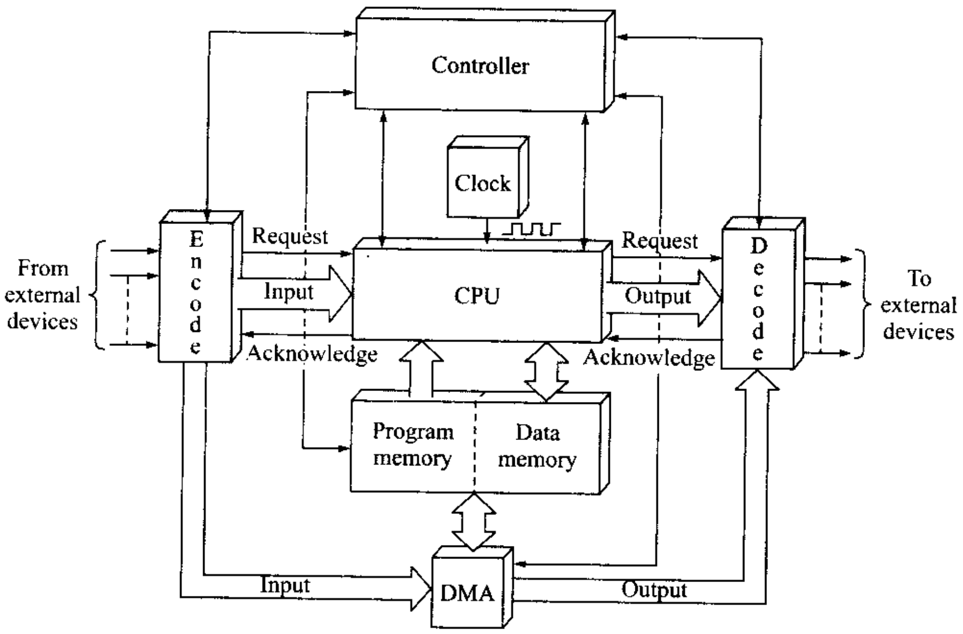


Fig. 1.32 A microprocessor-based digital computer

Referring to the block diagram in Fig. 1.32, a modern fully integrated microprocessor contains the controller, the clock, the CPU, and portions of the memory, the DMA, and the input and output blocks. The purpose of this section is to provide a general understanding of a digital computer. In the remainder of this text, the blocks used to build any digital system, including a digital computer, will be studied in detail.

**SELF-TEST**

23. Why is the system clock considered the heart of a digital computer system?
24. What is a computer program?
25. What functions are carried out in an ALU?
26. What is the purpose of the DMA in Fig. 1.32?

## 1.7 DIGITAL INTEGRATED CIRCUITS

A digital IC is constructed by an interconnection of resistors, transistors, and perhaps small capacitors, all of which have been formed on the surface of a semiconductor wafer. The entire circuit resides on a tiny piece of semiconductor material called a *chip*.

The semiconductor wafer is typically a slice of monocrystalline silicon about 0.2 mm thick and perhaps 8 to 15 cm in diameter, as illustrated in Fig. 1.33a. The wafer is divided checkerboard fashion into 1000 or so rectangular areas. Each area will become a single chip. The resistors and transistors necessary for each digital

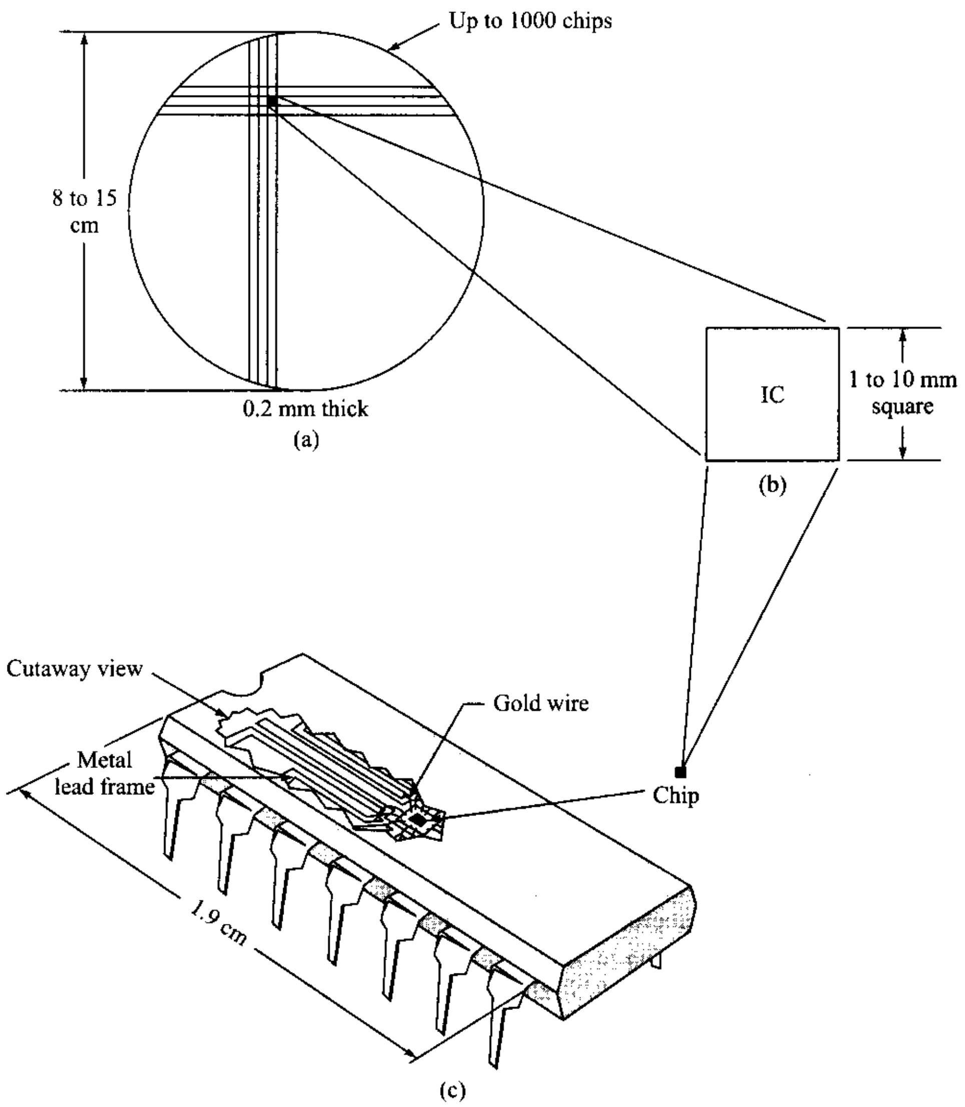


**Fig. 1.33** (a) A wafer, (b) One chip, (c) Dual-inline package (DIP)

circuit are then formed on each chip by a series of semiconductor processing steps. In this fashion, identical digital circuits are manufactured simultaneously on the same silicon wafer.

After the processing steps are completed, the wafer is separated into individual chips as seen in Fig. 1.33b. Each chip is a digital circuit, for example, an inverter or an AND gate. An individual digital circuit may have only a few components, but some circuits have a few hundred components! Each chip is then mounted in a suitable package, as shown in Fig. 1.33c. The package illustrated here is a 14-pin *dual-inline package* (*DIP*). Some additional packages for ICs are shown in Fig. 1.34.

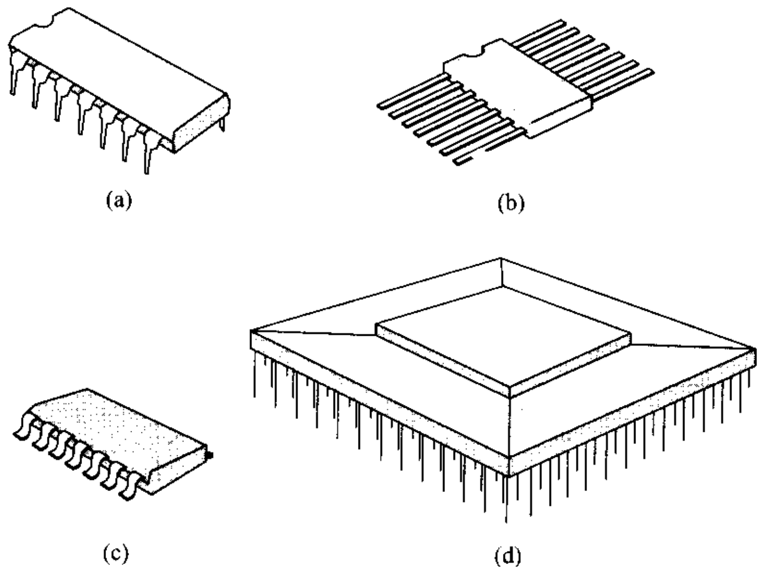


**Fig. 1.34** **Some IC packages: (a) DIP, (b) Flat pack, (c) Surface mount, (d) Pin-grid array**

## IC Families

ICs are categorized by size according to the number of gates contained on each chip. There is no absolute rule, but an IC having fewer than 10 or 12 gates is usually referred to as a *small-scale integration (SSI) IC*. For instance, a 7404 has six inverters in a 14-pin DIP. ICs having more than 12 but fewer than 100 gates are called *medium-scale integration (MSI) ICs*; encoders and decoders are examples of MSI ICs. If there are more than 100 gates but fewer than 1000, the IC is called a *large-scale integration (LSI) IC*. An IC having more than 1000 gates is referred to as a *very large-scale integration (VLSI) IC*. A large complex system such as semiconductor memory or a microprocessor will be either LSI or VLSI.

ICs are further categorized according to the type of transistors used. The two basic transistor types are *bipolar* and *metal-oxide-semiconductor (MOS)*. Bipolar technology is faster but requires more power, and is generally preferred for SSI and MSI. MOS is slower, but requires much less power and also occupies a much smaller chip area for a given function. MOS is therefore preferred for LSI and is widely used in applications such as pocket calculators, wristwatches, hearing aids, and so on. For the moment, let's consider the overall characteristics of each digital IC family.

## Bipolar Transistors

There are two important digital circuit families constructed using bipolar transistors:

- **Transistor-Transistor Logic** (TTL)
- **Emitter-Coupled Logic** (ECL)

**Transistor-Transistor Logic** TTL was first introduced by Texas Instruments in 1964 using the numbers 54XX and 74XX. These two families are now widely available from a number of different manufacturers. The 74XX ICs operate over a temperature range of 0°C to 75°C. The 54XX devices are more rugged; they operate over a temperature range of −55°C to +125°C. As you might expect, the 54XX devices are more expensive. Otherwise, the logical operations of these two families are the same. In each case, the XX portion of the part number refers to a specific device. For instance, "04" stands for inverter, and a 7404 is a TTL inverter. A 7411 is a TTL AND gate, and so on. When there is no danger of confusion, it is common practice to shorten the description by omitting the first two digits. Thus, a 7404 inverter is referred to as a 04, and the 7411 AND gate is designated as 11. Table 1.2 is a partial listing of some widely used 74XX ICs. Note that a 5404 is logically the same as a 7404; it is simply guaranteed to operate over a wider temperature range.

**Table 1.2 Some Standard Digital Circuits**

| *TTL* | *ECL* | *CMOS* | *Function* |
|---|---|---|---|
| 7400 | — | 74HC00 | Quad 2-input NAND gate |
| 7402 | MC10102 | 74HC02 | Quad 2-input NOR gate |
| 7404 | | 74HC04 | Hex inverter |
| 7408 | MC10104 | 74HC08 | Quad 2-input AND gate |
| 7432 | MC10103 | 74HC32 | Quad 2-input OR gate |

In the interest of higher operating speed, the 74XX family was improved with the introduction of the 74HXX (where the H stands for high speed) family of devices. The price paid for increased speed was an increase in power required to operate each gate. This led to another family of devices designed to minimize power requirements—the 74LXX (where the L stands for low power) series.

A major improvement in the TTL series came with the development of a special transistor arrangement called a *Schottky transistor*. Using this device, the 74SXX (S for Schottky) family came into being. These devices greatly improved operating speed, but again at the cost of increased power consumption. At this point, a family of devices designated 74LSXX (low-power Schottky) was developed. The 74LSXX family offers high-speed operation with minimal power consumption and today is preferred in most designs. The original 7400 also remains popular.

There are two additional families, 74ASXX (advanced Schottky) and 74ALSXX (advanced low-power Schottky), available. One might anticipate the development of other families with characteristics to match specific needs. Table 1.3 is a comparison of power consumption and speed of operation (delay time) for some TTL families.

**Emitter-Coupled Logic** Emitter-coupled logic (ECL) is considerably faster than any of the TTL families, but the power required for each gate is also much higher. With a propagation delay of only 2 ns, the industry standard for ECL circuits is 10,000 ECL, abbreviated 10K. The 100K (100,000) series is even faster, with a delay time of only 1 ns. Motorola markets a family of devices designated MECL 10K and MECL 10KH (Motorola Emitter Coupled Logic). Representative 10K MECL circuits are listed in Table 1.2.

## MOS Transistors

There are three digital logic families constructed using MOS field-effect transistors (MOSFETs):

- **PMOS** Using *p*-channel MOSFETs
- **NMOS** Using *n*-channel MOSFETs
- **CMOS** Using both *n*-channel and *p*-channel MOSFETs

PMOS, the slowest and oldest type, is nearly obsolete today. NMOS dominates the LSI field and is widely used in semiconductor memories and microprocessors. CMOS is preferred where individual logic circuits are used and where very low power consumption is required.

Table 1.3 **TTL Power-Delay Values**

| *Type* | *Name* | *Power, mW* | *Delay Time, ns* |
|---|---|---|---|
| 74XX | Standard TTL | 10 | 10 |
| 74HXX | High-speed TTL | 22 | 6 |
| 74LXX | Low-power TTL | 1 | 35 |
| 74SXX | Schottky TTL | 20 | 3 |
| 74LSXX | Low-power Schottky | 2 | 10 |

The original 4000 series of MOS devices was introduced by RCA. It was slow, was not compatible with TTL, and is rarely seen in modern designs. The 74CXX (C for CMOS) is a series of digital ICs that are manufactured using MOS technology. These devices are pin-for-pin replacements for similarly numbered 7400 TTL devices. For instance, a 7404 is an IC that contains six inverters, and the 74C04 also contains six inverters. Thus, digital circuits designed using 74XX TTL devices can also be implemented using 74CXX MOS devices. The 74CXX design will require considerably less operating power but will be restricted to lower operating speeds. The 74HCXX (where the HC stands for high-speed CMOS) family of circuits is the most widely used today (see Table 1.2).

Complete digital logic systems can be constructed using entirely 74XX devices or 74HCXX devices, and the two types of devices can even be used together, provided certain precautions are observed. The necessary precautions involve voltage levels, current requirements, and switching times; this comes under the subject of *interfacing*, which will be addressed in Chapter 13. A second family of CMOS devices that is entirely compatible with TTL circuits is the 74HCTXX series (where the H stands for high-speed, the C for CMOS, and the T for TTL-compatible).

Since the mid-1980s, two additional CMOS families have been available—the 74ACXX (advanced CMOS), and the 74ACTXX (advanced CMOS TTL-compatible) families. As with TTL, there will no doubt be further advancements to produce additional CMOS families.

## Digital Logic Symbols

The Institute of Electrical and Electronics Engineers (IEEE) along with the American National Standards Institute (ANSI) have developed a new symbolic language and set of symbols to be used with digital logic circuits. These new symbols are now being used on manufacturers' data sheets along with traditional symbols. The most recent revision of *IEEE Standard Graphic Symbols for Logic Functions*, ANSI/IEEE Std 91–1984, provides for two different types of symbols. Symbols of the first type, called *distinctive-shape symbols*, are exactly as have been shown throughout this chapter. The second system, which is called the *rectangular-shape system*, uses a rectangular box with a special symbol for each type of gate. The IEEE standard does not express a preference for either shape. Most people presently involved in digital electronics seem to prefer the distinctive-shape system, and since this type of symbol is still included on data sheets, we will use these symbols in this text. Below is a brief introduction to the new rectangular symbols, which are presented along with their traditional, distinctive-shape counterparts. More detailed information is available in Appendix 7.

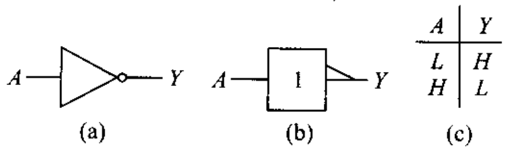


Fig. 1.35 **(a) Standard symbol, (b) New IEEE symbol, (c) Truth table**

The standard logic symbol for an inverter is shown in Fig. 1.35a, where *Y* is the complement of

*A*. The new IEEE symbol is shown in part *b* of this figure. A rectangular box is used for the gate, the input is labeled *A*, and the output is labeled *Y*. The 1 inside the box signifies that the input must be active in order to have an active output. The triangle on the output line signifies that the output is active when low. Thus, when the input is active (high), the output will be active (low). The truth table is shown in Fig. 1.35c.

The 7404 is a hex inverter; that is, it is an IC that contains six inverters. The DIP package for this device is shown in Fig. 1.36a, along with the proper pin numbers on the package. Figure 1.36b shows the six standard logic symbols for the inverters. Figure 1.36c shows the new IEEE logic symbol.

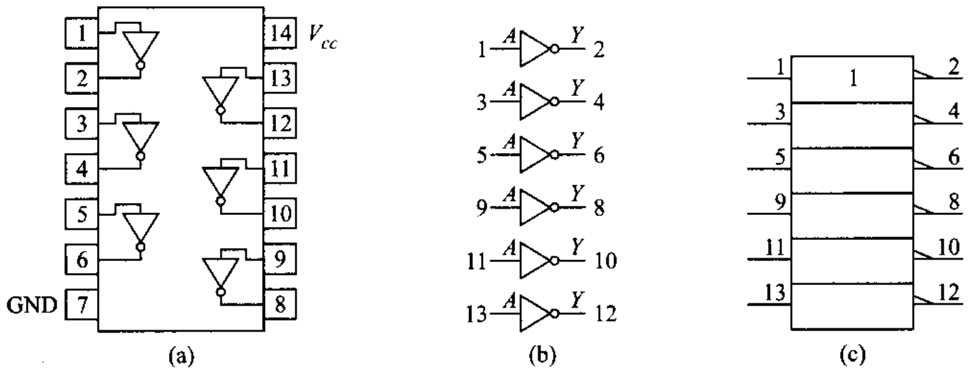


**Fig. 1.36** **Hex inverter, 7404: (a) Pin configuration, (b) Logic symbol, (c) Logic symbol (IEEE)**

A 7411 is an IC that contains three 3-input AND gates. The DIP package and pinout for the 7411 are shown in Fig. 1.37a, and the standard logic symbols are given in Fig. 1.37b. The new IEEE symbol for the AND gate is a rectangle with the ampersand (&) symbol written in it; is used in Fig. 1.37c to show the three AND gates in the 7411.

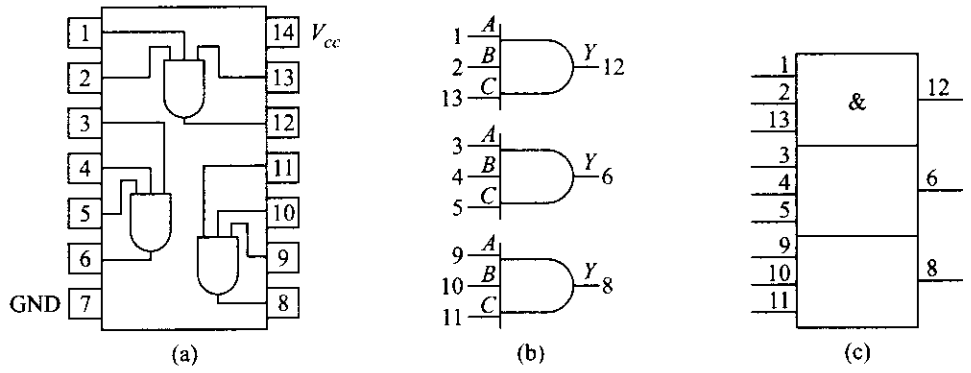


**Fig. 1.37** **Triple three-input AND gate, 7411: (a) Pin configuration, (b) Logic symbol, (c) Logic symbol (IEEE)**

The pinout and symbols for the 7432 quad 2-input OR gate are shown in Fig. 1.38. The term *quad* means “four,” and there are four gates in this DIP. The IEEE symbol for an OR gate is a rectangular box with the greater than or equal to (≥) symbol inside. This symbol means “at least one input must be high in order for the output to be high.”

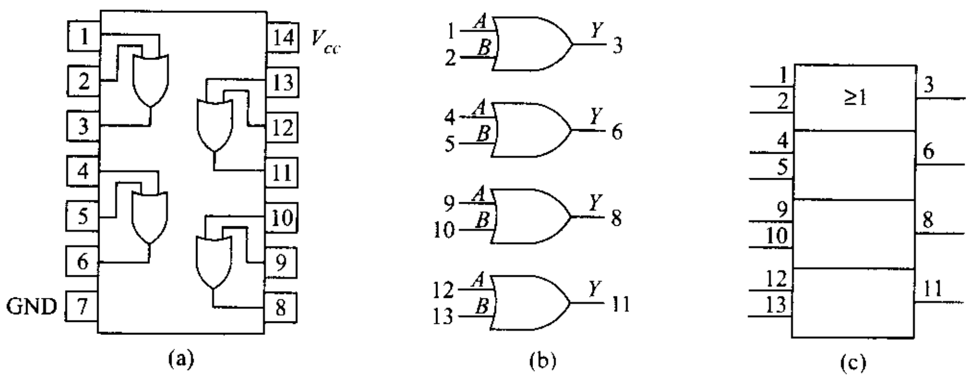


Fig. 1.38 **Quad two-input OR gate, 7432: (a) Pin configuration, (b) Logic symbol, (c) Logic symbol (IEEE)**

**SELF-TEST**

27. What is generally accepted as the number of gates per chip for SSI, MSI, and LSI?
28. Which is faster, TTL or ECL? Which requires more power to operate?
29. Over what temperature range will 74XX TTL operate?
30. When referring to TTL ICs, what is the meaning of quad? of hex?
31. In the 74ACTXX family of ICs, what do the letters A, C, and T stand for?
32. What is the significance of the triangle on the output line of the inverter in Fig. 1.35b?

## 1.8 DIGITAL IC SIGNAL LEVELS

The voltages in Fig. 1.39 are used to define the two digital logic levels, $H = 1 = T$ and $L = 0 = F$. Logic level 1 is any voltage between $+V_{CC}$ and $+V_{H,\min}$. Logic level 0 is any voltage between $+V_{L.\max}$ and 0. Voltages within the *forbidden* region are not allowed. This illustration is often called a *profile*, and it can be used to define the operation of any digital logic circuit. Each family of digital circuits has its own unique operating characteristics, and each individual circuit has an input and an output. Thus, there must be an input profile and an output profile for each family. An understanding of the correct voltage levels for each digital family is absolutely essential. Measuring logic levels in the laboratory, interconnecting different logic families, and connecting digital logic circuits with other digital circuits require a detailed knowledge of voltage levels. For now, we will consider the two most popular TTL families and one of the widely used CMOS families. Profiles for other circuits are easily obtained from manufacturers' data books.

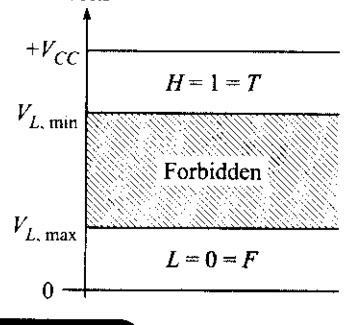


Fig. 1.39 **Logic level profile**

## TTL Logic Levels

The 74XX and 74LSXX are the two most widely used TTL families, and they have identical voltage-level characteristics (there is a difference in current capabilities, however).

The input profile and the output profile for the 74XX and the 74LSXX family are shown in Fig. 1.40. From the output profile, each circuit will produce a voltage between $+V_{CC}$ = +5 Vdc and $V_{OH,min}$ = 2.4 Vdc to signify $H = 1$. A voltage level between $+V_{OL,max}$ = 0.4 Vdc and 0 Vdc will be produced to signify $L = 0$.

From the input profile, it is seen that any voltage between $+V_{CC}$ = +5 Vdc and $+V_{IH,min}$ = 2.0 Vdc is recognized as $H = 1$. Any voltage between $+V_{IL,max}$ = 0.8 Vdc and 0 Vdc is recognized as $L = 0$.

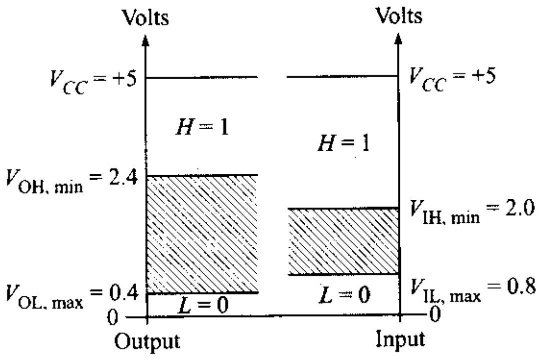


Fig. 1.40 74XX and 74LSXX profiles

Look carefully at Fig. 1.40 and note that any output voltage within the high range is *within* the input range recognized as a high. Similarly, any output voltage within the low range is *within* the input range recognized as a low. Clearly an output voltage from any 74XX circuit can be used as the input signal to any other 74XX circuit! This family of circuits is thus said to be *compatible*. This shouldn't come as a surprise, since any circuit within a family should be able to "drive" any other circuit within the same family. Similarly, any 74LSXX circuit can drive any other 74LSXX circuit—this is also a compatible family. Furthermore, the 74XX and 74LSXX families are compatible with one another.

There are, however, differences in the *number* of circuits that can be connected to the output in each family. This consideration, called *fanout*, is discussed in Chapter 13.

## CMOS Logic Levels

74HCXX is the most widely used CMOS family. The input profile and the output profile for the 74HCXX family are shown in Fig. 1.41. From the output profile, each circuit will produce a voltage between $+V_{CC}$ = +5 Vdc and $+V_{OH,min}$ = 4.9 Vdc to signify $H = 1$. A voltage level between $+V_{OL,max}$ = 0.1 Vdc and 0 Vdc will be produced to signify $L = 0$.

From the input profile, it is seen that any voltage between $+V_{CC}$ = +5 Vdc and $V_{IH,min}$ = 3.5 Vdc is recognized as $H = 1$. Any voltage between $+V_{IL,max}$ = 1.5 Vdc and 0 Vdc is recognized as $L = 0$.

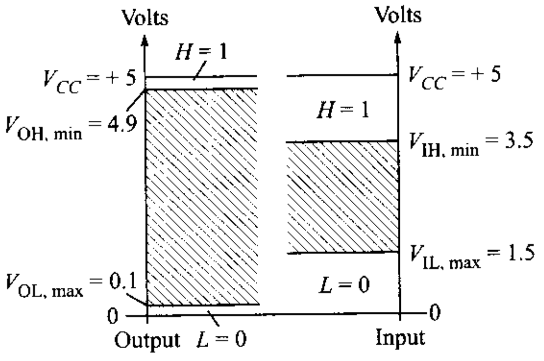


Fig. 1.41 74HCXX profiles

Look carefully at Fig. 1.41 and note that any output voltage within the high range is *within* the input range recognized as a high. Similarly, any output voltage within the low range is *within* the input range recognized as a low. Clearly an output voltage from any 74HCXX circuit can be used as the input signal to any other 74HCXX circuit! This family of circuits is thus said to be *compatible*.

By comparing the profiles in Figs. 1.40 and 1.41, you can see that a 74HCXX CMOS circuit can be used to drive any 74XX TTL circuit. However, a 74XX TTL *cannot* be used to drive a 74HCXX CMOS. The voltage levels *are not* compatible. Interconnecting different families like this is called *interfacing*. Both interfacing and fanout are considered in Chapter 13.

## Noise Margin

We shall end the discussion on digital IC signal levels by introducing the concept of *noise margin*. We have noted that level $H = 1$ and $L = 0$ are represented by a range of voltages. Consider, output of a digital device is connected to input of another digital device (see Fig. 1.42) but some noise (in the form of a random voltage) can get added to the output voltage before it arrives at the input of next device.

Now, refer to Fig. 1.40 for TTL digital devices. The output for $H = 1$ can be lower than 5 V and can go as low as $V_{\text{OH,min}}$(2.4 V). The input of a TTL device is treated as $H = 1$ if it's within the range +5 V to $V_{\text{IH,min}}$(2.0 V). An addition of any random noise voltage greater than 2.0 V–2.4 V = –0.4 V (e.g., –0.3 V, –0.2 V) makes the summer output greater than 2.0 V, the minimum acceptable level $V_{\text{IH,min}}$ at input side. Thus, there exists a noise margin, the amount of noise that can get added without any possibility of logic value misinterpretation. This can be defined for $H = 1$ as

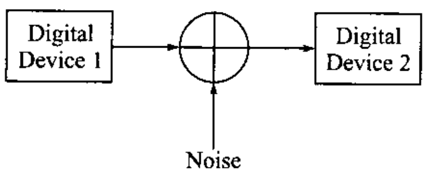


**Fig. 1.42 Noise affecting output of a digital device**

$$NM_{\text{H}} = V_{\text{IH,min}} - V_{\text{OH,min}} \tag{1.2}$$

Similarly, a noise margin for $L = 0$ exists, beyond which output of a device at $L = 0$ may fail to get identified as the same, at the input of a similar device due to noise corruption. The worst-case scenario here can make $V_{\text{OL,max}}$ plus noise go above $V_{\text{IL,max}}$. Thus noise margin for $L = 0$ can be defined as

$$NM_{\text{L}} = V_{\text{IL,max}} - V_{\text{OL,max}} \tag{1.3}$$

Note the polarity of the noise voltage in above two cases. For $NM_{\text{H}}$ the corrupting noise voltage should be negative and less than the noise margin to cause any misinterpretation. If it is positive, there is no such issue. However, for $NM_{\text{L}}$ the corrupting noise voltage should be positive when misinterpretation may occur.

For TTL 74XX and 74LSXX family,

$$NM_{\text{H}} = V_{\text{IH,min}} - V_{\text{OH,min}} = 2.0 - 2.4\text{ V} = -0.4\text{ V}$$
$$NM_{\text{L}} = V_{\text{IL,max}} - V_{\text{OL,max}} = 0.8 - 0.4 = 0.4\text{ V}$$

For CMOS 74HCXX family, from Fig. 1.41

$$NM_{\text{H}} = V_{\text{IH,min}} - V_{\text{OH,min}} = 3.5 - 4.9\text{ V} = -1.4\text{ V}$$
$$NM_{\text{L}} = V_{\text{IL,max}} - V_{\text{OL,max}} = 1.5 - 0.1 = 1.4\text{ V}$$

Note that CMOS has better noise margin characteristics over TTL.

## SUMMARY

This introductory chapter in *digital principles* is intended to provide a clear concept of a digital signal in an electronic circuit or system. Both ideal and realistic digital voltage levels are presented. How these levels vary with time (waveforms) is illustrated, and the concepts of rise time, fall time, and duty cycles are defined. Simple ideal switch models are used to illustrate digital circuit operation. Symbols and operation of the following basic digital circuits are presented: buffer, tri-state buffer, inverter, tri-state inverter, AND gate, and OR gate. The flip-flop is introduced as a basic memory element, and both serial and parallel shift registers are discussed. The basic conceptual operation of a number of common MSI and LSI digital circuits is covered: encoders, decoders, multiplexers, demultiplexers, ALUs, counters, and comparators. These basic elements are then discussed in the context of their use in a simple digital computer. Finally, currently available digital ICs, including the 54/74XX TTL, 74HCXX CMOS, and MECL families, are discussed.

## GLOSSARY

- ***ALU*** Arithmetic logic unit.
- ***analog signal*** A signal whose amplitude can take any value between given limits. A continuous signal.
- ***binary number*** A number code that uses only the digits 0 and I to represent quantities.
- ***bipolar*** Having two types of charge carriers; a bipolar transistor is *npn* or *pnp*.
- ***bit*** *b*inary dig*it*.
- ***buffer*** A digital circuit capable of maintaining a required logic level while acting as a current source or a current sink for a given load.
- ***chip*** A small piece of semiconductor on which an IC is formed.
- ***CMOS*** Complementary metal-oxide silicon. An IC using both *n*-channel and *p*-channel field-effect transistors (FETs).
- ***CPU*** Central processing unit.
- ***CRT*** Cathode-ray tube.
- ***clock*** A periodic, rectangular waveform used as a basic timing signal.
- ***computer architecture*** Microprocessor and other elements building a computer.
- ***counter*** A digital circuit designed to keep track of (to count) a number of events.
- ***decoder*** A unit designed to change a digital number into another form.
- ***demultiplexer (DEMUX)*** A digital circuit that will select only one of many inputs.
- ***digital signal*** A signal whose amplitude can have only given discrete values between defined limits. A signal that changes amplitude in discrete steps.
- ***DIP*** Dual-inline package.
- ***DMA*** Direct memory access.
- ***Duty cycle*** For a periodic digital signal, the ratio of high level time to the period or the ratio of low level time to the period.
- ***ECL*** Emitter-coupled logic.
- ***encoder*** A unit designed to change a given signal into a digital number.
- ***flip-flop*** An electronic circuit that can store one bit of a binary number.
- ***floppy disk*** A magnetically coated disk used to store digital data.
- ***gate*** A digital circuit having two or more inputs and a single output.
- ***handshaking*** A "request" to transfer data into or out of a computer, followed by an "acknowledge" signal, allowing data transfer to begin.
- ***IC*** Integrated circuit.
- ***logic circuit*** A digital circuit, a switching circuit, or any kind of two-state circuit that duplicates mental processes.
- ***LSI*** Large-scale integration.
- ***memory*** The area of a digital computer used to store programs and data.

- ***memory element*** Any device or circuit used to store 1 bit of a binary number.
- ***microprocessor*** An IC around which many small computer systems are constructed.
- ***MSI*** Medium-scale integration.
- ***multiplexer (MUX)*** A digital circuit that will connect a single input to any one of many possible outputs.
- ***noise margin*** Allowable level of additive noise for proper interpretation of logic value.
- ***parallel shifting*** Transferring all bits in a binary number (digital data) simultaneously.
- ***port*** A register that serves as a place to either input data to or extract data from a digital system.
- ***program*** A detailed set of instructions used to direct the operation of a computer.
- ***programmer*** A person who writes programs for digital computer systems.
- ***Schottky diode*** A special kind of diode that can be very rapidly switched on and off. The combination of a Schottky diode with a bipolar transistor is called a *Schottky transistor.*
- ***serial shifting*** Transferring each bit in a binary number (digital data), one bit at a time, one after the other.
- ***SSI*** Small-scale integration.
- ***tri-state circuit*** A digital circuit having three states—high, low, and open.
- ***truth table*** A table that shows all of the input-output possibilities of a digital circuit.
- ***TTL*** Transistor-transistor logic. The widely used 54XX/74XX family of bipolar junction transistor (BJT) integrated circuits.
- ***VLSI*** Very large-scale integration.

## PROBLEMS

### Section 1.1

1.1 Describe in your own words the characteristics of an analog signal and a digital signal.

1.2 Use four indicator lamps to illustrate the decimal number 3.

1.3 Demonstrate that only three indicator lamps are required to display the eight decimal numbers 0. 1, 2, 3. 4, 5, 6, and 7.

1.4 On certain wall clocks, the representation of the progression of seconds is analog, while on other clocks it is digital. What is the difference?

### Section 1.2

1.5 Make a sketch similar to Fig. 1.4c to illustrate that $V_{OH,min}$ = 3.9 Vdc, $V_{OL,max}$ = 0.8 Vdc, +5 Vdc $\geq H \geq V_{OH,min}$, and 0 Vdc $\leq L \leq V_{OL,max}$.

1.6 A certain digital logic family has levels which are given as 0 Vdc $\geq H \geq -0.2$ Vdc and $-2.0$ Vdc $\geq L \geq -2.5$ Vdc. Make a sketch similar to Fig. 1.4c to illustrate the allowed logic levels. (This is similar to the popular ECL family of digital circuits.) Is this positive or negative logic?

1.7 The waveform in Fig. 1.6b has a duty cycle $H$ = 20 percent, and the positive pulses occur every 500 μs. What is the width of each positive pulse?

1.8 Draw a waveform similar to Fig. 1.6b if $H$ = +5 Vdc, $L$ = 0 Vdc, and Duty cycle $H$ = 90 percent. Does this resemble a series of negative pulses?

1.9 Make a sketch of an ideal symmetrical 1-MHz square wave having $H$ = +5 Vdc and $L$ = 0 Vdc (similar to Fig. 1.6). Directly under this waveform, sketch a nonideal waveform having the same values but with a rise time of 0.5 μs and a fall time of 0.5 μs. What has happened to the square wave?

## Section 1.3

1.10 Construct a truth table for a 3-input AND gate. *Hint:* Use the binary numbers in Table 1.1.

1.11 Construct a truth table for a 3-input OR gate. *Hint:* Use the binary numbers in Table 1.1.

1.12 Determine the state of $V_o$ in Fig. 1.43.

1.13 In order to obtain a digital signal at $V_o$, in Fig. 1.44, $G$ must be high or low. What must the state of $V_i$ and $G$ in Fig. 1.44 be if $V_o = L$?

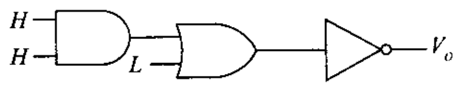


Fig. 1.43

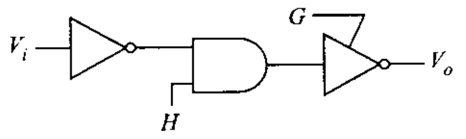


Fig. 1.44

## Section 1.4

1.14 Draw a circuit that can be used to SET and RESET the flip-flop in Fig. 1.16. Use only one switch and one inverter. With the switch in one position, the flip-flop will SET. Moving the switch to the other position will RESET the flip-flop.

1.15 Explain how two 4-bit serial registers could be used to form a single 8-bit serial register. Draw a diagram to illustrate this.

1.16 A 16-bit serial register requires 500 ns for each shift operation. How much time is required to enter or extract a 16-bit binary number?

1.17 A modern computer uses a 32-bit microprocessor and shifts data in parallel between its microprocessor and the input-output port register. How many connections (wires) must be made between the microprocessor and the register if the data are truly shifted in parallel?

To reduce the number of connections, data are sometimes shifted in groups of 2. That is, the first 16 bits are shifted in parallel, and then the next 16 bits are shifted in parallel. If this is done, the number of connections is cut in half. How many connections are needed here if this is done?

Compare the times required to shift a 32-bit number for each case if one shift operation requires 250 ns.

## Section 1.5

1.18 What is the largest decimal count possible with a 7-flip-flop counter?

1.19 If both the $A$ and the $B$ inputs to the ALU in Fig. 1.22 are 4 bits, what is the largest single-digit decimal number that can be represented? What if the buses were only three bits each?

1.20 The ALU in Fig. 1.22 is set to perform an AND function. What is $F$ if $A = 0101$ and $B = 0100$? What is $F$ if the ALU is changed to the OR function and $A$ and $B$ remain the same?

1.21 Write binary values for $F$ and CARRY OUT shown in Fig. 1.23 if $A = 0011$ and $B = 0001$.

1.22 Determine the output logic levels for the comparator in Fig. 1.24 if

a. $A = 0111, B = 0111$
b. $A = 1000, B = 0111$
c. $A = 0011, B = 0100$

1.23 Show the proper logic levels on the decoder in Fig. 1.26b if the digital input is $ABCD = 1001$.

## Section 1.6

1.24 Give definitions for the following abbreviations:

a. ALU b. CPU
c. ECL

1.25 Discuss the term *handshaking* in reference to the block diagram in Fig. 1.32.

1.26 What are the basic blocks included in the architecture of a typical microprocessor?

1.27 Why must the bus connecting the CPU and the data memory be bidirectional?

1.28 Draw a block diagram, similar to Fig. 1.27, of a machine, process, or system (similar to the lawn-watering system discussed) that is controlled by a computer. Describe any sensors needed, and give a general discussion of the system operation.

## Section 1.7

1.29 You are asked to recommend a family of TTL digital circuits that will operate at temperatures as high as 50°C and as low as −10°C. Power requirements are to be kept as low as possible, and the system is to operate at the highest possible clock frequency. What would you recommend?

1.30 A co-worker asks you to explain the functions of the TTL designated as a 7404 and also wants to know the difference between a 7404 and a 74HC04.

1.31 You are designing a system that uses 250 standard 74XX TTL gates. What is the total power required?

1.32 Repeat Prob. 1.31 using 74LSXX gates.

1.33 Draw from memory the standard symbols and the new IEEE/IEC symbols for an inverter, a 2-input AND gate, and a 2-input OR gate.

## Answers to Self-tests

1. continuous
2. T
3. 0111
4. Positive logic
5. The term $V_{OH,min}$ stands for the minimum value of the output voltage when *high*. The term $V_{OL,max}$ stands for the maximum value of the output voltage when low.
6. $V_o$ may have a value within the forbidden region only during the short time while transitioning from high to low or low to high. When not switching (static), $V_o$ must either be in the high band or the low band.
7. $0.9H = 4.5$ Vdc, and $1.1L = 1.1$ Vdc.
8. Duty cycle $H = 2/7 = 0.286 = 28.6$ percent.
9. $V_o$ is open in the first instance, and then it is high.
10. $V_o$ is high in the first instance, and then it is open.
11. $V_o$ is low.
12. $V_o$ would be low. To produce $V_o = H$, *all* three inputs must be high.
13. $V_o$ would be high. To produce $V_o = L$, *all* three inputs must be low.
14. The possibilities are almost endless!
15. $A = L$
16. This is a 4-bit number. $DCBA = 1001$.
17. It would take only 1 µs for the parallel register, but it would require 8 µs for the serial register.
18. A port is usually a register that serves as a place to either input data to or extract data from the microprocessor or computer.
19. 0111
20. Ten
21. a. No b. Yes c. Yes
22. 0110
23. The clock provides a periodic digital signal that is used to time computer operations.
24. It is a specific list of instructions, prepared by a programmer, that directs the actions of the computer.
25. Arithmetic computations and logical decisions.

26. The DMA allows the transfer of data directly between memory and external devices, without passing through the CPU.
27. SSI, less than 12; MSI, more than 12 but less than 100; LSI, more than 100.
28. ECL is faster but requires more power.
29. 0°C to 75°C
30. *Quad* means "four." *Hex* means "six."
31. *A*—advanced, *C*—CMOS, *T*—TTL compatible.
32. The output is active when low.

# Digital Logic

# 2

OBJECTIVES

- ✦ **Write the truth tables for, and draw the symbols for, 2-input OR, AND, NOR, and NAND gates.**
- ✦ **Write Boolean equations for logic circuits and draw logic circuits for Boolean equations.**
- ✦ **Use DeMorgan's first and second theorems to create equivalent circuits.**
- ✦ **Understand the operation of AND-OR-INVERT gates and expanders.**

A digital circuit having one or more input signals but only one output signal is called a *gate*. In Chapter 1, the most basic gates—the NOT gate (inverter), the OR gate and the AND gate—were introduced. Connecting the basic gates in different ways makes it possible to produce circuits that perform arithmetic and other functions associated with the human brain (an ALU). Because they simulate mental processes, gates are often called *logic circuits*. A discussion of both positive and negative logic leads to the important concept of *assertion-level logic*.

Hardware description languages (HDL) are an alternative way of describing logic circuits. This uses a set of textual codes that is machine (computer) readable. The concept is relatively new and is useful for design, testing and fabrication of complex digital circuits. We'll have a soft introduction of HDL towards the end of this chapter. We'll learn it in detail in later part of this book introducing features relevant to each chapter.

## 2.1 THE BASIC GATES—NOT, OR, AND

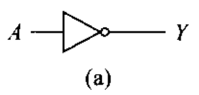
Is an action right or wrong? A motive good or bad? A conclusion true or false? Much of our thinking involves trying to find the answer to two-valued questions like these. Two-state logic had a major influence on Aristotle, who worked out precise methods for getting to the truth. Logic next attracted mathematicians, who intuitively sensed some kind of algebraic process running through all thought.

Augustus De Morgan came close to finding the link between logic and mathematics. But it was George Boole (1854) who put it all together. He invented a new kind of algebra that replaced Aristotle's verbal methods. Boolean algebra did not have an impact on technology, however, until almost a century later. In 1938 Shannon applied the new algebra to telephone switching circuits. Because of Shannon's work, engineers soon realized that Boolean algebra could be used to analyze and design computer circuits.

Three logic circuits, the *inverter*, the *OR gate*, and the *AND gate*, can be used to produce any digital system. The function of each of these gates was introduced in Chapter 1. Let's look more closely at the operation of each circuit and also at their Boolean expressions.

## The Inverter (NOT Gate)

Figure 2.1 is the symbol and truth table for an inverter. In one truth table, the symbols $H$ and $L$ are used, while the binary numbers 0 and 1 are used in the other. The information in each table is identical, however, since we know $L = 0$ and $H = 1$. In this text, both symbols are used, hence since there is no chance for confusion. You will find both symbols used in other texts, as well as in manufacturers' data sheets. The important idea is that there are only two possible voltage levels (low and high) associated with a digital circuit. This fits nicely with the binary number system, since it has only two values (0 and 1). This is often referred to as *two-state operation.* By definition, this is *positive logic*, since the *higher* voltage level is assigned binary 1. Later in this chapter, we will consider *negative logic*, where the *higher* voltage level is assigned binary (zero).



(a)

| $A$ | $Y$ |
|---|---|
| $L$ | $H$ |
| $H$ | $L$ |

| $A$ | $Y$ |
|---|---|
| 0 | 1 |
| 1 | 0 |

(b)

**Fig. 2.1 (a) Inverter symbol, (b) Truth tables**

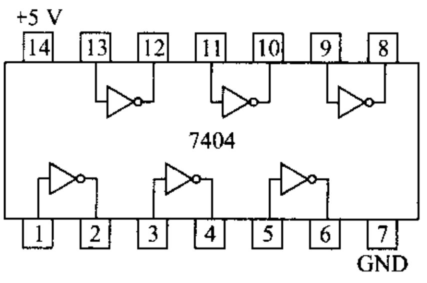


**Fig. 2.2 Pinout diagram of a 7404**

Figure 2.2 shows the pinout diagram of a 7404 hex inverter. This IC contains six inverters. After applying +5 Vdc (the supply voltage for all TTL devices) to pin 14 and grounding pin 7, you can connect any or all inverters to other TTL devices. For instance, if you only need one inverter, you can connect an input signal to pin 1 and take the output signal from pin 2; the other inverters can be left unconnected.

In Boolean algebra a variable can be either 0 or 1. The output $Y$ of not gate is always complement of input $A$. In equation form

$$Y = \text{not } A \quad \text{i.e.} \quad Y = A' \quad \text{so that, if } A = 0,\ Y = 0' = 1 \quad \text{and if} \quad A = 1,\ Y = 1' = 0$$

The truth tables in Fig. 2.1 illustrate signal levels that do not change with time. However, almost all digital signals do in fact change with time, as illustrated by the waveforms in Chapter 1 (Sec. 1.2). Here are two examples that illustrate how to use the truth table information with signals that vary with time.

**Example 2.1** A 1-kHz square wave drives pin 1 of a 7404 (see Fig. 2.2). What does the voltage waveform at pin 2 look like?

*Solution* Figure 2.3a shows what you will see on a dual-trace oscilloscope. Assuming you have set the sweep timing to get the upper waveform (pin 1), then you would see an inverted square wave on pin 2.

**Example 2.2** If a 500-Hz square wave drives pin 3 of a 7404, what is the waveform on pin 4?

*Solution* Pins 3 and 4 are the input and output pins of an inverter (see Fig. 2.2). A glance at Fig. 2.3b shows the typical waveforms on the input (pin 3) and output (pin 4) of a 7404. Again, the output waveform is the complement of the input waveform. Because of two-state operation, rectangular waveforms like this are the normal shape of digital signals. Incidentally, a *timing diagram* is a picture of the input and output waveforms of a digital circuit. Examples of timing diagrams are shown in Figs. 2.3a and b.

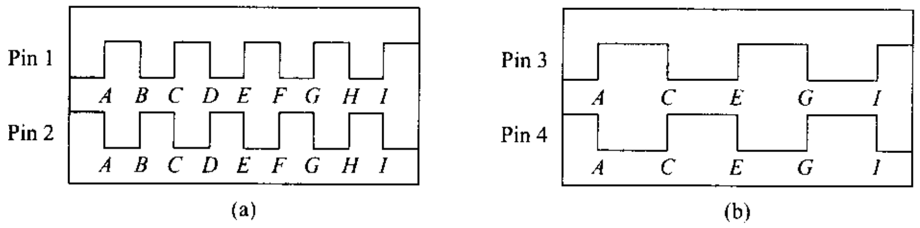
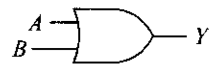


**Fig. 2.3**

## OR Gates

An OR gate has two or more input signals but only one output signal. It is called an OR gate because the output voltage is high if any or all of the input voltages are high. For instance, the output of a 2-input OR gate is high if either or both inputs are high. Figure 2.4a shows the logic symbol of a 2-input OR gate and Fig. 2.4b its truth table.



| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

(a) (b)

**Fig. 2.4** (a) OR gate, (b) Truth table

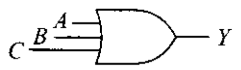
In Boolean equation form

$$Y = A \text{ OR } B, \quad \text{i.e.} \quad Y = A + B$$

so that $Y = 0 + 0 = 0$, $Y = 0 + 1 = 1$, $Y = 1 + 0 = 1$ and $Y = 1 + 1 = 1$.

The '+' sign here represents logic operation OR and not addition operation of basic arithmetic. Note that in arithmetic $1 + 1 = 2$ in decimal and $1 + 1 = 10$ in binary number system (Table 1.1 of Chapter 1). Binary addition is discussed in detail in Chapter 6.

**Three Inputs** Figure 2.5 shows a 3-input OR gate. The inputs are $A$, $B$, and $C$. When all inputs are low, $Y$ is low. If $A$ or $B$ or $C$ is high, $Y$ will be high. The truth table summarizes all input possibilities. In equation form, the three input OR gate is represented as: $Y = A + B + C$.



| $A$ | $B$ | $C$ | $Y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

(a) (b)

**Fig. 2.5** (a) Three-input OR gate, (b) Truth table

The truth table (Fig. 2.5b) allows us to check that all input possibilities are included. Why? Because every possibility is included when the input entries follow a binary sequence. For example, the first $ABC$ entry is 000, the next is 001, then 010, and so on, up to the final entry of 111. Since all binary numbers are present, all input possibilities are included.

Incidentally, the number of rows in a truth table equals $2^n$, where $n$ is the number of inputs. For a 2-input OR gate, the truth table has $2^2$, or 4 rows. A 3-input OR gate has a truth table with $2^3$, or 8 rows, while a 4-input OR gate results in $2^4$, or 16 rows, and so on.

**An OR gate can have** as many inputs as desired. No matter how many inputs, the action of any OR gate is summarized like this: One or more high inputs produce a high output.

**Logic Symbols** Figure 2.6a shows the symbol for a 2-input OR gate of any design. Whenever you see this symbol, remember the output is high if either input is high.

Shown in Fig. 2.6b is the logic symbol for a 3-input OR gate. Figure 2.6c is the symbol for a 4-input OR gate. For these gates, the output is high when any input is high. The only way to get a low output is by having all inputs low.

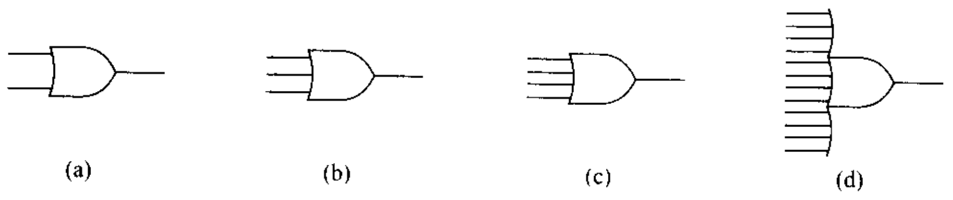


**Fig. 2.6 OR gate symbols: (a) Two-input, (b) Three-input, (c) Four input, (d) Twelve-input**

When there are many input signals, it's common drafting practice to extend the input side as needed to allow sufficient space between the input lines. For instance, Fig. 2.6d is the symbol for a 12-input OR gate. The same idea applies to any type of gate; extend the input side when necessary to accommodate a large number of input signals.

**TTL OR Gates** Figure 2.7 shows the pinout diagram of a 7432, a TTL quad 2-input OR gate. This digital IC contains four 2-input OR gates inside a 14-pin DIP. After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more of the OR gates to other TTL devices.

**Timing Diagram** Figure 2.8 shows an example of a timing diagram for a 2-input OR gate. The input voltages drive pins 1 and 2 of a 7432. Notice that the output (pin 3) is low only when both inputs are low. The output is high the rest of the time because one or more input pins are high.

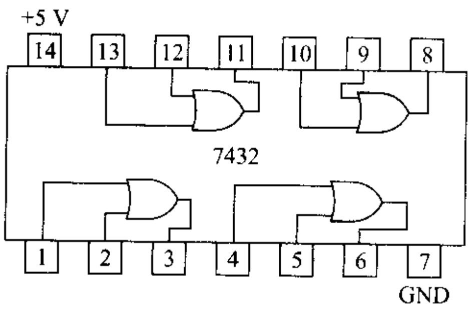


**Fig. 2.7 Pinout diagram of a 7432**

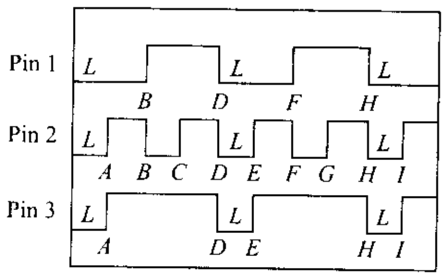


**Fig. 2.8 Timing diagram**

**Example 2.3** Work out the truth table for Fig. 2.9a.

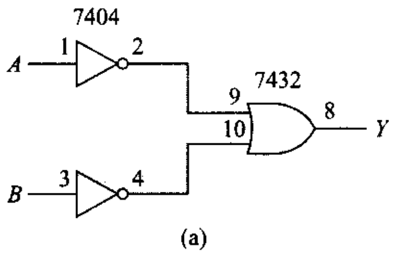
*Solution* With two input signals ($A$ and $B$), four input cases are possible: low-low, low-high, high-low, and high-high. For convenience, let $L$ stand for low and $H$ for high. Then, the input possibilities are $LL$, $LH$, $HL$, and $HH$, as listed in Fig. 2.9b. Here is what happens for each input possibility.

CASE 1 $A$ is low and $B$ is low. With both input voltages in the low state, each inverter has a high output. This means that the OR gate has a high output, the first entry of Fig. 2.9b.

CASE 2 $A$ is low and $B$ is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output. Since the OR gate still has a high input, the output $Y$ is high.

CASE 3 $A$ is high and $B$ is low. Now, the upper inverter has a low output and the lower inverter has a high output. Again, the OR gate produces a high output, so that $Y$ is high.

CASE 4 $A$ is high and $B$ is high. With both inputs high, each inverter has a low output. This time, the OR gate has all inputs in the low state, so that $Y$ is low, as shown by the final entry of Fig. 2.9b.



| $A$ | $B$ | $Y$ |
|---|---|---|
| $L$ | $L$ | $H$ |
| $L$ | $H$ | $H$ |
| $H$ | $L$ | $H$ |
| $H$ | $H$ | $L$ |

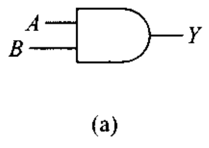
(b)

**Fig. 2.9 Logic circuit and truth table of Example 2.3**

Incidentally, the circuit of Fig. 2.9a uses only one-third of a 7404 and one-fourth of a 7432. The other gates in these digital ICs are not connected, which is all right because you don't have to use all of the available gates.

## AND Gates

The AND gate has a high output only when all inputs are high. Figure 2.10a shows a 2-input AND gate. The truth table (Fig. 2.10b) summarizes all input-output possibilities for a 2-input AND gate. Examine this table carefully and remember the following: the AND gate has a high output only when $A$ and $B$ are high. In other words, the AND gate is an all-or-nothing gate; a high output occurs only when all inputs are high. This truth table uses 1s and 0s, where $1 = H$ and $0 = L$.



| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 1 |

(b)

**Fig. 2.10 (a) Two-input AND gate, (b) Truth table**

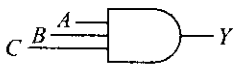
In Boolean equation form

$$Y = A \text{ AND } B, \text{ i.e. } Y = A.B \text{ or } Y = AB$$

so that, $Y = 0.0 = 0$, $Y = 0.1 = 0$, $Y = 1.0 = 0$ and $Y = 1.1 = 1$

The '.' sign here represents logic AND operation and not multiplication operation of basic arithmetic though the result are same for both.

**Three Inputs** Figure 2.11a shows a 3-input AND gate. The inputs are $A$, $B$, and $C$. When all inputs are low, $Y$ is low. If even one input is low, $Y$ is in the low state. The only way to get a high output is to raise all inputs to the high state (+5 V) The truth table (Fig. 2.11b) summarizes all input-output possibilities. In equation form, the three input AND gate is represented as: $Y = A.B.C = ABC$.



(a)

| $A$ | $B$ | $C$ | $Y$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 |

(b)

**Fig. 2.11 (a) Three-input AND gate, (b) Truth table**

**Logic Symbols** Figure 2.12a shows the symbol for a 2-input AND gate of any design. Shown in Fig. 2.12b is the logic symbol for a 3-input AND gate. Figure 2.12c is the symbol for a 4-input AND gate. Remember: For any of these gates, the output is high only if all inputs are high. As before, it's common drafting practice to extend the input sides when there are many input signals. For instance, Fig. 2.12d is the symbol for a 12-input AND gate.

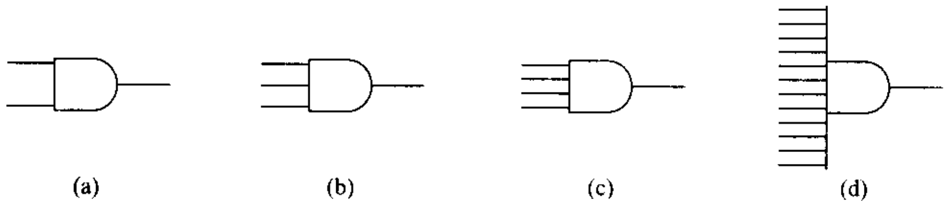


**Fig. 2.12 AND gate symbols: (a) Two-input, (b) Three-input, (c) Four-input, (d) Twelve-input**

**TTL AND Gates** Figure 2.13 shows the pinout diagram of a 7408, a TTL quad 2-input AND gate. This digital IC contains four 2-input AND gates. After connecting a supply voltage of +5V to pin 14 and a ground to pin 7, you can connect one or more of the AND gates to other TTL devices. TTL AND gates are also available in triple 3-input and dual 4-input packages. (See Appendix 3 for pinout diagrams.)

**Timing Diagram** Figure 2.14 shows an example of a timing diagram for a 2-input AND gate. The input voltages drive pins 1 and 2 of a 7408. Notice that the output (pin 3) is high only when both inputs are high (between $C$ and $D$, $G$ and $H$, etc.). The output is low the rest of the time.

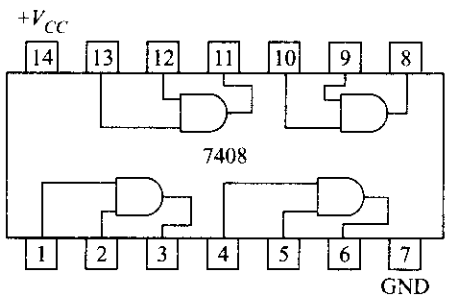


**Fig. 2.13 Pinout diagram of a 7408**

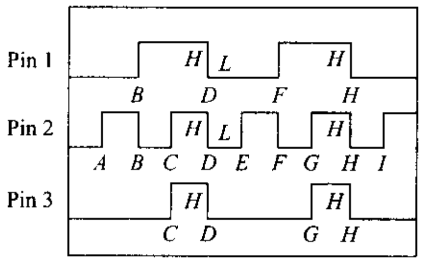
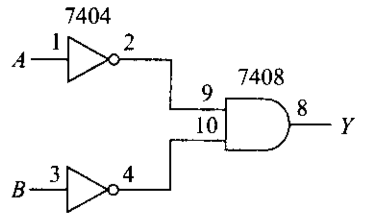


**Fig. 2.14 Timing diagram**

**Example 2.4** Work out the truth table for Fig. 2.15a.



(a)

| $A$ | $B$ | $A'$ | $B'$ | $Y = A'.B'$ |
|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 1 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |

(b)

**Fig. 2.15** **Logic circuit and truth table of Example 2.4**

*Solution* We get the final truth table here in slightly different way. Consider, one logic gate at a time as shown in Fig. 2.15b. The NOT gate connected to $A$ gives $A'$ at its output and is shown in column 3. The NOT gate connected to $B$ gives $B'$ at its output and is shown in column 3. Finally, the 4th column shows OR operation on column 3 and 4 to give the final output $Y$. Here is what happens for each input possibility.

CASE 1 $A$ is low and $B$ is low. With both input voltages in the low state, each inverter has a high output. This means the AND gate has a high output, the first entry of Fig. 2.16.

CASE 2 $A$ is low and $B$ is high. With these inputs the upper inverter has a high output, while the lower inverter has a low output. Since the AND gate produces a low output, $Y$ is low.

CASE 3 $A$ is high and $B$ is low. Now, the upper inverter has a low output and the lower inverter has a high output. Again, the AND gate produces a low output, so $Y$ is low.

CASE 4 $A$ is high and $B$ is high. With both inputs high, each inverter has a low output. Again, the AND gate has a low output, as shown by the final entry of Fig. 2.16.

| $A$ | $B$ | $Y$ |
|---|---|---|
| $L$ | $L$ | $H$ |
| $L$ | $H$ | $L$ |
| $H$ | $L$ | $L$ |
| $H$ | $H$ | $L$ |

**Fig. 2.16**

Note that input-output relations described in Fig. 2.15b and Fig. 2.16 are same.

**Example 2.5** What is the Boolean equation for the logic circuit of Fig. 2.17a?

*Solution* This circuit is called an AND-OR network because input AND gates drive an output OR gate. The intermediate outputs are

$$Y_3 = AB \qquad Y_6 = CD$$

The final output is

$$Y_8 = Y_3 + Y_6$$
$$Y = AB + CD$$

An equation in this form is referred to as a *sum-of-products equation*. AND-OR networks always produce sum-of-products equations.

**Example 2.6** Write the Boolean equation for Fig. 2.17b.

*Solution* This logic circuit is called an OR-AND network because input OR gates drive an output AND gate. The intermediate outputs are $Y_8 = A + B$ and $Y_{11} = C + D$.

The final output is

$$Y_6 = Y_8 Y_{11}$$

or

$$Y = (A + B)(C + D)$$

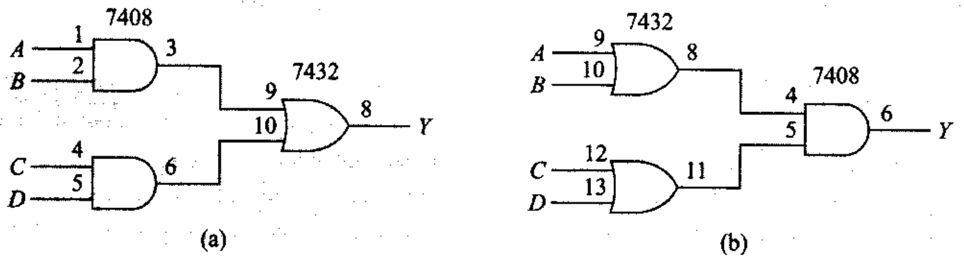


Fig. 2.17 (a) AND-OR, (b) OR-AND network

As shown in this equation, parentheses may be used to indicate a logical product (ANDing). Also notice that the final answer is a product of sums. OR-AND networks always produce *product-of-sums equations*.

**Example 2.7** What is the logic circuit whose Boolean equation is

$$Y = \bar{A}BC + A\bar{B}C$$

*Solution* This is a sum-of-products equation with some of the inputs in complemented form. Figure 2.18a shows an AND-OR circuit with the foregoing Boolean equation. The upper AND gate produces a logical product of

$$Y_{12} = \bar{A}BC$$

The lower AND gate produces

$$Y_6 = A\bar{B}C$$

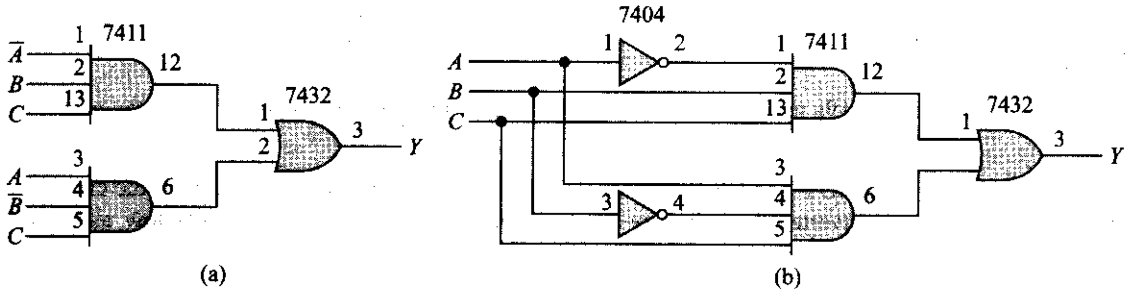


Fig. 2.18 (a) Intermediate, (b) Final logic circuit of Example 2.7

**SELF-TEST**

1. A system in which $H = 1$ and $L = 0$ is (positive, negative) logic.
2. A gate whose output is $H$ if any input is $H$ is an _______ gate.
3. A gate whose output is $H$ only when all inputs are $H$ is an _______ gate.

The final output therefore equals the sum of the $Y_{12}$ and $Y_6$ products:

$$Y = \overline{A}BC + A\overline{B}C$$

The complemented inputs $A$ and $B$ may be produced by other circuits (discussed later). Alternatively, inverters on the $A$ and $B$ input lines may produce the complemented variables, as shown in Fig. 2.18b.

SELF-TEST

4. Write an expression for an inverter, or NOT gate equivalent to $Y$ = not $A$.
5. Write a Boolean expression for an OR gate having $A$ and $B$ as inputs and $Y$ as the output.
6. Write the Boolean expression for an AND gate with $A$ and $B$ as inputs and $Y$ as the output.

This example illustrates one method of logic design. Whenever you are given a sum-of-products equation, you can draw the corresponding AND-OR network using AND gates to produce the logical products and an OR gate to produce the sum.

## 2.2 UNIVERSAL LOGIC GATES—NOR, NAND

In the previous section we have seen how AND, OR and NOT gates can be connected together to realize any logic function. Here, we address an interesting question. Is it possible to use only one type of gate for this purpose? If possible, one needs to procure only one type of gate for his design. And more importantly, fabrication of Integrated Circuit that performs a logic operation becomes easier when gate of only one kind is used. Gates, which can perform this task, are called universal logic gates. Clearly, basic gates like AND, OR and NOT don't fit into this category for the simple reason that conversion among themselves itself are not possible. As for example, one cannot gate OR operation by using any number or combination of AND gates. In this section, we discuss two universal logic gates NOR and NAND.

### NOR Gates

The logic circuit of Fig. 2.19a used to be called a NOT-OR gate because the output is

$$Y = \overline{A + B}$$

Read this as "$Y$ equals NOT $A$ OR $B$" or "$Y$ equals the complement of $A$ OR $B$." Because the circuit is an OR gate followed by an inverter, the only way to get a high output is to have both inputs low, as shown in the truth table of Table 2.1.

### NOR Gate Symbol

The logic circuit of Fig. 2.19a has become so popular that the abbreviated symbol of Fig. 2.19b is used for it. The bubble (small circle) on the output is a reminder of the inversion that takes place after the ORing. Furthermore, the words NOT-OR are contracted to the word NOR. So from now, we will call the circuit a NOR gate and will use the symbol of Fig. 2.19b. Whenever you see this symbol, remember that the output is NOT the OR of the inputs. With a NOR gate, all inputs must be low to get a high output. If any input is high, the output is low.

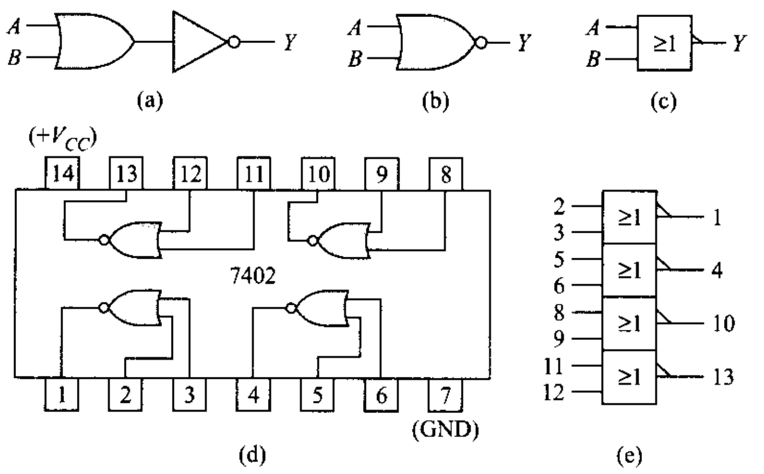


Fig. 2.19 NOR logic gate

Figure 2.19c shows the new IEEE rectangular symbol for the NOR gate. The small triangle on the output is equivalent to the bubble used on the standard symbol. The indicator ≥ inside the box means "if one or more of the inputs are high, the output is high."

The 7402 is a quad 2-input NOR gate in a 14-pin DIP as illustrated in Fig. 2.19d. The new rectangular symbol for the 7402 is shown in Fig. 2.19e.

Table 2.1 NOR Gate

| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 0 |
| 1 | 0 | 0 |
| 1 | 1 | 0 |

## Bubbled AND Gate

Figure 2.20a shows inverters on the input lines of an AND gate. This logic circuit is often drawn in the abbreviated form shown in Fig. 2.20b. The bubbles on the inputs are a reminder of the inversion that takes place before the AND operation. We will refer to the abbreviated drawing of Fig. 2.20b as *a bubbled AND gate.* We have already analyzed this circuit in Example 2.4 and obtained its truth table as shown in Fig. 2.15b. We find that output $Y$ and inputs $A$, $B$ are identical for bubbled AND gate and NOR gate. Therefore, these two circuits are equivalent and thus interchangeable. Given any logic circuit with NOR gates, we can replace it by bubbled AND gates and converse is also true.

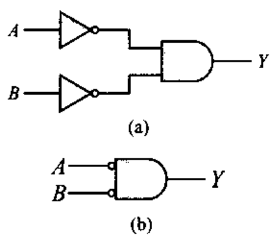


Fig. 2.20 (a) AND gate with inverted inputs, (b) Equivalent symbol

## De Morgan's First Theorem

The Boolean equation for Fig. 2.19b is

$$Y = \overline{A + B}$$

The Boolean equation for Fig. 2.20b is

$$Y = \overline{A}\,\overline{B}$$

The first equation describes a NOR gate, and the second equation a bubbled AND gate. Since the outputs are equal for the same inputs, we can equate the right-hand members to get

$$\overline{A+B} = \overline{A}\,\overline{B} \tag{2.1}$$

This identity is known as *De Morgan's first theorem*. In words, it says the complement of a sum equals the product of the complements. This can also be proved by comparing the truth tables shown in Fig. 2.4(b) and NOR gate truth table of Table 2.1. A similar exercise that compares truth tables of three input NOR gate and three input bubbled AND gate show they are identical and we can write, $(A + B + C)' = A'B'C'$. Note that this equivalence can be extended to gates or circuits for larger number of inputs, too.

## Universality of NOR Gate

Figure 2.21 shows how all other logic gates can be obtained from NOR gates. To get a NOT gate we tie inputs of NOR gate together (Fig. 2.21a) so that there is only one input to the circuit. If input is 0, then both the inputs to NOR gate are 0. Following NOR gate truth table (Table 2.1) we see output now is 1. Similarly, if input is 1, both the inputs to NOR gate are 1 that gives output 0. Therefore output of circuit, shown in Fig. 2.21a is complement of its input and thus gives NOT operation.

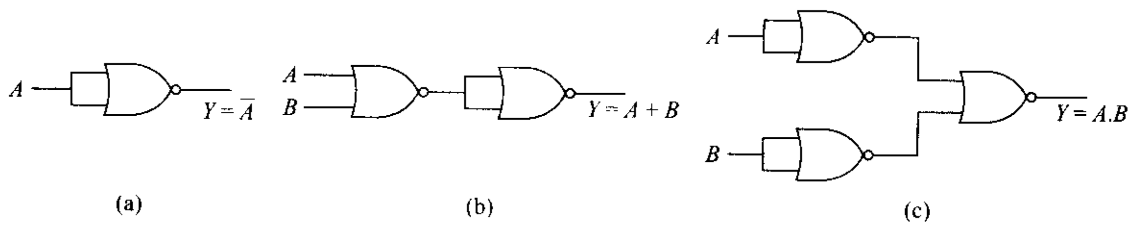


**Fig. 2.21 Universality of NOR gate (a) NOT from NOR, (b) OR from NOR, (c) AND from NOR**

Figure 2.21b shows how to get OR circuit using only NOR gates. The first NOR gate performs usual NOR operation while second NOR gate performs as NOT gate and inverts the NOR logic to OR.

To understand how we get AND circuit using only NOR gates (Fig. 2.21c) let us refer to example 2.3. The configuration is similar except the output there is generated from OR and here from NOR and of course the NOT gates are replaced by NOR equivalent. Since NOR gate is NOT operation followed by OR we invert the output of example 2.3, shown in Fig. 2.9b to get output of this circuit. Thus output of circuit in Fig. 2.21c is high only when both the inputs are high and it functions like an AND gate.

The above equivalences can be proved simply, by applying Boolean theorems and we'll discuss those theorems in next chapter. Since, we can perform all the Boolean operations using only NOR gates it is termed as universal logic gate.

## Eye of the Beholder

Which brings us to a principle. Truth tables, logic circuits, and Boolean equations are different ways of looking at the same thing. Whatever we learn from one viewpoint applies to the other two. If we prove that truth tables are identical, this immediately tells us the corresponding logic circuits are interchangeable, and their Boolean equations are equivalent. When analyzing, we generally start with a logic circuit, construct its truth table, and summarize with the Boolean equation. When designing, we often start with a truth table, generate a Boolean equation, and arrive at a logic circuit.

**Example 2.8** A 7402 is a quad 2-input NOR gate. This TTL IC has four 2-input NOR gates in a 14-pin DIP as shown in Appendix 3. What is the Boolean equation for the output of Fig. 2.22a?

*Solution* The AND gates produce $AB$ and $CD$. These are ORed to get $AB + CD$. The final inversion gives

$$Y = \overline{AB + CD}$$

The circuit of Fig. 2.22a is known as an AND-OR-INVERT network because it starts with ANDing, follows with ORing, and ends with INVERTing.

The AND-OR-INVERT network is available as a separate TTL gate. For instance, the 7451 is a dual 2-input 2-wide AND-OR-INVERT gate, meaning two networks like Fig. 2.22a in a single 14-pin TTL package. Appendix 3 shows the pinout diagram. Figure 2.22b shows how we can use half of a 7451 to produce the same output as the circuit of Fig. 2.22a.

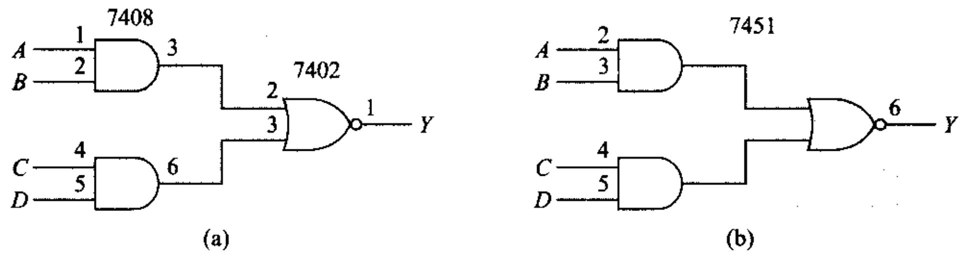


**Fig. 2.22** AND-OR-INVERT network

**Example 2.9** Prove that Fig. 2.23c is logically equivalent to Fig. 2.23a.

*Solution* De Morgan's first theorem says we can replace the final NOR gate of Fig. 2.23a by a bubbled AND gate to get the equivalent circuit of Fig. 2.23b. If you invert a signal twice, you get the original signal back again. Put another way, double inversion has no effect on the logic state; double invert a low and you still have a low; double-invert a high and you still have a high. Therefore, each double inversion in Fig. 2.23b (a pair of bubbles on the same signal line) cancels out, leaving the simplified circuit of Fig. 2.23c. Therefore, Fig. 2.23a and Fig. 2.23c are equivalent or interchangeable.

Why would anyone want to replace Fig. 2.23a by 2.23c? Suppose your shelves are full of AND gates and OR gates. If you have just run out of NOR gates and you are trying to build a NOR-NOR network like Fig. 2.23a, you can connect the OR-AND circuit of Fig. 2.23c because it produces the same output as the original circuit. In general, this idea applies to any circuit that you can rearrange with De Morgan's theorem. You can build whichever equivalent circuit is convenient.

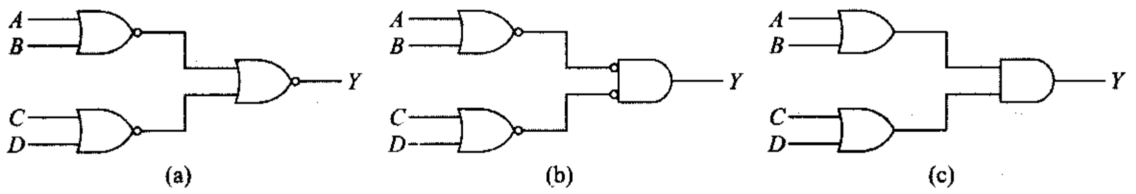


**Fig. 2.23** Equivalence among logic circuits: Example 2.9

**Example 2.10** What is the truth table for the NOR-NOR circuit of Fig. 2.23a?

*Solution* First, realize that the truth table of Fig. 2.23a is the same for 2.23b and 2.23c because all circuits are equivalent. Therefore, we can analyze whichever circuit we want to. Figure 2.23c is the easiest for most people to analyze, so let us use it in the following discussion.

Table 2.2 lists every possibility starting with all inputs low and progressing to all inputs high. Notice that the total number of entries equals $2^4$ or 16. By analyzing each input possibility, we can work out the corresponding output. For instance, when all inputs are low in Fig. 2.23c. both OR gates have low outputs, so the AND gate produces a low output. This is the first entry of Table 2.2. Proceeding like this, we can determine the output for the remaining possibilities and arrive at all the entries shown in Table 2.2.

**Example 2.11** Convert Table 2.2 into a timing diagram.

*Solution* In Table 2.2, input *D* changes states for each entry, input *C* changes states every other entry, input *B* every fourth entry, and input *A* every eighth entry. Figure 2.24 shows how to draw the truth table in the form of a timing diagram. First, notice that the transitions on input *D* are 1, 2, 3, and so on. Notice that input *D* changes states each transition, input *C* every other transition, input *B* every fourth transition, and input *A* every eighth transition. To agree with the truth table, output *Y* is low up to transition 5, high between 5 and 8, low between 8 and 9, and so forth.

**Table 2.2** **NOR-NOR Circuit**

| *A* | *B* | *C* | *D* | *Y* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

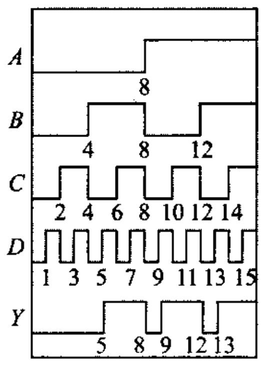


**Fig. 2.24** **Timing diagram**

**SELF-TEST**

7. Write the Boolean expression for a 2-input NOR gate.
8. Write De Morgan's first theorem.
9. What symbol is used inside the new IEEE rectangular box to define a NOR gate?

## NAND Gates

Originally, the logic circuit of Fig. 2.25a was called NOT-AND gate because the output is

$$Y = \overline{AB}$$

Read this as "Y equals NOT $A$ AND $B$" or "$Y$ equals the complement of $A$ AND $B$." Because the circuit is an AND gate followed by an inverter, the only way to get a low output is for both inputs to be high, as shown in the truth table of Table 2.3.

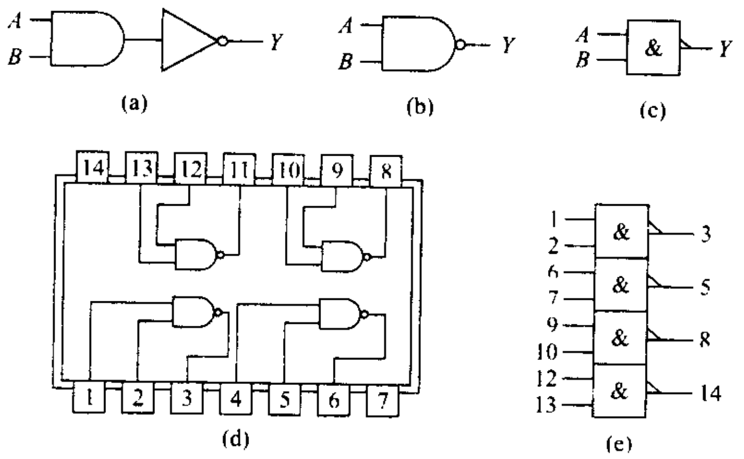


Fig. 2.25 NAND logic gate

## NAND-Gate Symbol

The logic circuit of Fig. 2.25a has become so popular that the abbreviated symbol of Fig. 2.25b is used for it. The bubble on the output reminds us of the inversion after the ANDing. Also, the words NOT-AND are contracted to NAND. Whenever you see this symbol, remember that the output is NOT the AND of the inputs. With a NAND gate, all inputs must be high to get a low output. If any input is low, the output is high.

Table 2.3 NAND Gate

| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 1 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Figure 2.25c shows the new IEEE rectangular symbol for the NAND gate. The small triangle on the output is equivalent to the bubble used on the standard symbol. The indicator "&" inside the box means "the output is high only when *all* inputs are high."

The 7400 is a quad 2-input NAND gate in a 14-pin DIP as illustrated in Fig. 2.25d. The new rectangular symbol for the 7402 is shown in Fig. 2.25e.

## Bubbled OR Gate

Figure 2.26a shows inverters on the input lines of an OR gate. The circuit is often drawn in the abbreviated form shown in Fig. 2.26b, where the bubbles represent inversion. We will refer to the abbreviated drawing of Fig. 2.2b as a *bubbled OR gate.* We have already analyzed this circuit in Example 2.3 and obtained its truth table in Fig. 2.9b. We see that output $Y$ and inputs $A$, $B$ are identical for bubbled OR gate and NAND gate. Therefore, these two circuits are equivalent and thus interchangeable. Given any logic circuit with NOR gates, we can replace it by bubbled AND gates and converse is also true.

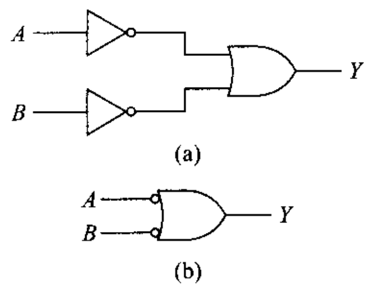


Fig. 2.26 (a) OR gate with inverted inputs, (b) Equivalent symbol

## De Morgan's Second Theorem

The Boolean equation for Fig. 2.24b is

$$Y = \overline{AB}$$

The Boolean equation for Fig. 2.25b is

$$Y = \overline{A} + \overline{B}$$

The first equation describes a NAND gate, and the second equation a bubbled OR gate. Since the outputs are equal for the same inputs, we can equate the right-hand members to get

$$\overline{AB} = \overline{A} + \overline{B} \qquad (2.2)$$

This identity is known as *De Morgan's second theorem*. It says the complement of a product equals the sum of the complements. This can also be proved by comparing the truth tables shown in Fig. 2.3(b) and NAND gate truth table of Table 2.2. A similar exercise that compares truth tables of three input NAND gate and three input bubbled OR gate show they are identical and we can write, $(A.B.C)' = A' + B' + C'$. Note that this equivalence can be extended to gates or circuits with any number of inputs.

## Universality of NAND Gate

Figure 2.27 shows how all other logic gates can be obtained from NAND gates and why it is called a universal logic gate. Figure 2.27a shows how we tie inputs of NAND gate together (as we had done in case of NOR gate) to get a NOT gate that has only one input. If input is 0, then both the inputs to NAND gate are 0. Following NAND gate truth table (Table 2.3) we see output now is 1. Similarly, if input is 1, both the inputs to NAND gate are 1 that gives output 0. Therefore output of circuit, shown in Fig. 2.27a is complement of its input and thus gives NOT operation.

Figure 2.27b shows how we get AND circuit using only NAND gates. The second NAND gate performs as a NOT gate and inverts the NAND logic of first NAND gate to AND logic.

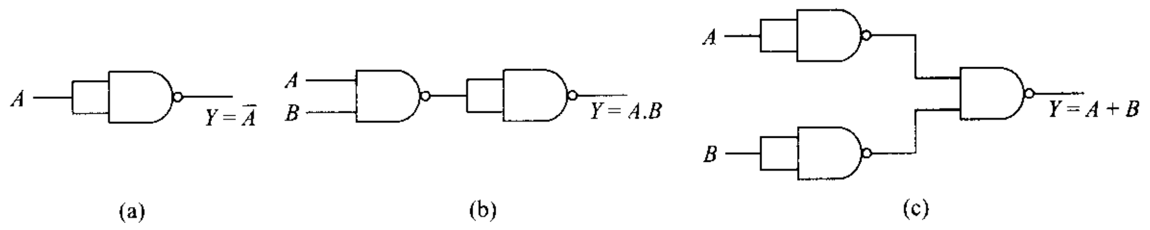


Fig. 2.27 Universality of NAND gate: (a) NOT from NAND, (b) AND from NAND, (c) OR from NAND

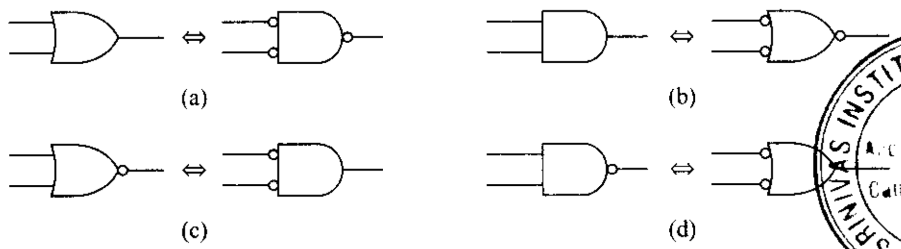


Fig. 2.28 Useful logic equivalences

To obtain OR logic using NAND gate we compare Fig. 2.27c circuit with Fig. 2.15a. The later gives NOR logic and has AND gate at output. The present circuit has NAND gate at output and thus inverts the output of previous circuit, from NOR to OR.

## TTL NAND Gates

The NAND gate is the backbone of the 7400 TTL series because most devices in this family are derived from it. Because of its central role in TTL technology, the NAND gate has become the least expensive and most widely used TTL gate. Furthermore, the NAND gate is available in more configurations than other gates, as shown in Table 2.4. Notice that the NAND gate is available as a 2-, 3-, 4-, or 8-input gate. The other gates have fewer configurations, with the OR gate available only in 2-input form.

Table 2.4 Standard TTL Gates

| *Type* | *Quad 2-Input* | *Triple 3-Input* | *Dual 4-Input* | *Single 8-Input* |
|---|---|---|---|---|
| NAND | 7400 | 7410 | 7420 | 7430 |
| NOR | 7402 | 7427 | 7425 | |
| AND | 7408 | 7411 | 7421 | |
| OR | 7432 | | | |

**Example 2.12** Prove that Fig. 2.29c is logically equivalent to Fig. 2.29a.

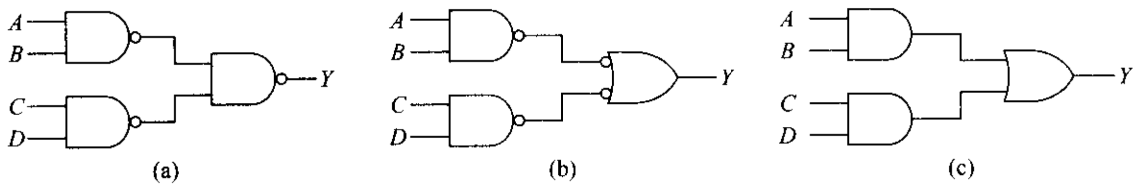


Fig. 2.29 Equivalence of logic gates: Example 2.12

***Solution*** De Morgan's second theorem says we can replace the final NAND gate of Fig. 2.29a by a bubbled OR gate to get the equivalent circuit of Fig. 2.29b. Each double inversion in Fig. 2.29b cancels out, leaving the simplified circuit of Fig. 2.29c. Therefore, Figs. 2.29a and 2.29c are equivalent.

Incidentally, most people find Fig. 2.29b easy to analyze because they learn to ignore the double inversions and see only the simplified AND-OR circuit of Fig. 2.29c. For this reason, if you build a NAND-NAND Network like Fig. 2.29a, you can draw it like Fig. 2.29b. Anyone who sees Fig. 2.29b on a schematic diagram will know it is two input NAND gates driving an output NAND gate. Furthermore, when troubleshooting the circuit, they can ignore the bubbles and visualize the easy-to-analyze AND-OR circuit of Fig. 2.29c.

**Example 2.13** What is the truth table for the NAND-NAND circuit of Fig. 2.29a?

*Solution* Let us analyze the equivalent circuit of Fig. 2.29c because it is simpler to work with. Table 2.5 lists every possibility starting with all inputs low and progressing to all inputs high. By analyzing each input possibility, we can determine the resulting output. For instance, when all inputs are low in Fig. 2.29c, both AND gates have low outputs, so the OR gate produces a low output. This is the first entry of Table 2.5. Proceeding like this, we can arrive at the output for the remaining possibilities of Table 2.5.

**Example 2.14** Show a timing diagram for the NAND-NAND circuit of Fig. 2.29a.

*Solution* All you have to do is convert the low-high states of Table 2.5 into low-high waveforms like Fig. 2.30. First, notice that the transitions on input *D* are numbered 1, 2, 3, and so on. Input *D* changes states each transition, input *C* every other transition, input *B* every fourth transition, and input *A* every eighth transition. To agree with the truth table, output *Y* is low up to transition 3, high between 3 and 4, low between 4 and 7, and so forth.

**Table 2.5 NAND-NAND Circuit**

| *A* | *B* | *C* | *D* | *Y* |
|---|---|---|---|---|
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 0 |
| 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 1 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 |

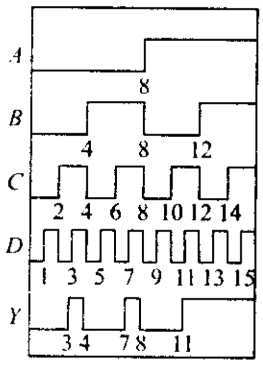


**Fig. 2.30 Timing diagram**

**SELF-TEST**

10. Write the Boolean expression for a 2 input NAND gate.
11. Write De Morgan's second theorem.
12. What symbol is used inside the IEEE rectangular box to define a NAND gate?

## 2.3 AND-OR-INVERT GATES

Figure 2.31a shows an AND-OR circuit. Figure 2.31b shows the De Morgan equivalent circuit, a NAND-NAND network. In either case, the Boolean equation is

$$Y = AB + CD$$

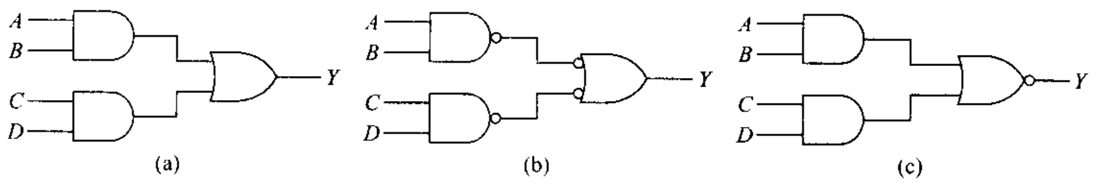


Fig. 2.31 (a) AND-OR circuit, (b) NAND-NAND circuit, (c) AND-OR-INVERT circuit

Since NAND gates are the preferred TTL gates, we would build the circuit of Fig. 2.31b. As you know, NAND-NAND circuits like this are important because with them you can build any desired logic circuit.

### TTL Devices

AND-OR circuits are not easily derived from the basic NAND-gate design. But it is easy to get an AND-OR-INVERT circuit as in Fig. 2.31c. A variety of circuits like this are available as TTL chips. Because of the inversion, the output has the equation shown below.

$$Y = \overline{AB + CD} \tag{2.3}$$

Table 2.6 lists the AND-OR-INVERT gates available in the 7400 series. In this table, *2-wide* means two AND gates across, *4-wide* means four AND gates across, and so on. For instance, the 7454 is a 2-input 4-wide AND-OR-INVERT gate as in Fig. 2.32a; each AND gate has two inputs (2-input), and there are four AND gates (4-wide). Figure 2.32b shows the 7464; it is a 2-2-3-4-input 4-wide AND-OR-INVERT gate.

Table 2.6 AND-OR-INVERT Gates

| *Device* | *Description* |
|---|---|
| 7451 | Dual 2-input 2-wide |
| 7454 | 2-input 4-wide |
| 7459 | Dual 2-3-input 2-wide |
| 7464 | 2-2-3-4-input 4-wide |

Connecting the output of a 2-input 2-wide AND-OR-INVERT gate to an inverter will give us the same output as an AND-OR circuit.

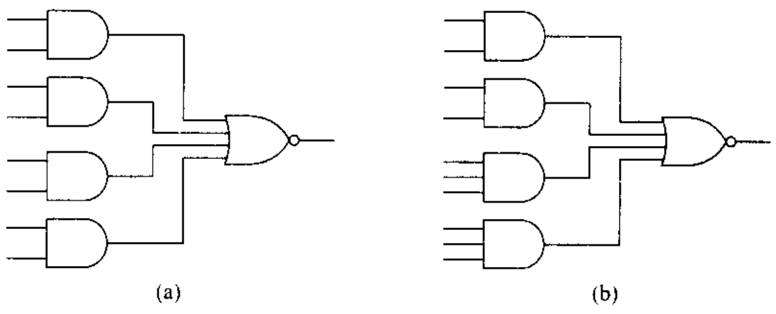


Fig. 2.32 Examples of AND-OR circuits

## Expandable AND-OR-INVERT Gates

The widest AND-OR-INVERT gate available in the 7400 series is 4-wide. What do we do when we need a 6- or 8-wide circuit? One solution is to use an *expandable* AND-OR-INVERT gate.

Figure 2.33 shows the logic symbol for an expandable AND-OR-INVERT gate. There are two additional inputs, labeled *bubble* and *arrow*. Table 2.7 lists the expandable AND-OR-INVERT gates in the 7400 series.

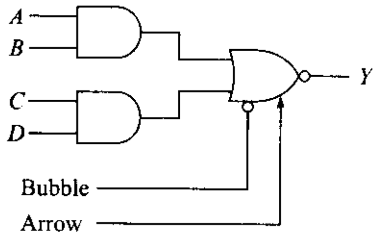


**Fig. 2.33** **Expandable AND-OR-INVERT gate**

**Table 2.7** **Expandable AND-OR-INVERT Gates**

| *Device* | *Description* |
|---|---|
| 7450 | Dual 2-input 2-wide |
| 7453 | 2-input 4-wide |
| 7455 | 4-input 2-wide |

## Expanders

What do we connect to the arrow and bubble inputs of an expandable gate? We connect the output of an *expander* as in Fig. 2.34a. Connect bubble to bubble and arrow to arrow.

Visualize the outputs of Fig. 2.34a connected to the arrow and bubble inputs of Fig. 2.33. Figure 2.34b shows the logic circuit. This means that the expander outputs are being ORed with the signals of the AND-OR-INVERT gate. In other words, Fig. 2.34b is equivalent to the AND-OR-INVERT circuit of Fig. 2.34c.

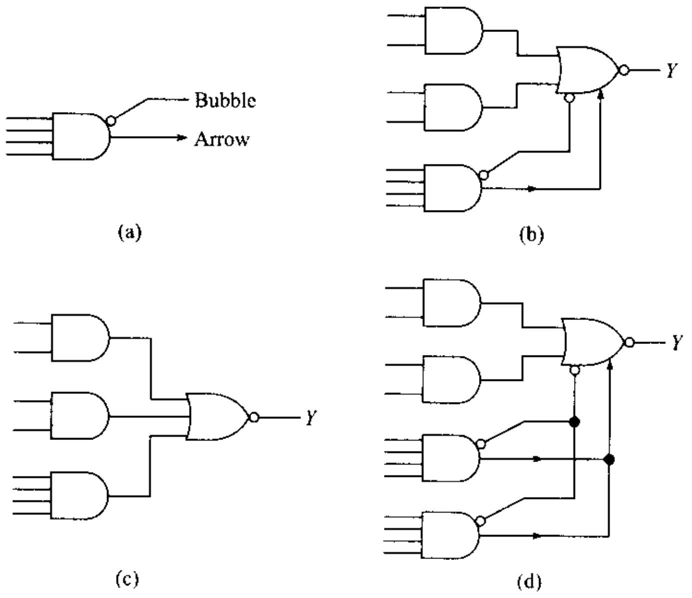


**Fig. 2.34** **(a) Expander, (b) Expander driving expandable AND-OR-INVERT gate, (c) AND-OR-INVERT circuit, (d) Expandable AND-OR-INVERT with two expanders**

We can connect more expanders. Figure 2.34d shows two expanders driving the expandable gate. Now we have a 2-2-4-4-input 4-wide AND-OR-INVERT circuit.

The 7460 is a dual 4-input expander. The 7450, a dual expandable AND-OR-INVERT gate, is designed for use with up to four 7460 expanders. This means that we can add two more expanders in Fig. 2-34d to get a 2-2-4-4-4-4-input 6-wide AND-OR-INVERT circuit.

**SELF-TEST**

13. **When we speak of an AND-OR-INVERT gate, what is the meaning of 2-wide?**
14. **What is the purpose of using an *expander* with an AND-OR-INVERT gate?**

## 2.4 POSITIVE AND NEGATIVE LOGIC

Up to now, we have used a binary 0 for low voltage and a binary 1 for high voltage. This is called *positive logic*. People are comfortable with positive logic because it feels right. But there is another code known as *negative logic* where binary 0 stands for high voltage and binary 1 for low voltage. Even though it seems unnatural, negative logic has many uses. The following discussion introduces some of the terminology and concepts for both types of logic.

### Positive and Negative Gates

An OR gate in a positive logic system becomes an AND gate in a negative logic system. Why? Look at the gate of Fig. 2.35. We have been calling it an OR gate. This is correct, provided we are using positive logic. Table 2.8 shows the *truth* for the gate of Fig. 2.35, no matter what you call it. That is, if either input is high in Fig. 2.35, the output is high.

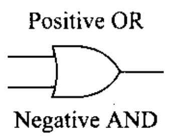


**Fig. 2.35** **Meaning of symbol depends on whether you use positive or negative logic**

**Table 2.8**

| *A* | *B* | *Y* |
|---|---|---|
| Low | Low | Low |
| Low | High | High |
| High | Low | High |
| High | High | High |

In a positive logic system, binary 0 stands for low and binary 1 for high. So, we can convert Table 2.8 to Table 2.9. Note that *Y* is a 1 if either *A* or *B* is 1. This sounds like an OR gate. And it is, because we are using positive logic. To avoid ambiguity, we can call Fig. 2.35 a positive OR gate because it performs the OR function with positive logic. (Some data sheets describe gates as positive OR gate, positive AND gate, etc.)

In a negative logic system, binary 1 stands for low and binary 0 for high. With this code, we can convert Table 2.8 to Table 2.10. Now, watch what happens. The output *Y* is a 1 only when both *A* and *B* are 1. This sounds like an AND gate! And it is, because we are now using negative logic. In other words, gates are defined by the way they process the binary 0s and 1s. If you use binary 1 for low voltage and binary 0 for high voltage, then you have to refer to Fig. 2.35 as a negative AND gate.

As you see, the gate of Fig. 2.35 always produces a high output if either input is high. But what you call it depends on whether you see positive or negative logic. Use whichever name applies. With positive logic, call it a positive OR gate. With negative logic, call it a negative AND gate.

Table 2.9

| $A$ | $B$ | $Y$ |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

Table 2.10

| $A$ | $B$ | $Y$ |
|---|---|---|
| 1 | 1 | 1 |
| 1 | 0 | 0 |
| 0 | 1 | 0 |
| 0 | 0 | 0 |

In a similar way, we can show the truth table of other gates with positive or negative logic. By analyzing the inputs and outputs in terms of 0s and 1s, you find these equivalences between the positive and negative logic:

| | |
|---|---|
| Positive OR | ↔ negative AND |
| Positive AND | ↔ negative OR |
| Positive NOR | ↔ negative NAND |
| Positive NAND | ↔ negative NOR |

Table 2.11 summarizes these gates and their definitions in terms of voltage levels. These definitions are always valid. If you get confused from time to time, refer to Table 2.11 to get back to the ultimate meaning of the basic gates.

Table 2.11 **Voltage Definitions of Basic Gates**

| *Gate* | *Definition* |
|---|---|
| Positive OR/negative AND | Output is high if any input is high. |
| Positive AND/negative OR | Output is high when all inputs are high. |
| Positive NOR/negative NAND | Output is low if any input is high. |
| Positive NAND/negative NOR | Output is low when all inputs are high. |

## Assertion-Level Logic

Why do we even bother with negative logic? The reason is related to the concept of *active-low signals*. For instance, the 74150 multiplexer has an active-low input strobe; this input turns on the chip only when it is low (negative true). This is an active-low signal; it causes something to happen when it is low, rather than high. As another example, the 74154 decoder has 16 output lines; the decoded output signal is low (negative true). In other words, all output lines have a high voltage, except the decoded output line. Besides TTL devices, microprocessor chips like the 8085 have a lot of active-low input and output signals.

Many designers draw their logic circuits with bubbles on all pins with active-low signals and omit bubbles on all pins with active-high signals. This use of bubbles with active-low signals is called *assertion-level logic*. It means that you draw chips with the kind of input that causes something to happen, or with the kind of output that indicates something has happened. If a low input signal turns on a chip, you show a bubble on that input. If a low output is a sign of chip action, you draw a bubble on that output. Once you get used to assertion-level logic, you may prefer drawing logic circuits this way.

One final point. Sometimes you hear expressions such as "The inputs are asserted" or "What happens when the inputs are asserted?" An input is *asserted* when it is active. This means it may be low or high, depending on whether it is an active-low or active-high input. For instance, given a positive AND gate, all inputs must be asserted (high) to get a high output. As another example, the STROBE input of a TTL multiplexer must be asserted (low) to turn on the multiplexer. In short, you can equate the word *assert* with *activate*. You assert, or activate, the inputs of a gate or device to get something to happen.

**Points to Remember**

Here are some ideas that you should try to remember:

1. Positive true always represents a high voltage, and negative true always represents a low voltage.
2. If possible, draw basic gates with bubbles on active-low signal lines.
3. When a signal is active-low, use an overbar as a reminder that the signal voltage is negative true when the underlying statement is true.

**Example 2.15**

(a) The number stored in a register may be zero (all bits low). Show how to detect this condition.
(b) What change in (a) will detect presence of the word 10110101 in the 8-bit register?

*Solution*

(a) Figure 2.36 shows a design using assertion-level logic. The bits go to a bubbled AND gate (the same as positive NOR gate). When all the bits are low, output ZERO is high. Because of the inverter, the final output $\overline{\text{ZERO}}$ is active-low. Therefore, when the sum is zero, $\overline{\text{ZERO}}$ is negative true.
(b) Some of the bubbles at the input of the bubbled AND gate need to be removed. These are for locations in the code word where '1' is present, specifically $S_7$, $S_5$, $S_4$, $S_2$ and $S_0$.

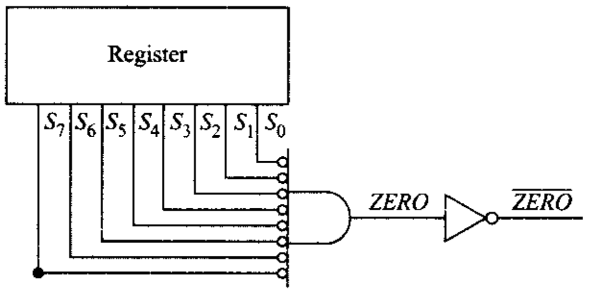


**Fig. 2.36** **Assertion-level logic diagram showing the detection of zero and minus accumulator contents**

**SELF-TEST**

15. What is negative logic?
16. What is meant by *assertion-level logic*?

## 2.5 INTRODUCTION TO HDL

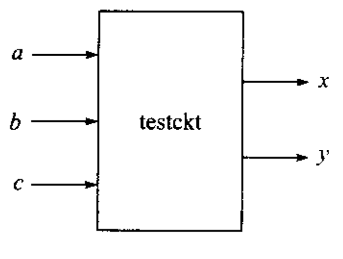
In this section, we introduce an interesting development in the field of hardware design. This is textual description of a digital circuit. Though we have already described hardware, can there be a language which is more crisp and more importantly, machine-readable? The advantage of course, is to be able to (i) describe a large complex design requiring hundreds of logic gates in a convenient manner, in a smaller space, (ii) use software test-bench to detect functional error, if any, and correct it (called *simulation*) and finally, (iii) get hardware implementation details (called *synthesis*). Hardware Description Language, more popular with its acronym HDL is an answer for that.

Currently, there are two widely used HDLs—Verilog and VHDL (Very high speed integrated circuit Hardware Description Language). Verilog is considered simpler of the two and is more popular. However, both share lot of common features and it is not too difficult to switch from one to the other. In this book, we'll deal with Verilog and shall discuss it over a span of number of chapters by introducing features relevant to that chapter. We expect by the time you finish Chapter 11, you'll have reasonable knowledge about HDL to deal with any digital logic design problem. We discuss target hardware devices on which HDL code can be directly exported in Section 13.6 of Chapter 13.

## Verilog HDL

Verilog as a hardware description language has a small history. Introduced in 1980, primarily as a simulation and verification tool by Gateway Design Automation, it was later acquired by Cadence Data Systems. Put to public domain in 1990, it gained popularity and is now controlled by a group of companies and universities, called Open Verilog International. The reader with an exposure to any programming language like C will find it relatively easier to learn Verilog or any HDL.

**Describing Input/Output** In any digital circuit, we find there are a set of inputs and a set of outputs. Often termed as *ports*, the relationship between these input and outputs are explained within the digital circuit. To design any circuit that has say, three inputs $a$, $b$, $c$ and two outputs say, $x$, $y$ as shown in Fig. 2.37 the corresponding Verilog code can be written as shown next.



```
module testckt(x,y,a,b,c); //module name with port list

input  a,b,c; //defines input ports
output x,y; //defines output ports

//module body begins next describing logic relation
.
.
//module body ends
endmodule
```

**Fig. 2.37 Input/output definition in Verilog HDL for logic circuit described within black-box testckt**

Note that, **module** and **endmodule** written in bold are keywords for Verilog. A module describes a design entity with a name or identifier selected by user (here, testckt) followed by input output port list. This entity if used by another then arguments (i.e. ports ) are to be passed in the same order as it appears here. The symbol '//' is used to put comments and improve readability for a human but not used by the machine, i.e. compiler. The module body describes the logic within the black box which acts on the inputs $a$, $b$, $c$ and generates output $x$, $y$. Observe, where semicolon ';' is used and where not to end a statement, e.g. *endmodule* in above code does not end with semicolon.

**Writing Module Body** There are three different models of writing module body in Verilog HDL. Each one has its own advantage and suited for certain kind of design. We start with structural model by example of two-input OR gate described in Fig. 2.4a.

```
module or_gate(A,B,Y);
input A,B;   // defines two input port
output Y;    // defines one output port
or g1(Y,A,B); /*Gate declaration with predefined keyword or representing
                logic OR, g1 is optional user defined gate identifier */
endmodule
```

Verilog supports predefined gate level primitives such as **and, or, not, nand, nor, xor, xnor** etc. The syntax followed above can be extended to other gates and for 4 input OR gate it is as given next,

**or** (output, input 1, input 2, input 3, input 4)

For NOT gate, **not** (output, input)

Note that, Verilog can take up to 12 inputs for logic gates. Comments when extends to next line is written within /* ..... */. Identifiers in Verilog are case sensitive, begin with a letter or underscore and can be of any length.

Let us now look at description of a logic circuit shown in Fig. 2.17a that has 4 inputs and 1 output. The inputs are fed to two 2-input AND gate. AND gate outputs are fed to a 2-input OR gate to generate final output. The verilog code for this is given below. Note that, we define two intermediate variables and_op1 and and_op2 representing two AND gate outputs through keyword **wire**. Wire represents a physical wire in a circuit.

```
module fig2_24a(A,B,C,D,Y);
  input A,B,C,D;
  output Y;
  wire and_op1, and_op2;  // internal connections
  and g1(and_op1,A,B); // g1 represents upper AND gate
  and g2(and_op2,C,D); // g2 represents lower AND gate
  or g3(Y,and_op1,and_op2); // g3 represents the OR gate
endmodule
```

One can see that structural model tries to replicate graphical layout design of a logic circuit. It does not matter if **or** statement in above example is written before **and** statements. This is as if one draws or connects the OR gate first on a design board and then the AND gates.

**Example 2.16** Consider, the black box testckt of Fig. 2.38 has following logic circuit in it. Give Verilog structural code for the same.

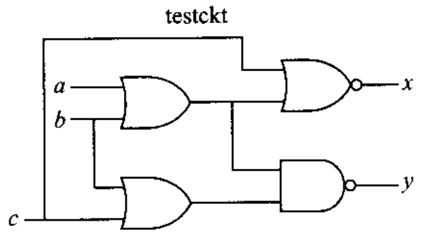


**Fig. 2.38** **Logic circuit for Example 2.16**

*Solution* The code from the above discussion can be written as follows.

```
  module testckt(a,b,c,x,y);
   input a,b,c;
   output x,y;
   wire or_op1, or_op2; /* internal connections, outputs of upper and
      lower OR gates respectively */
   or g1(or_op1,a,b); // g1 represents upper OR gate
   or g2(or_op2,b,c); // g2 represents lower OR gate
   nor g3(x,c,or_op1); // g3 represents the NOR gate
   nand g4(y,or_op1,or_op2); // g4 represents the NAND gate
  endmodule
```

**Preparation of Test Bench** We shall discuss data flow model and behavioral model of Verilog VHDL in subsequent chapters. But, before we wind up this chapter let us see how to prepare a test bench in Verilog to simulate a digital circuit. For those of you with no programming background, this may appear little difficult. We could have postponed this discussion to a later chapter, but this gives you a feel of how simulation works or how a circuit you design can be tested. More clarity is assured as you go through discussions of subsequent chapters.

We take up the example of simulating a simple OR gate (Fig. 2.4a) for which Verilog code is already described. The test bench, creates an input in the form of a timing waveform and passes this to OR gate module through a function or procedural call (passing arguments in proper order). To generate timing waveform we use time delay available in Verilog in the form of *#n* where *n* denotes a number in decimal that gives delay in nanosecond. Input values to a variable can be provided through syntax *m'tn* where *m* represents number of digits, *t* represents type of number and *n* represents value to be provided.

The test bench used here generates all possible combinations of two inputs AB as 00,01,10 and 11 but at an interval of 20 ns. Note that, we have provided a 20 ns gate delay with **or** statement by #(20). All practical logic circuit comes with finite gate delay, i.e. output changes according to input after certain time. To change the gate delay to 10 ns we should write #(10) in **or** statement. The keyword **reg** is used to hold value of a data object in a procedural assignment. The keyword **initial** ensures sequential execution of codes following it, but once. We'll learn another keyword **always** in later chapter, which too is used for sequential execution but for infinite time.

```
 module testor; //Simulation module given a name testor
 reg A,B;        //Storage of data for passing it to module or_gate
 wire x;
 or_gate org(A,B,x); //circuit is instantiated with the name, or_gate
 initial     // Starts simulation
   begin   /* Input is generated to test the circuit through following
     statements, simulation begins*/
 A=1'b0;B=1'b0; /* 1'b0 signifies on binary digit with a value 0, AB is
assigned 00*/
     #20                  // Delay of 20 ns
     A=1'b0;B=1'b1;  //After 20ns AB=01
```

```
    #20          // Another Delay of 20 ns
    A=1'b1;B=1'b0; //After 40ns from start point  AB=10
    #20
    A=1'b1;B=1'b1; //After 60ns from start point  AB=11
    #20 $finish;// the simulation terminates after 80 ns
  end
endmodule

module or_gate(A,B,x); // OR gate used as a procedure in simulation
    input A,B;    // defines two input port
    output x;     // defines one output port
    or #(20) g1(x,A,B);  /*Gate declaration with a gate delay of 20ns,
     output is effected after 20 ns*/
endmodule
```

Execution of above Verilog code generates following timing diagram. One can see that input AB, given by testor.A and testor.B (testor is module name of the test bench) is taking value 00,01,10,11 as expected and retain them for 20 ns. Output of OR gate, testor.x changes according to input but after a delay of 20 ns. For first 20 ns, OR gate output is unknown as it needs 20ns (gate delay) to respond to first appearance of input logic at A,B. Note that Verilog, in general offers four logic values in simulation 0,1, unknown (or **x**) and high impedance (or **z**). Unknown value is exhibited when input is ambiguous and high impedance is shown when a wire by mistake is left unconnected or the circuit is following tri-state logic (Chapter 14, Section 6).

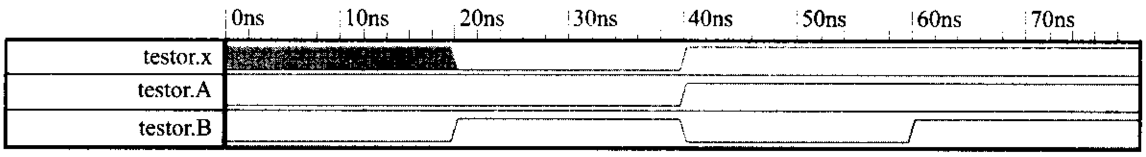


**Fig. 2.39** **Verilog simulation of 2 input OR gate with 20ns gate delay**

**Example 2.17** Write the statements between **begin** and **end** of a test bench for circuit described in Example 2.16 with 50 ns holding time of each input combination.

*Solution* Since the circuit has three inputs we need $2^3 = 8$ different combinations of inputs. Thus the statements would look like as follows:

```
begin   // Input is generated to test the circuit through following
           statements, simulation begins
    a=1'b0;b=1'b0;c=1'b0; //  ABC is assigned 000
    #50                   // Delay of 50 ns
    a=1'b0;b=1'b0;c=1'b1; //  ABC is assigned 001
    #50                   // Delay of 50 ns
    a=1'b0;b=1'b1;c=1'b0; //  ABC is assigned 010
    #50                         // Delay of 50 ns
    a=1'b0;b=1'b1;c=1'b1;  //  ABC is assigned 011
    #50                         // Delay of 50 ns
    a=1'b1;b=1'b0;c=1'b0;  //  ABC is assigned 100
    #50                         // Delay of 50 ns
    a=1'b1;b=1'b0;c=1'b1;  //  ABC is assigned 101
    #50                         // Delay of 50 ns
    a=1'b1;b=1'b1;c=1'b0;  //  ABC is assigned 110
    #50                         // Delay of 50 ns
    a=1'b1;b=1'b1;c=1'b1;  //  ABC is assigned 111
    #50 $finish;// the simulation terminates after 400 ns
end
```

**Example 2.18** The following timing diagram is generated by simulation of Verilog code for 2-input, 1-output device where A and B are input and x is output. Can you (i) estimate the gate delay and (ii) identify the logic?

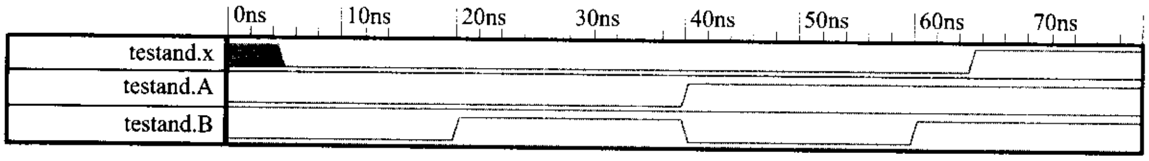


**Fig. 2.40** **Verilog simulation for Example 2.18**

*Solution*

(i) The unknown value of the output is approximately half of 10 ns time scale. Hence, gate delay is 5ns.
(ii) Output goes HIGH when both the inputs go high after a delay of 5 ns. Hence, the logic underlying is AND.

## PROBLEM SOLVING WITH MULTIPLE METHODS

**Problem** Realize $Y = AB + \overline{C}$ using only one type of gate.

*Solution* The function to be realized involves AND, NOT and OR operations. We can realize this expression using universal logic gate.

**In Method-1,** we realize it using NOR gate. We realize individual logic operations like AND, NOT and OR as depicted in Fig. 2.21. The solution is given in Fig. 2.41.

**In Method-2,** we realize it using NAND gate. We realize individual logic operations like AND, NOT and OR as depicted in Fig. 2.27. The solution is given in Fig. 2.42.

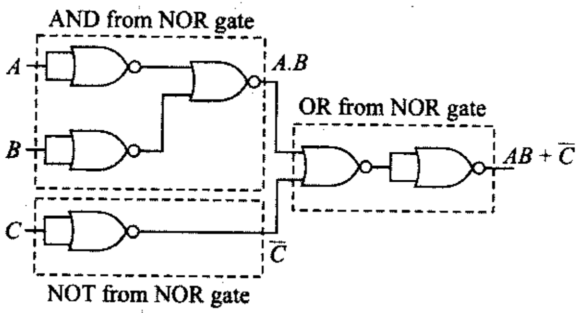


**Fig. 2.41** **Realization of $Y = AB + \overline{C}$ using only NOR gate**

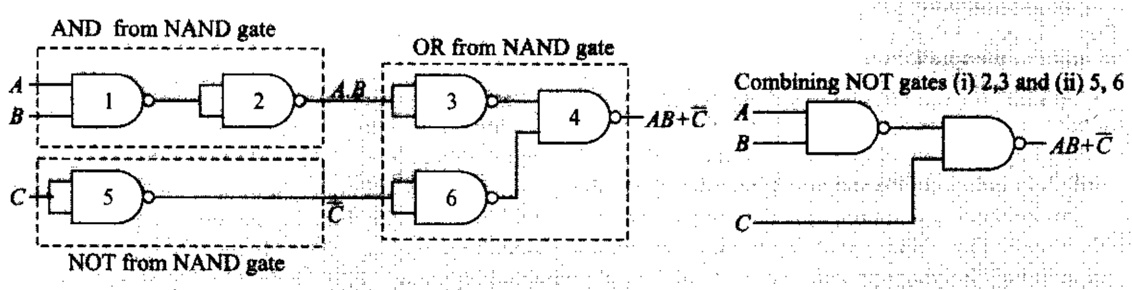


Fig. 2.42 **Realization of $Y = AB + \bar{C}$ using only NAND gate**

Note that the final logic circuit achieved involves only 2 NAND gates as two NOT gates can easily be replaced by directly drawing connection from uncomplemented input.

**In Method-3,** we again get a solution using NOR gate but with a different approach where we use De Morgan's first (Eq. 2.1) and second (Eq. 2.2) theorem. The objective is to have only NOR relation throughout.

Given,

$$
\begin{aligned}
Y &= AB + \bar{C} \\
&= \overline{\overline{AB + \bar{C}}} && \text{double complement} \\
&= \overline{\overline{AB}.C} && \text{from Eq. 2.1} \\
&= \overline{(\bar{A} + \bar{B}).C} && \text{from Eq. 2.2} \\
&= \overline{(\bar{A} + \bar{B})} + \bar{C} && \text{from Eq. 2.2} \\
&= \overline{\overline{\overline{(\bar{A} + \bar{B})} + \bar{C}}} && \text{double complement}
\end{aligned}
$$

Thus, two NOR operations $(\bar{A} + \bar{B})'$ and $((\bar{A} + \bar{B})' + \bar{C})'$ [' represents NOT operation such that $\bar{X} = X'$] require two NOR gates, and four inversions $\bar{A}$, $\bar{B}$, $\bar{C}$ and $(((\bar{A} + \bar{B})' + \bar{C})')'$ require four NOR gates totaling six NOR gates for realization. You will find that the solution is similar to what is achieved by Method-1.

**In Method-4,** similar to Method-3 we use De Morgan's theorems to get a solution but only with NAND gates. The objective is to have only NAND relation throughout. We need to go only up to step two of Method-3 analysis to get an all NAND realization.

Given,

$$
\begin{aligned}
Y &= AB + \bar{C} \\
&= \overline{\overline{AB + \bar{C}}} && \text{double complement} \\
&= \overline{\overline{AB}.C} && \text{from Eq. 2.1}
\end{aligned}
$$

Thus, we need two NAND gates—one for $(A.B)'$ and another for $((A.B)'.C)'$ realization. We find that the final logic circuit of Method-2 is similar to what is obtained here.

## SUMMARY

Almost all digital circuits are designed for two-state operation, which means the signal voltages are either at a low level or a high level. Because they duplicate mental processes, digital circuits are often called logic circuits. A gate is a digital circuit with 1 or more inputs, but only 1 output. The output is high only for certain combinations of the input signals.

An inverter is one type of logic circuit, it produces an output that is the complement of the input. An OR gate has 2 or more input signals; it produces a high output if any input is high. An AND gate has 2 or more input signals; it produces a high output only when all inputs are high. Truth tables often use binary 0 for the low state and binary 1 for the high state. The number of entries in a truth table equals $2^n$, where $n$ is the number of input signals.

The overbar is the algebraic symbol for the NOT operation, the plus sign is the symbol for the OR operation, and the times sign is the symbol for the AND operation. Since the Boolean operators are codes for the OR gate, AND gate, and inverter, we can use Boolean algebra to analyze digital circuits. An AND-OR circuit always produces a sum-of-products equation, while the OR-AND circuit results in a product-of-sums equation.

The NOR gate is equivalent to an OR gate followed by an inverter. De Morgan's first theorem tells us that a NOR gate is equivalent to a bubbled AND gate. Because of De Morgan's first theorem, a NOR-NOR circuit is equivalent to an OR-AND circuit.

The NAND gate represents an AND gate followed by an inverter. De Morgan's second theorem says the NAND gate is equivalent to a bubbled OR gate. Furthermore, a NAND-NAND circuit is equivalent to an AND-OR circuit. The NAND gate is the backbone of the 7400 TTL series because most devices in this family are derived from the NAND-gate design. The NAND gate is a universal gate since any logic circuit can be built with NAND gates only.

With positive logic, binary 1 represents high voltage and binary 0 represents low voltage. Also, positive true stands for high voltage and positive false for low voltage. With negative logic, binary 1 stands for low voltage and binary 0 for high voltage. In this system, negative true is equivalent to low voltage and negative false to high voltage.

With assertion-level logic, we draw gates and other devices with bubbled pins for active-low signals. Also, signal voltages are labeled with abbreviations of statements that describe circuit behavior. An overbar is used on a label whenever the signal is active-low.

Figure 2.43 shows three sets of equivalent gates. Changing from one to the other is accomplished by adding or deleting bubbles, and changing AND to OR or OR to AND. The NOR gate and NAND gate equivalents illustrate De Morgan's first and second theorems.

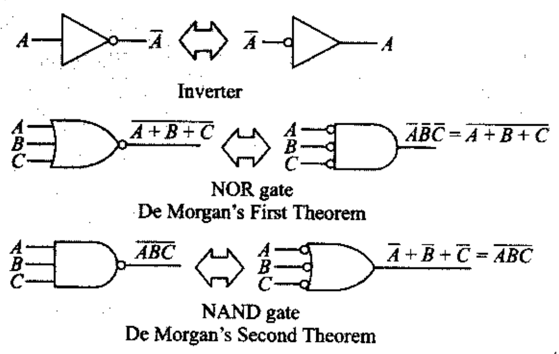

A
$\overline{A}$
$\overline{A}$
A
Inverter
A
B
C
$\overline{A+B+C}$
A
B
C
$\overline{A}\overline{B}\overline{C} = \overline{A+B+C}$
NOR gate
De Morgan's First Theorem
A
B
C
$\overline{ABC}$
A
B
C
$\overline{A}+\overline{B}+\overline{C} = \overline{ABC}$
NAND gate
De Morgan's Second Theorem

Fig. 2.43

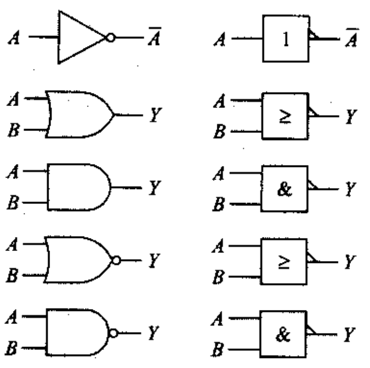


Fig. 2.44

Figure 2.44 shows the additional logic symbols for five basic gates along with the corresponding IEEE rectangular symbols.

## GLOSSARY

- ***active-low*** Active-low refers to the concept in which a signal must be low to cause something to happen or to indicate that something has happened.
- ***AND gate*** A gate with 2 or more inputs. The output is high only when all inputs are high.
- ***assert*** To activate. If an input line has a bubble on it, you assert the input by making it low. If there is no bubble, you assert the input by making it high.
- ***De Morgan's first theorem*** In words, the complement of a logical sum equals the logical product of the complements. In terms of circuits, a NOR gate equals a bubbled AND gate.
- ***De Morgan's second theorem*** In words, the complement of a logical product equals the logical sum of the complements. In terms of circuits, a NAND gate is equivalent to a bubbled OR gate.
- ***gate*** A digital circuit with one or more input voltages but only one output voltage.
- ***inverter*** A gate with only one input and a complemented output.
- ***logic circuit*** A digital circuit, a switching circuit, or any kind of two-state circuit that duplicates mental processes.
- ***negative true*** A signal is negative true when the voltage is low.
- ***OR gate*** A gate with two or more inputs. The output is high when any input is high.
- ***positive true*** A signal is positive true when the voltage is high.
- ***product-of-sums equation*** A Boolean equation that is the logical product of logical sums. This type of equation applies to an OR-AND circuit.
- ***sum-of-products equation*** A Boolean equation that is the logical sum of logical products. This type of equation applies to an AND-OR circuit.
- ***timing diagram*** A picture that shows the input-output waveforms of a logic circuit.
- ***truth table*** A table that shows all of the input-output possibilities of a logic circuit.
- ***two-state operation*** The use of only two points on the load line of a device, resulting in all voltages being either low or high.

## PROBLEMS

### Section 2.1

2.1 What is the output in Fig. 2.45a if the input is low? The output if the input is high?

2.2 The input is low in Fig. 2.45b. Is the output low or high? Is the circuit equivalent to an inverter? If you cascade an odd number of inverters, what kind of gate is the overall circuit equivalent to?

2.3 Construct the truth table for Fig. 2.46a. After you are finished, discuss the relation between

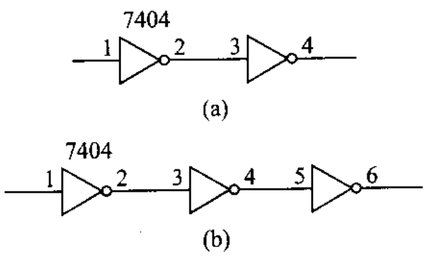


Fig. 2.45

the circuit of Fig. 2.46a and a 3-input OR gate.

2.4 Construct the truth table for Fig. 2.46b.

2.5 Construct the truth table for Fig. 2.46c.

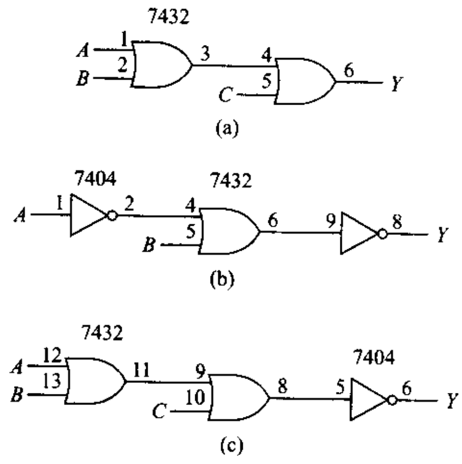


Fig. 2.46

2.6 The circuit of Fig. 2.46b has trouble. Figure 2.47 shows its timing diagram. Which of the following is the trouble:

a. Input inverter acts like OR gate.

b. Pin 6 is shorted to ground.

c. AND gate is used instead of OR gate.

d. Output inverter is faulty.

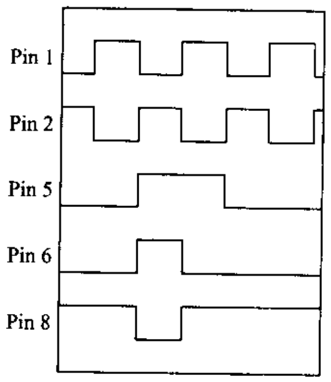


Fig. 2.47

2.7 Construct the truth table for Fig. 2.48a. Then discuss the relation between the circuit of Fig. 2.48a and a 3-input AND gate.

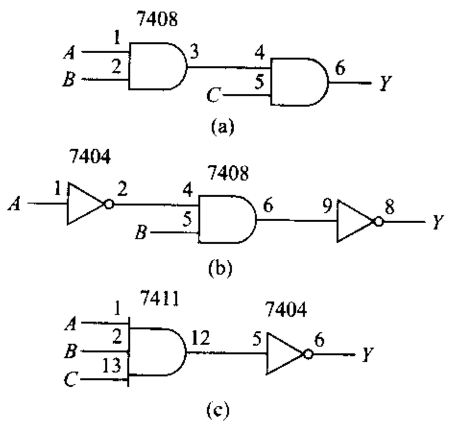


Fig. 2.48

2.8 Construct the truth table of Fig. 2.48b.

2.9 Construct the truth table for Fig. 2.48c.

2.10 Assume the circuit of Fig. 2.48b has trouble. If Fig. 2.49 is the timing diagram, which of the following is the trouble:

a. Input inverter is shorted.

b. OR gate is used instead of AND gate

c. Pin 6 is shorted to ground.

d. Pin 8 is shorted to +5 V.

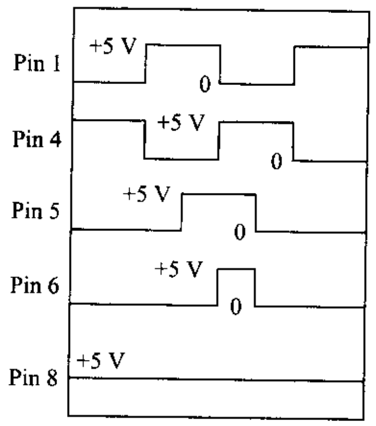


Fig. 2.49

2.11 What is the Boolean equation for the output of Fig. 2.46a?
For Fig. 2.46b? For Fig. 2.46c?

2.12 Draw the logic circuit whose Boolean equation is

$$Y = \overline{A + B} + \overline{C}$$

2.13 Use the 7404 and the 7432 with pin numbers.
What is the Boolean equation for the output of Fig. 2.46a?

2.14 For Fig. 2.46b? For Fig. 2.46c?
Draw the logic circuit described by

$$Y = (\overline{ABC})\overline{D}$$

2.15 Use a 7404, 7408, and 7411 with pin numbers.
Draw the logic circuit given by this Boolean equation:

$$Y = \overline{A}BC + A\overline{B}C + AB\overline{C} + \overline{A}B\overline{C}$$

Use the following devices with pin numbers: 7404, 7411, and 7432.

## Section 2.2

2.16 Construct the truth table for the 3-input NOR gate of Fig. 2.50a.

2.17 Construct the truth table for the 4-input NOR gate of Fig. 2.50b.

2.18 Show an equivalent NOR-NOR circuit for Fig. 2.50c. Use the 7402 and the 7427 with pin numbers.

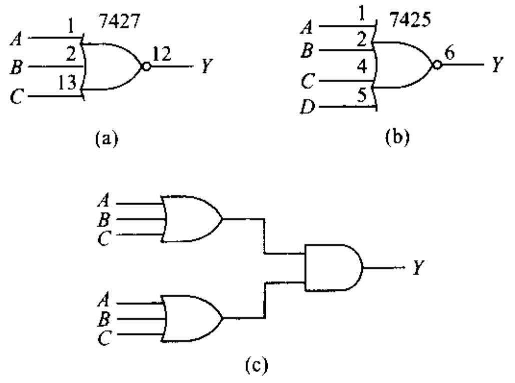


Fig. 2.50

2.19 The circuit of Fig. 2.50c has trouble. If output $Y$ is stuck in the high state, which of the following is the trouble:

a. Either input pin of the AND gate is shorted to ground.
b. Any input of either OR gate is shorted to ground.
c. Any input of either OR gate is shorted to a high voltage.
d. AND gate is defective.

2.20 Construct the truth table for the 4-input NAND gate of Fig. 2.51a.

2.21 The inputs are $A_0, A_1, A_2, \ldots, A_7$ in Fig. 2.51b. What is the Boolean equation for the input of the NAND gate?

2.22 Draw an equivalent NAND-NAND circuit for Fig. 2.51c. Use the 7420 and include pin numbers.

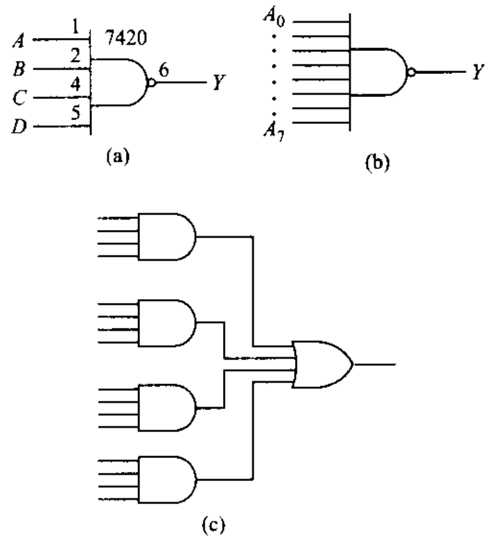


Fig. 2.51

2.23 Suppose the final output of Fig. 2.51c is stuck in the high state. Which of the following is the trouble?

a. Any input to the OR gate is shorted to a high voltage.
b. Any AND-gate input is shorted to ground.

c. Any AND-gate input is shorted to a high voltage.

d. One of the AND gates is defective because its output is always low.

## Section 2.3

2.24 What is the output in Fig. 2.31c for these inputs?

a. $ABCD = 0000$ b. $ABCD = 0101$

c. $ABCD = 1100$ d. $ABCD = 1111$

2.25 Is the output $Y$ of Fig. 2.52 low or high for these conditions?

a. Both switches open, $A$ is low.

b. Both switches closed, $A$ is high.

c. Left switch open, right switch closed, $A$ is low.

d. Left switch closed, right switch open, $A$ is high.

2.26 If all inputs are low in Fig. 2.34b, what is the output? If all inputs are high, what is the output?

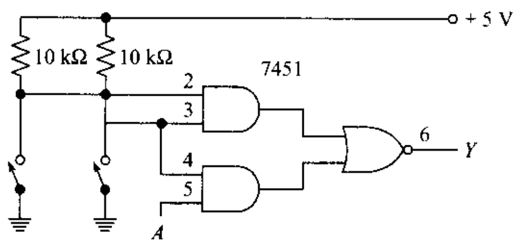


Fig. 2.52

2.27 What is the value of $Y$ in Fig. 2.53 for each of these?

a. $ABCD = 0000$ b. $ABCD = 0101$

c. $ABCD = 1000$ d. $ABCD = 1111$

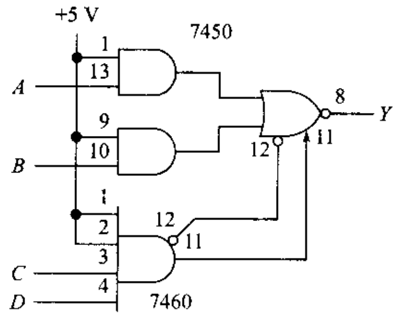


Fig. 2.53

## Section 2.4

2.28 In Fig. 2.54, is each of the following an active-low or an active-high?

a. Pin 1 b. Pin 2

c. Pin 3 d. Pin 5

e. Pin 6

2.29 An 8085 microprocessor uses the following labels with assertion-level logic. Is each signal active-low or active-high?

a. HOLD b. $\overline{\text{RESET IN}}$

c. $\overline{\text{RD}}$ d. $\overline{\text{WR}}$

e. ALE f. INTR

g. $\overline{\text{INTA}}$

2.30 When switch $B$ of Fig. 2.54 is closed, is pin 6 high or low? For this condition, is B CLOSED negative true or negative false?

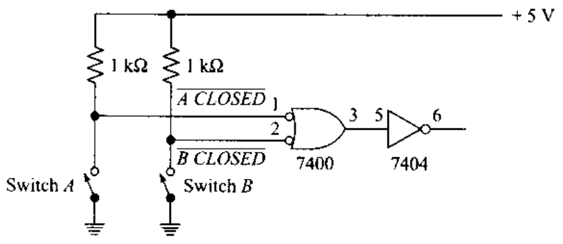


Fig. 2.54

## LABORATORY EXPERIMENT

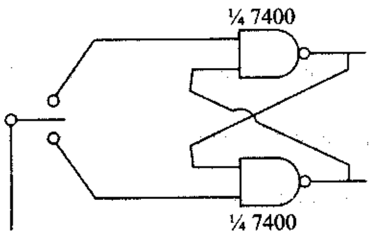
**AIM:** The aim of this experiment is to study basic NAND gate and implement a bounce-free switch using basic gates.

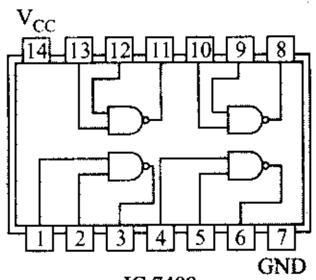
**Theory:** The NAND gate implements the logic

$Y = (A.B.C...)'$

where $A$, $B$, $C$, ... are inputs and $Y$ is output.

The mechanical switches used for electrical connection go through several make-break situations before resting in a particular position. For digital circuits, this could amount to a series of HIGH-LOW which is particularly detrimental to sequential logic circuit. The following circuit presents a NAND based debounce switch.





IC 7400

**Apparatus:** +5 Volt dc power supply, multimeter, bread board, and oscilloscope

**Work element:** IC 7400 is a quad 2-input NAND gate. Study its pin configuration and verify the NAND truth table. Study the debounce switch circuit and find out the principle behind its working. Use two NAND gates of IC 7400 and wire it as shown in the diagram. A wire may be used as a switch, the other side of which is connected to a dc power supply or Ground. Observe voltage level at the oscilloscope, at various points of the circuit when switching is done at the input side. Discuss the result. Design and implement a NOR (IC 7402) gate based debounce switch.

## Answers to Self-tests

1. positive
2. OR
3. AND
4. $Y = \overline{A}$
5. $Y = A + B$
6. $Y = A \cdot B = AB$
7. $Y = \overline{A + B}$
8. $\overline{A + B} = \overline{A} \cdot \overline{B}$
9. ≥
10. $Y = \overline{A \cdot B}$
11. $\overline{A \cdot B} = \overline{A} + \overline{B}$
12. &
13. There are two AND gates at the input.
14. It is used to increase the number of input AND gates.
15. Converse of positive logic. Here binary 0 stands for high voltage and binary 1 stands for low voltage.
16. It means drawing logic symbols to indicate the action of each signal. If the signal causes something to happen when low, it is drawn with a bubble; this is an active-low signal. If a signal causes something to happen when high, it is drawn without a bubble; this is an active-high signal.

# Combinational Logic Circuits

3

OBJECTIVES

- Demonstrate the ability to use basic Boolean laws.
- Use the sum-of-products method to design a logic circuit based on a design truth table.
- Be able to make Karnaugh maps and Entered variable maps and use them to simplify Boolean expressions.
- Use the product-of-sums method to design a logic circuit based on a design truth table.
- Use Quine-McClusky tabular method for logic simplification
- Analyze hazards in logic circuit and provide solution for them.

This chapter discusses Boolean algebra and several simplification techniques. After learning the laws and theorems of Boolean algebra, you can rearrange Boolean equations to arrive at simpler logic circuits. An alternative method of simplification is based on the Karnaugh map. In this approach, geometric rather than algebraic techniques are used to simplify logic circuits. Quine-McClusky tabular method provides a more systematic reduction technique, which is preferred when a large number of variables are in consideration.

There are two fundamental approaches in logic design: the sum-of-products method and the product-of-sums method. Either method produces a logic circuit corresponding to a given truth table. The sum-of-products solution results in an AND-OR or NAND-NAND network, while the product-of-sums solution results in an OR-AND or NOR-NOR network. Either can be used, although a designer usually selects the simpler circuit because it costs less and is more reliable. A practical logic circuit can show hazard due to finite propagation delay involved in each logic gate. This gives glitches or shows multiple transitions at the output. This chapter discusses different types of hazards and ways to prevent them.