

Digital Concepts

INTRODUCTION

The main difference between analog and digital operation is the way the loadline is used. With *analog* circuits, adjacent points on the loadline may be used, so that the output voltage is *continuous*. Because of this, the output voltage can have an infinite number of values. One way to get analog operation is with a *sinusoidal* input. The continuously changing input voltage produces a continuously changing output voltage.

Digital circuits are different. Almost all digital circuits are designed for *two-state* operation. This means using only two non-adjacent points on the loadline, typically *saturation* and *cut-off*. As a result, the output voltage has only two states (values), either *LOW* or *HIGH*. One way to get digital operation is with a *square-wave* input. If large enough, this type of input drives the transistors into saturation and cut-off, producing a two-state operation, as illustrated in Fig. 1.1.

DIGITAL/ANALOG SIGNALS

A *signal* can be defined as useful information transmitted within, to, or from electronic circuits. The circuit in Fig. 1.2(a) puts out an *analog signal* or voltage. As the wiper on the potentiometer is moved upward, the voltage from points A to B gradually increases. When the wiper is moved downward, the voltage gradually decreases. The waveform diagram in Fig. 1.2(b) is a graph of the analog output. On the left side, the voltage from A to B is gradually increasing to 5 V; on the right side, the voltage is gradually decreasing to 0 V. By stopping the wiper at any midpoint, we can get an output voltage anywhere between 0 and 5 V. An *analog device* then, is

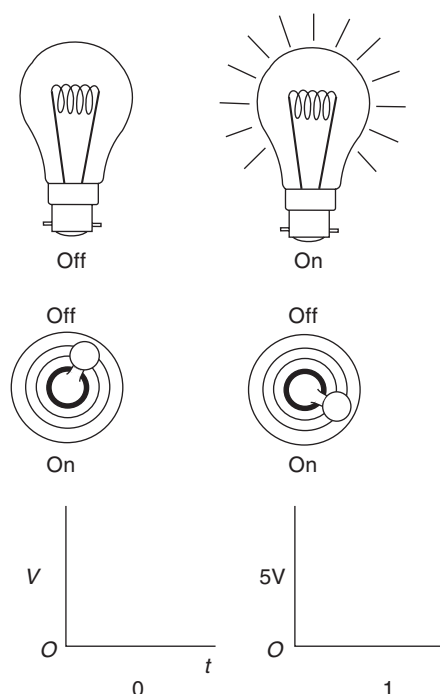


Fig. 1.1 Two-state (Binary) Operation.

one that has a signal which varies continuously in step with the input.

A *digital device* operates with a digital signal. The generator in Fig. 1.3(a) produces a square waveform that is displayed on the oscilloscope. The *digital signal* is either at +5 V or at 0 V as shown in Fig. 1.3(b). The voltage at point A moves from 0 to 5 V. The voltage then stays at +5 V for a time. At point B the voltage drops immediately from +5 V to 0 V. The voltage then stays at 0 V for a time. Only two voltages are present

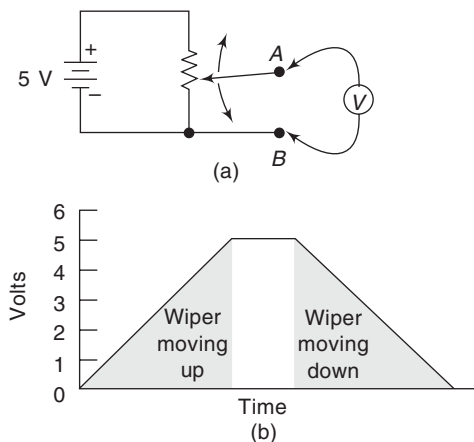


Fig. 1.2 (a) Analog Output from a Potentiometer.
(b) Analog Signal Waveform.

in a digital electronic circuit. In the waveform diagram in Fig. 1.3(b) these voltages are labelled HIGH and LOW. The HIGH voltage is called a *logical 1* (5 V) and the LOW voltage a *logical 0* (0 V). Circuits that handle only high and low signals are called digital circuits.

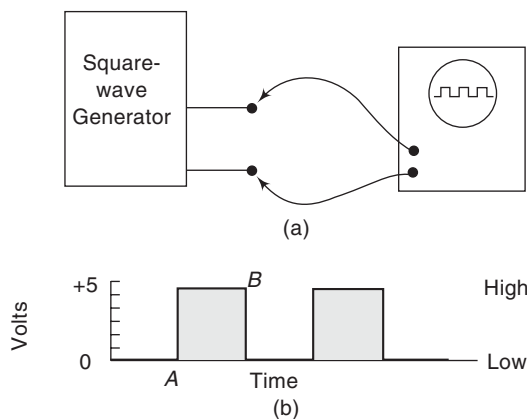


Fig. 1.3 (a) Digital Signal Displayed on Scope.
(b) Digital Signal Waveform.

Signals are commonly represented as a *voltage* varying with time, as in Figs 1.1 and 1.2. However, a signal could be an electrical *current* that varies continuously (analog) or has an *on-off* characteristic (digital).

Within most digital circuits, it is customary to represent signals in the *voltage versus time* format. When digital circuits are *interfaced* with non-digital devices, such as lamps and motors, then the signal can be thought of as *current versus time*. Interfacing refers to the design of interconnections that shift the *levels* of voltage and current to make them *compatible*.

DIGITAL DEVICES

An *analog device* is one in which data is represented by *physical variables*. A *digital device* is one in which data is represented by *numerical quantities*.

Figure 1.4 shows an analog clock and a digital clock. Both are in common use today. The analog device can be thought of as showing the information desired as an *analog*; the position of the hands of the clock is *analogous* to the time of day. The digital clock, on the other hand, shows the time of day as a *set of numbers*.

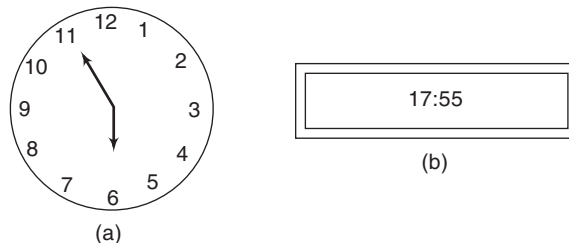


Fig. 1.4 (a) An Analog Clock. (b) A Digital Clock.

The standard volt-ohm milliammeter (VOM) shown in Fig. 1.5(a) is another example of an *analog measuring device*. As the voltage, resistance, or current being measured by the VOM increases, *the needle gradually and continuously moves up the scale*. The digital multimeter shown in Fig. 1.5(b) is an example of a *digital measuring device*. As the voltage, resistance, or current being measured by the DMM increases, *the display jumps upward in small steps*. The DMM is an example of digital circuitry taking over tasks previously performed only by analog devices.



Fig. 1.5 (a) Analog Multimeter (BPL).
(b) Digital Multimeter (Philips).

ADVANTAGES OF DIGITAL CIRCUITS

Digital circuits which use only two signal states have the following *advantages*:

- (1) Changes in component values have very little effect on digital signals.
- (2) Noise and other interfering signals have very little effect on digital signals.
- (3) The voltage anywhere in a digital circuit will always be in one state or the other, so that there is very little chance of confusion or error.
- (4) Digital signals are ideally suited for logic use, and for counting in the binary scale.
- (5) Information storage is easy.
- (6) Operation can be programmed.
- (7) More digital circuitry can be fabricated on IC chips.

1.1 Draw the waveform of (a) an analog signal, (b) a digital signal, and (c) a digital binary signal. Explain briefly.

Solution:

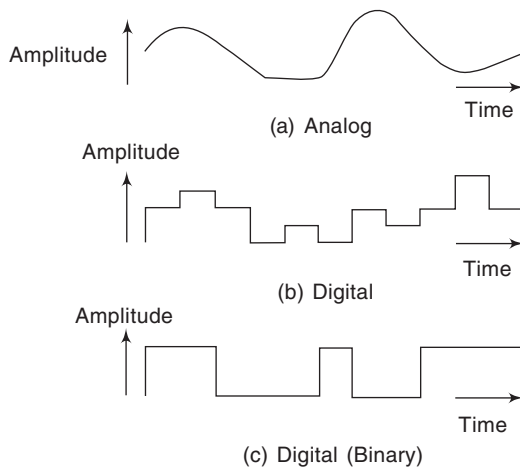


Fig. 1.6

- (a) An analog signal is *smooth* and not abrupt.
- (b) A digital signal is abrupt and not smooth.
- (c) A digital binary signal is a digital signal *so constrained* as to take on the values +5 V (1 or *high*) and 0 V (0 or *low*).

1.2 Compare two very common electronic systems—one an analog system and the other, its counterpart, a digital system.

Solution:

Record player (Fig. 1.7(a)): A sensor is used to detect the motion of a stylus as it tracks the displacement of a groove in a vinyl disc (record). The signal produced by this sensor is *analog* in form and corresponds to the sound pressure of the original recording. To reproduce

this sound, the sensor's output is amplified to produce a signal of sufficient power to drive a loudspeaker.

Compact disc player (Fig. 1.7(b)): Sound is stored on a compact disc in a *digital* format. This is achieved by repeatedly measuring the magnitude of the original analog sound signal and representing these measurements by numbers. This information is converted into a *binary form* and written onto the CD as a series of pits. The space between two pits is called *land*. The data stored on the disc is retrieved by a sensing arrangement that uses a laser to detect the presence or absence of these pits as the disc spins at high speed. The result is a train of binary information that is a *coded form* of the original measurements. In order to reproduce the recorded sound, this data is first *decoded* and then *converted* back to analog form using a digital-to-analog converter. This signal is then amplified to drive a loudspeaker in a CD player.

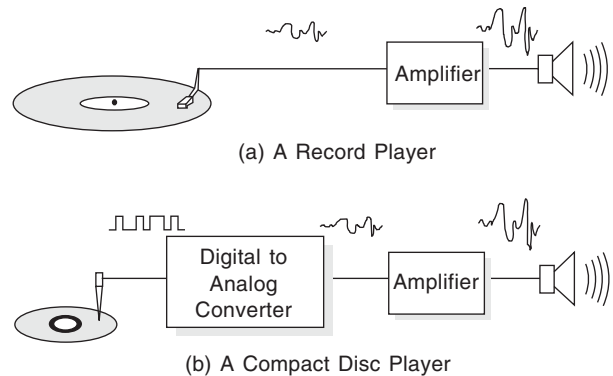


Fig. 1.7

1.3 Explain in detail the advantages of digital circuits.

Solution:

Let us look at the *reasons* for these advantages. Because any voltage which is close to the reference (earth) voltage counts as logic zero and any voltage which is close to the supply voltage counts as logic 1, *small changes* in signal voltages, which might be caused by resistors going higher or lower in value, or changes in the current gain of transistors, have no effect *unless the changes are very great*. Only a voltage change which causes the output of a digital circuit to be *indeterminate*—that is half way between 0 and 1—will cause problems. Similarly *unwanted noise signals* which are added to the 0 or 1 signals have no effect unless their amplitude is *comparatively large*, around half the supply voltage.

An *earthed input* in a digital circuit is a signal input—it sets the input voltage to *logic level 0*. An input which is connected to the *supply voltage* is also a signal input—it sets the input to *logic level 1*.

The use of only two states is an advantage because it *reduces the possibility of errors*—a volt or so change in a level will not mean that it can be mistaken for a different level. A digital type of signal, such as the staircase waveform in Fig. 1.8, which uses several levels, is very likely to cause errors because voltage levels cannot be kept within close limits.

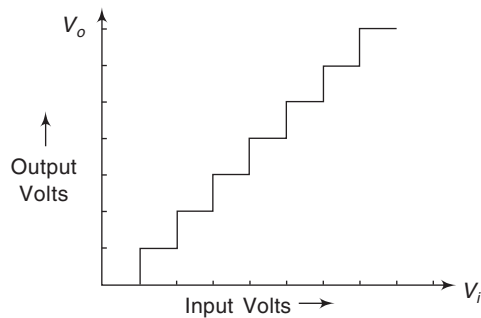


Fig. 1.8 A Staircase Waveform. The Output Voltage Changes only at Definite Input Values, Equally Spaced, and it Remains Steady between these Values.

Digital circuits which use only two states can be used for counting if a *binary scale* (scale of two) is used. The outputs of a digital circuit are usually these same logic levels, so that *voltage amplification* (as we understand it in linear circuits) is not used except to speed up the change from one level to the other. *Current amplification* is used to increase the amount of current that can be used at the output.

Information storage in digital systems is easy. This is accomplished by special devices and circuits that can latch onto digital information and hold it for as long as necessary. *Mass storage* techniques can store billions of bits of information in a relatively small space. Analog storage capabilities are, by contrast, extremely limited.

It is fairly easy to design digital systems whose operation is controlled by a set of stored instructions called a *program*. Analog circuits can also be *programmed*, but the variety and complexity of the available operations is extremely limited.

Analog circuits have also benefited from the unprecedented development of IC (integrated circuit) technology, but its relative complexity and its use of devices that cannot be economically integrated (high-value capacitors, precision resistors, inductors, transformers) have prevented analog systems from achieving the same degree of integration—*much more digital circuitry can be fabricated on digital ICs*.

LIMITATIONS OF DIGITAL CIRCUITS

There is only one major drawback when using digital techniques. *The real-world is analog in nature*. To take advantage of digital techniques when dealing with *analog inputs and outputs*, three steps must be followed:

- (1) *Convert* the real-world analog inputs to digital form (analog-to-digital conversion, *ADC*).
- (2) *Process* (operate on) the digital information.
- (3) *Convert* the digital outputs back to real-world analog form (digital-to-analog conversion, *DAC*).

1.4 Explain a simple temperature control system that requires analog/digital conversions in order to allow the use of digital processing techniques.

Solution:

Figure 1.9 shows the block diagram of a simple temperature control system. The analog temperature is measured and the measured value is then *converted* to a digital quantity by an analog-to-digital converter (ADC). The digital quantity is then *processed* by the digital circuitry, which may or may not include a digital computer.

Its digital output is *converted* back to an analog quantity by a digital-to-analog converter (DAC). This analog output is fed to a controller which takes some kind of action to adjust the temperature.

DIGITAL (BINARY) OPERATION OF A SYSTEM

A digital system functions in a binary manner. It employs devices which are permitted to exist in only two possible states. A transistor is allowed to operate at cut-off or in

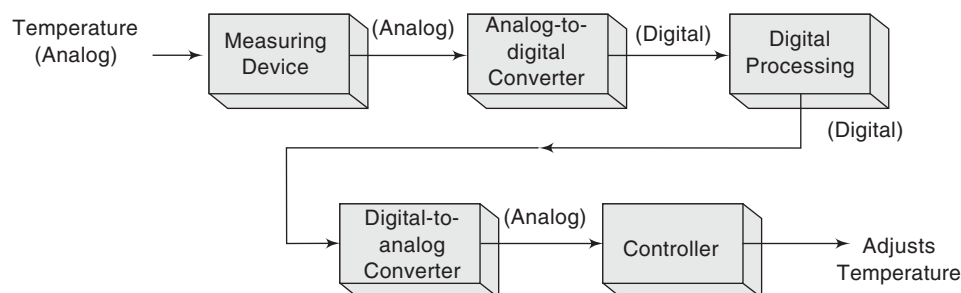


Fig. 1.9 Temperature Control System.

saturation, *but not in its active region*. A node may be at a high voltage, say, 12 ± 2 V or at a low voltage of, say 0 ± 0.2 V, *but no other values are allowed*. Various designations are used for these *quantized states*, and the most common of these are listed in Table 1.1. In logic, a statement is characterized as *true* or *false*, and this is the first binary classification listed in the table. A switch may be *closed* or *open*, which is the notation under 9, etc. Binary arithmetic and mathematical manipulation of switching and logic functions are best carried out with classification 3, which involves two symbols, 0 (zero) and 1 (one).

Table 1.1 Binary-state Terminology

	1	2	3	4	5	6	7	8	9	10	11
One of the states	True	HIGH	1	Up	Pulse	Excited	Off	Hot	Closed	North	Yes
...											
The other state...	False	LOW	0	Down	No pulse	Non-excited	On	Cold	Open	South	No

In a digital system all the electronic components operate in switching mode, because most of the devices act very much like switches, being either ON or OFF. For a variety of technical reasons, devices that function in the *switching mode* are simpler and cheaper to construct than those that function in the *continuous mode* and they function much more efficiently and reliably.

1.5 An open switch represents 0 and a closed switch represents 1. What number do the switches in Fig. 1.10 represent?

Solution:

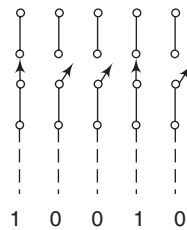


Fig. 1.10

1.6 The absence or presence of holes in paper represent 0 and 1, respectively. What binary numbers does the punched tape in Fig. 1.11 store?

Solution:

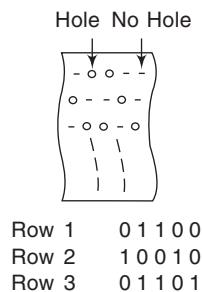


Fig. 1.11

WHY BINARY

A computer is often referred to as a *data processor*. Data means names, numbers, facts, anything needed to work out a problem. *Data* goes into a computer, where it is processed or manipulated to get new information. *A computer's circuits can respond only to binary numbers.*

Besides the data, someone has to work out a *program*, a list of instructions telling the computer what to do. *The program must be coded in binary form before it goes into the computer.*

So the two things we must *input* to a computer are the program and data. These are *stored* inside the computer before the processing begins. Once the run starts, each instruction is executed and the data is processed.

The electronic, magnetic, and mechanical devices of a computer are known as *hardware*. Programs are called *software*. Without software, a computer is a pile of 'dumb' metal.

Computers use *integrated circuits* (ICs) with thousands of transistors, either bipolar or MOS. These computer ICs work remarkably well despite variations. How is it possible? The answer is *two-state* design.

1.7 Explain the difference between computer hardware and software with the help of a suitable analogy.

Solution:

A *phonograph* is like hardware and *records* are like software. The phonograph is useless without records. Furthermore, the music you get depends on the record you play. A similar idea applies to computers. A *computer* is the hardware and *programs* are the software. The computer is useless without programs. The program stored in the computer determines what the computer will do: change the program and the computer processes the data in a different way.

MEMORY

When an input signal is applied to most devices or circuits, the output somehow changes in response to the input, and when the input signal is removed, the output returns to its original state. These circuits do not exhibit the property of memory. *Since their outputs revert back to normal, they are classed as non-memory circuits.*

In digital circuitry certain types of devices and circuits do have memory. When an input is applied to such a circuit, the output will change its state, but will remain in the same state even after the input is removed. *The property of retaining its response to a momentary input is called memory.* Figure 1.12 illustrates non-memory and memory operations.

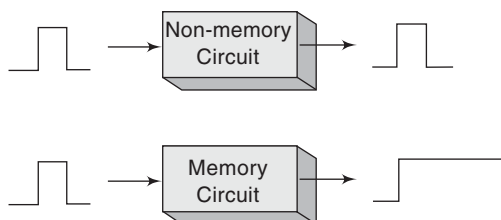


Fig. 1.12 Comparison of Memory and Non-memory Elements.

Memory devices and circuits play an important role in digital systems because they provide means for *storing* binary numbers, either temporarily or permanently, with the ability to *change* the stored information at any time.

A *register* is a string of devices that store data. Early computers used magnetic cores to store data.

1.8 Explain the working of a 4-bit (binary digit) core register.

Solution:

Figure 1.13(a) shows a *4-bit core register*. With the right-hand rule you can see that conventional current into a wire produces a *clockwise* flux; reversing the current produces a *counter-clockwise* flux. The same result is obtained if electron-flow is assumed and the left-hand rule is used.

The cores have rectangular hysteresis loops; this means that the flux *remains* in the core even after the magnetizing current is *removed* (see Fig. 1.13(b)). This is why a core register can *store* binary data indefinitely. By changing the magnetizing currents in Fig. 1.13(a) we can *change* the stored data.

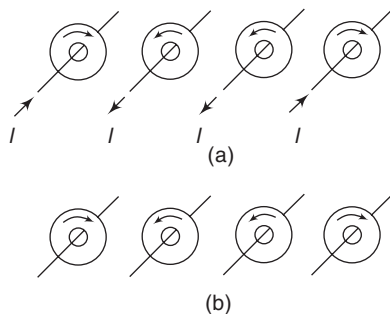


Fig. 1.13 Four-bit Core Register.

1.9 What binary number is stored in Fig. 1.13(b)?

Solution:

Using the following code:

Flux	Binary
Counterclockwise	0
Clockwise	1

Then, the core register of Fig. 1.13(b) stores binary 1001.

LOGIC CIRCUITS

We use many logical truths in everyday life. Most of the simple logical problems are distinguished by words such as *and*, *or*, *not*, *if*, *else* and *then*. Once a verbal reasoning process has been completed and results put into statements, the basic laws of logic can be used to evaluate the process. Simple *logic operations* can be performed by manipulating *verbal statements*. More complex relationships can be usefully represented by the use of symbols. These operations are known as *logic symbols*.

The circuits which perform the logic operations *sense the input conditions and provide an output only if certain input conditions exist*. They can be classified generally as *gate* circuits and they may employ transistors, semiconductor diodes, or magnetic cores. Logic gates most commonly employed are OR, AND, NOT and FLIP-FLOP. They are used to implement Boolean algebraic equations. Logic gates are used extensively in digital computers.

A *truth table* is a chart used in connection with logic circuits to illustrate *the states of the inputs and outputs under all possible signal conditions*. It provides a *ready reference* for use in analysing the operating theory of logic circuits.

MAJOR PARTS OF A COMPUTER

There are several types of computer systems, but each can be broken down into the same *functional units*. Each unit performs specific functions and all units function together to carry out the instructions given in the program. Figure 1.14 shows the five major functional parts of a digital computer and their *interaction*.

The *input device* lets us pass information from *person to machine* or *machine to machine*. The input device (keyboard, magnetic tape unit, network connection or telephone line, for example) must *encode* human language into the binary language of the computer.

The *arithmetic/logic unit* (ALU) adds, subtracts, multiplies, divides, compares, and does other logic functions. Data can be sent to the ALU for action and the results sent back to storage in the memory.

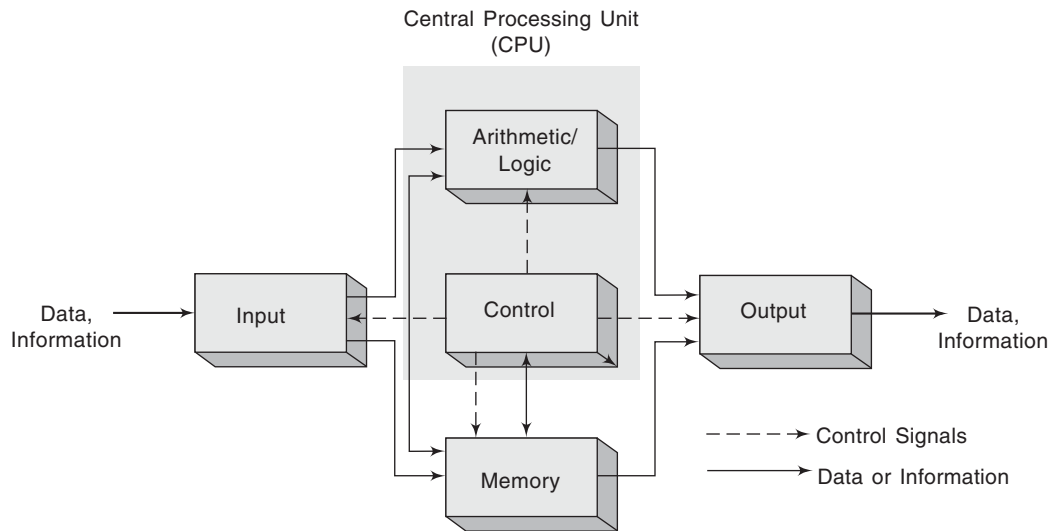


Fig. 1.14 Functional Parts of a Digital Computer.

The *memory unit* is the storage area for both data and programs. This storage can be *supplemented* by storage outside the processing unit.

The *control unit* is the *nervous system of the computer*. It directs all other sections to operate in the proper order and tells the input *when and where* to place information in memory. This unit takes instructions

from the memory unit one at a time and interprets them. It then sends appropriate signals to all the other units to cause the specific instruction to be executed.

The *output unit* is the link between the *machine and a person* (or to a device or network). The output unit must *decode* the language of the computer into human language.

SUMMARY

- An analog device is one that has a signal which varies continuously in step with the input.
- A digital device is one that has a signal with two-state operation.
- An analog device is one in which data is represented by physical variables.
- A digital device is one in which data is represented by numerical quantities.
- Digital signals are ideal for logic use, and for counting in the binary scale. Noise and other interfering signals have very little effect on digital signals. Information storage in digital systems is easy.
- Digital systems can be programmed. Much more digital circuitry can be fabricated on digital ICs.
- The real world is analog in nature.
- When dealing with analog inputs and output: convert (ADC), process, convert (DAC)
- Digital systems work in a binary manner.
- In a digital system all the components operate in switching mode.
- A computer's circuit can respond only to binary numbers.
- The property of retaining its response to a momentary input is called memory.
- A register is a string of devices that store data.
- Logic gates most commonly employed are OR, AND, NOT, and FLIP-FLOP.
- A truth table is a chart used in connection with logic circuits.
- A digital computer has five functional units: input, ALU, memory, control and output.

Test your
understanding

REVIEW QUESTIONS

1. Which of the following involve analog quantities and which involve digital quantities?
 - (a) Ten-position switch
 - (b) Current flowing out of an electrical outlet

- (c) Temperature of a room
- (d) Sand grains on the beach
- (e) Automobile speedometer
2. Concisely describe the difference between analog and digital quantities.
3. Give an example of a system that is analog and one that is a combination of both. Name a system that is entirely digital.
4. What are the advantages of digital techniques over analog?
5. What is the chief limitation to the use of digital techniques?
6. Define binary.
7. What does bit mean?
8. What are the bits in a binary system?
9. Why do we use binary?
10. Briefly describe logic circuits.
11. Which logic gates are most commonly employed?
12. What are the functional units of a digital computer?

Test your
understanding

SUPPLEMENTARY PROBLEMS

13. How will you generate a digital signal?
14. What type of signal will a push button generate?
15. How will you generate a string of digital pulses?
16. Figure 1.15 shows a strip of magnetic tape. The black circles are magnetized points and the white circles are unmagnetized points. What binary number does each horizontal row represent?
17. In Fig. 1.16, clockwise flux stands for binary 1 and counterclockwise flux stands for binary 0. What is the binary number stored in the 8-bit core register?

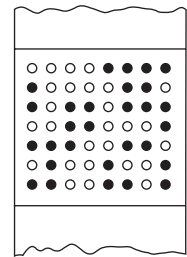


Fig. 1.15 Numbers on Magnetic Tape.

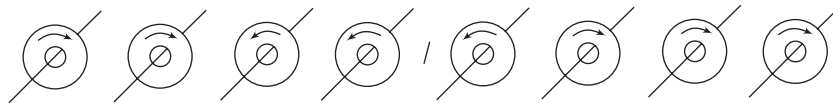


Fig. 1.16 An 8-bit Core Register.

18. Figure 1.17 shows a 5-bit switch register. By opening and closing the switches you can set up different binary numbers. HIGH output voltage stands for binary 1 and LOW output voltage for binary 0. What is the number stored in switch register?

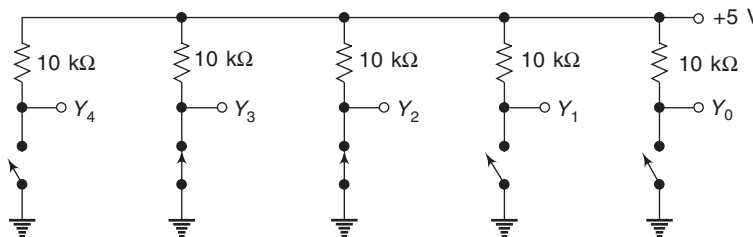


Fig. 1.17 A 5-bit Switch Register.

Test your
understanding

OBJECTIVE TYPE QUESTIONS

Fill in the Blanks

19. Analog signals are continuous and not _____.
20. A digital signal does not change voltage _____ it does so _____.

21. Only two voltages are present in a _____ electronic circuit.
22. An analog device is one in which data is represented by _____.
23. A digital device is one in which data is represented by _____.
24. The real world is _____ in nature.
25. In a digital system all the electronic components operate in a _____ mode.
26. The property of retaining its response to a momentary input is called _____.
27. Memory devices provide a means of _____ binary numbers.
28. The program must be coded in _____ before it goes into the computer.
29. Programs are called _____.
30. The input device lets us pass information from _____ to machine.
31. A truth table provides a _____ for use in analysing the operating theory of logic circuits.
32. Each functional unit performs a _____ function and all functional units function _____ to carry out program instructions.
33. The _____ unit is the nervous system of the computer.

True/False Questions

State whether the following statements are True or False.

34. The exact value of an input voltage is critical for a digital circuit.
35. A digital circuit is also referred to as a logic circuit.
36. One way to get analog operation is with a square-wave input.
37. Non-memory devices provide means for storing binary numbers.
38. The output unit of a computer is the link between machine and a person.
39. There is no interaction between the functional units of a computer.

Multiple Choice Questions

40. A quantity having continuous values is
 - (a) a digital quantity
 - (b) an analog quantity
 - (c) a binary quantity
 - (d) a natural quantity
41. Which of the following systems are digital:
 - (a) electronic calculator
 - (b) pressure gauge
 - (b) clinical thermometer
 - (d) ordinary electric switch
42. The term bit means
 - (a) a small amount of data
 - (b) a 1 or 0
 - (c) a binary digit
 - (d) both answers (b) and (c)
43. The property of retaining its response to a momentary input is called
 - (a) conversion
 - (b) memory
 - (c) storing
 - (d) coding
44. In a digital system all the electronic components operate in
 - (a) continuous mode
 - (b) pulse mode
 - (c) active mode
 - (d) switching mode
45. A digital signal can be generated with the help of
 - (a) mechanical switch
 - (b) push button switch
 - (c) mechanical relay
 - (d) multiplexer

ANSWERS

1. (a) Digital (b) Analog (c) Analog (d) Digital (e) Analog, if needle type; digital, if numerical readout type.
2. Analog quantities can take on *any* values over a continuous range; digital quantities can take on only *discrete* values.
3. A public address system is analog. A CD player is analog and digital. A computer is all digital.
4. Greater accuracy and precision; less affected by noise; easier to design; easier to store information; ideally suited for logic use; operations can be programmed; higher degree of integration.
5. Real world physical quantities are analog.
6. Binary means having two states or values.
7. A bit is a binary digit.

8. The bits are 1 and 0.
9. A computer's circuits can respond only to binary numbers; the program must be coded in binary form before it goes into the computer; computer ICs work remarkably well, despite variations, because of two-state design.
10. Logic circuits sense the input conditions and provide an output only if certain input conditions exist.
11. OR, AND, NOT and FLIP-FLOP.
12. Input, ALU, memory, control and output.
- 13.

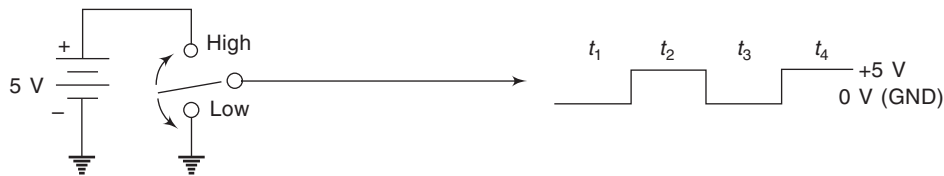


Fig. 1.18 Generating a Digital Signal.

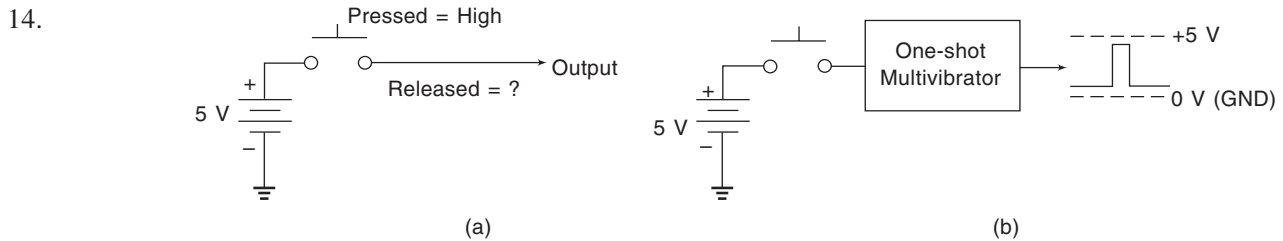


Fig. 1.19 (a) Push button will not generate a digital signal. (b) Push button used to trigger a one-shot multivibrator for a single-pulse digital signal.

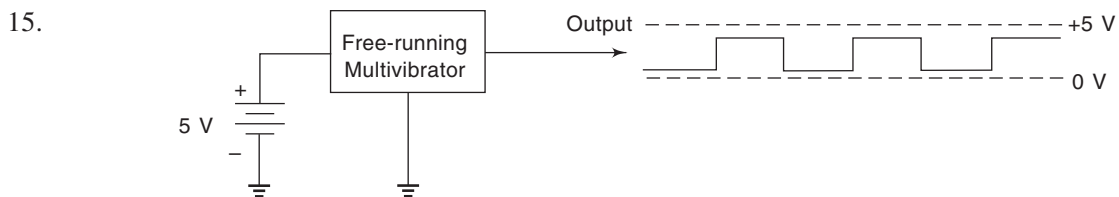


Fig. 1.20 A free-running multivibrator generates a string of digital pulses.

16. Row 1 0 0 0 0 1 1 1 1 Row 5 1 1 1 0 0 1 1 0
- Row 2 1 0 0 0 0 1 1 0 Row 6 0 1 0 0 1 0 0 1
- Row 3 1 0 1 1 0 1 1 1 Row 7 1 1 0 0 1 1 0 1
- Row 4 0 0 1 1 0 0 0 1
17. 1 1 0 0 0 1 1 1 18. 1 0 0 1 119. abrupt20. continuously, abruptly
21. digital22. physical variables23. numerical quantities
24. analog25. switching26. memory27. storing
28. binary29. software30. man31. ready reference
32. specific, together33. control34. False35. True
36. False37. False38. True39. False
40. (b)41. (a) and (d)42. (c)43. (b)
44. (d)45. (a)

Number Systems

INTRODUCTION

The base or *radix* of a chosen number system comes about through some particular *convenience*. Doubtless human beings prefer the decimal system based upon a radix of 10, because they began by counting on their fingers. A radix of 10 gives 10 different states. Since digital systems are based upon circuitry having only two different states, *binary arithmetic* is employed because this has a radix of 2.

Large numbers in the binary system become *unwieldy* in length, and whilst this presents no problems to the electronics of a system, it does present problems at the *human interface*. Because of this there are various other number systems in use which conveniently interface with the binary system. These systems express numbers in a more convenient and shorter form, e.g. *octal*, *binary coded decimal*, and *hexadecimal*.

2.1 How will you express the form of a number in any number system?

Solution:

Using the letter r to represent the radix of any number system, the form of any number can be expressed as:

$$Y = d_n r^n + d_{n-1} r^{n-1} + \dots + d_1 r^1 + d_0 r^0 \quad (2.1)$$

where Y is the value of the entire number, d_n is the value of the n th digit from the radix point and r is the radix or base.

2.2 Explain the terms:

- (a) most significant digit (b) least significant digit
(c) radix point.

Solution:

- (a) The most significant digit (*MSD*), the one with the *highest* power of the radix, is always at the left.

- (b) The least significant digit (*LSD*), the one with the *lowest* power of the radix, is always at the right.

- (c) Any decimal number can be expressed as:

$$\dots 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0 \ . \ 10^{-1} \ 10^{-2} \ 10^{-3} \ 10^{-4} \dots$$

↓

decimal point

Any binary number can be expressed as:

$$\dots 2^4 \ 2^3 \ 2^2 \ 2^1 \ 2^0 \ . \ 2^{-1} \ 2^{-2} \ 2^{-3} \ 2^{-4} \dots$$

↓

binary point

Using the letter r to represent the radix of any number system, the form of any number can be expressed as:

$$\dots r^4 \ r^3 \ r^2 \ r^1 \ r^0 \ . \ r^{-1} \ r^{-2} \ r^{-3} \ r^{-4} \dots$$

↓

radix point

2.3 Explain the idea of positional weighting.

Solution:

Any number of any magnitude can be expressed by using the system of *positional weighting*. The position of the digit with reference to the *radix point* determines the weight. The principle of positional weighting can be extended to any number system.

2.4 What weight does the digit 7 have in each of the following numbers?

- (a) 1370 (b) 6725 (c) 7501 (d) 75,898

Solution:

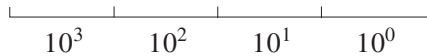
- (a) 10 (b) 100 (c) 1,000 (d) 10,000

2.5 Evaluate the four digit decimal number 3264.

Solution:

In a four digit decimal number, the *least significant* position (right most) has a weighting factor of 10^0 ; the

most significant position (left most) has a weighting factor of 10^3 :



where $10^3 = 1000$, $10^2 = 100$, $10^1 = 10$ and $10^0 = 1$

To evaluate the decimal number 3264, the digit in each position is multiplied by the appropriate weighting factor:

3	2	6	4		
				→	
				→	$4 \times 10^0 = 4$
				→	$6 \times 10^1 = 60$
				→	$2 \times 10^2 = 200$
				→	$3 \times 10^3 = +3000$
					3264

THE BINARY NUMBER SYSTEM

Binary numbers are used extensively in all digital systems because of the very nature of electronics. A 1 can be represented by a saturated transistor, a light turned on, a relay energized, or a magnet magnetized in a particular direction. A 0 can be represented by a cut-off transistor, a light turned off, a de-energized relay, or a magnet magnetized in the opposite direction. In each case there are *only two values* that the device can assume. The binary number system has a *radix* of 2.

2.6 Illustrate binary elements.

Solution:

Binary elements are illustrated in Fig. 2.1.

2.7 Evaluate the five bit binary numbers 10101.

Solution:

We can find its weighting value simply by adding the weighting value of each position that has a 1 below it. Adding from right to left, we obtain:

$$\begin{aligned}
 10101 &= 1 \times 2^0 + 1 \times 2^2 + 1 \times 2^4 \\
 &= 1 + 4 + 16 = 21
 \end{aligned}$$

2.8 Evaluate the number $(312.4)_5$.

Solution:

$$\begin{aligned}
 (312.4)_5 &= 3 \times 5^2 + 1 \times 5^1 + 2 \times 5^0 + 4 \times 5^{-1} \\
 &= 75 + 5 + 2 + 0.8 \\
 &= (82.8)_{10}
 \end{aligned}$$

Note: If there is any doubt about the number system being employed, it should be clarified by writing the radix of the number as a subscript to the number. Thus,

$$\begin{array}{ll}
 125_{10} & \text{decimal number system} \\
 100_2 & \text{binary number system}
 \end{array}$$

2.9 How will you use your fingers for coding binary numbers?

Solution:

A basic unit which we personally can use as a counter is the finger. If it is raised, we say it has value; if it is

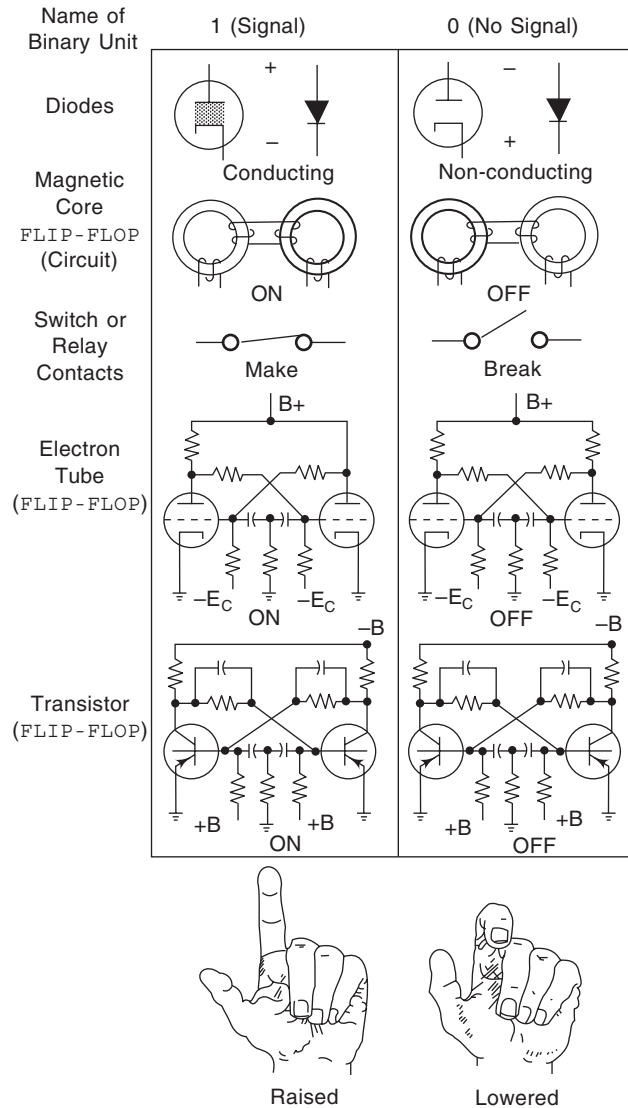


Fig. 2.1 Binary Elements.

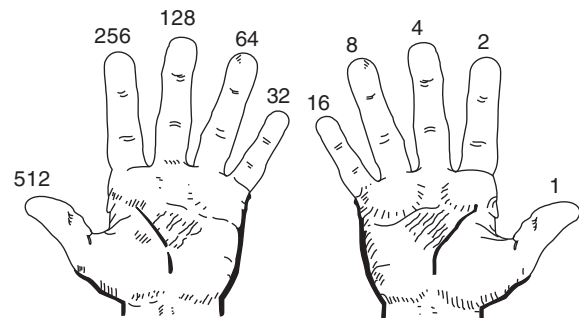


Fig. 2.2 A ready-made Binary Digital Computer.

lowered, it has no value. It is, therefore, a *binary element*. Ten figures provide us with a ready-made binary digital computer. By assigning values to the fingers of each hand, we can count any number between 0 and 1023.

2.10 Discuss the significance of zeros in binary numbers.

Solution:

Zeros, whether they are to the right or left, never add to the value of the binary number. Zero times any number is zero. Thus, for practical purposes, only the 1's have to be multiplied by their weighting values and added to each other as shown in Fig. 2.3.

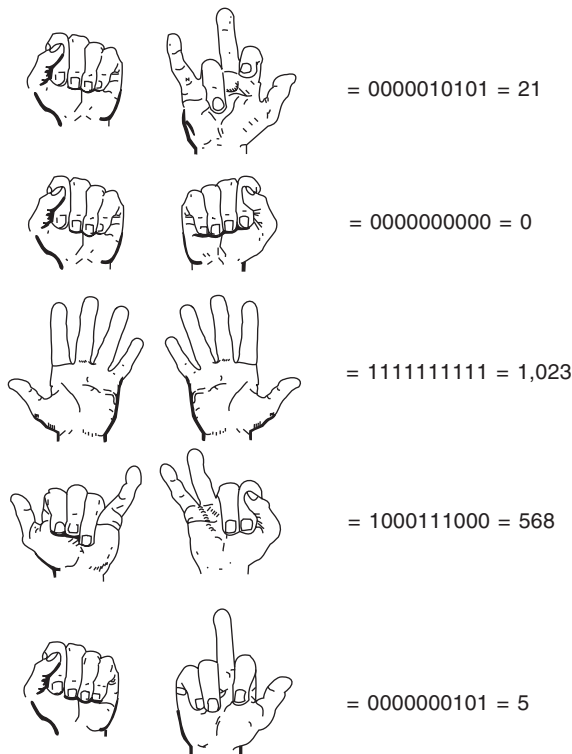


Fig. 2.3 Finding the Value of Binary Numbers.

Some zeros are required however. The zeros to the *right* of the highest valued 1 serve as *place-keepers* or *spaces* to retain the 1's in their correct position. The zeros to the *left* however provide no information about the number, and hence can be eliminated. Thus, 00101 can be written as 101.

2.11 How will you write a binary sequence?

Solution:

An easy way to write a binary sequence is illustrated in Fig. 2.4.

1. The right-most column begins with a 0 and *alternates each bit*.
2. The next column begins with two 0's and *alternates every two bits*
3. The next column begins with four 0's and *alternates ever four bits*.
4. The next column begins with eight 0's and *alternates every eight bits*.
5. The next column begins with sixteen 0's and *alternates every sixteen bits*.

Binary	Sequence	Decimal
0	0	0
0	0	1
0	0	2
0	0	3
0	1	4
0	1	5
0	1	6
0	1	7
1	0	8
1	0	9
1	0	10
1	0	11
1	1	12
1	1	13
1	1	14
1	1	15

Fig. 2.4 Writing Binary Sequence.

FRACTIONAL BINARY NUMBERS

Although seldom used in digital systems, binary weightings for values less than 1 (fractional binary numbers) is possible. These factors are developed by successively dividing the weighting factors by 2 for each decrease in power of 2 as shown in Fig. 2.5.

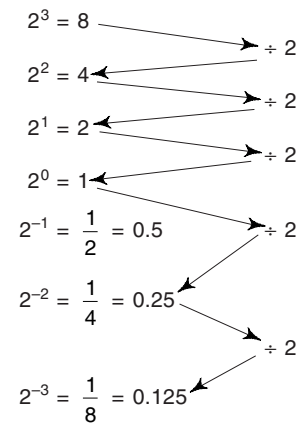


Fig. 2.5 Successive Division by 2 to Develop Fractional Binary Weighting Factors.

BINARY-TO-DECIMAL CONVERSION

Converting a binary number to decimal merely requires *substitution* of numbers into Eq. (2.1) remembering that the *d*'s will all be 0's or 1's, the *r*'s will all be 2 (the radix), and the *n*'s will be the various powers of 2, depending on the position of the digit with reference to the radix point.

2.12 Convert 10111₂ to decimal.

Solution:

$$\begin{aligned}
 Y &= d_4r^4 + d_3r^3 + d_2r^2 + d_1r^1 + d_0r^0 \\
 &= 1 \times 2^4 + 0 \times 2^3 + 1 \times 2^2 + 1 \times 2^1 + 1 \times 2^0 \\
 &= 16 + 0 + 4 + 2 + 1
 \end{aligned}$$

$$10111_2 = 23_{10}$$

(a) **110101**

(a) **110101**

(b) 101001

Powers of two	2^5	2^4	2^3	2^2	2^1	2^0
Decimal weights	32	16	8	4	2	1

(a) $110101 = 32 + 16 + 4 + 1 = 53$

(b) $101001 = 32 + 8 + 1 = 41$

2.14 Convert the fractional binary number 1011.1010, to decimal.

$1 \times 2^{-3} = 0.125$
 $1 \times 2^{-1} = 0.500$
 $1 \times 2^0 = 1$
 $1 \times 2^1 = 2$
 $1 \times 2^3 = 8$

 11.625

BINARY ARITHMETIC

2.15 What are the basic rules for *adding* binary numbers?

There are four basic rules for adding binary numbers. Three of the rules result in a single bit. The addition of two 1's yields a binary two (10_2). When binary numbers are added, the latter condition creates a sum of 0 in a given column and a *carry* of 1 over to the next higher column.

$$\left\{ \begin{array}{l} 0 + 0 = 0 \\ 0 + 1 = 1 \\ 1 + 0 = 1 \\ 1 + 1 = 10_2 \end{array} \right\}$$

$$\left\{ \begin{array}{ll} 1 + 0 + 0 = 01_2 & 1 \text{ with a carry of } 0 \\ 1 + 0 + 1 = 10_2 & 0 \text{ with a carry of } 1 \\ 1 + 1 + 0 = 10_2 & 0 \text{ with a carry of } 1 \\ 1 + 1 + 1 = 11_2 & 1 \text{ with a carry of } 1 \end{array} \right.$$

(a) 111, and 11, (b) 11100, and 10011,

$$\begin{array}{rcl} \text{(a)} & \begin{array}{r} 1 \ 1 \ 1_2 \\ + \ 1 \ 1_2 \\ \hline 10 \ 1 \ 0_2 \end{array} & \rightarrow \begin{array}{r} 7_{10} \\ + \ 3_{10} \\ \hline 10_{10} \end{array} \\ \text{(b)} & \begin{array}{r} 1 \ 1 \ 1 \ 0 \ 0_2 \\ + \ 1 \ 0 \ 0 \ 1 \ 1_2 \\ \hline 10 \ 1 \ 1 \ 1 \ 1_2 \end{array} & \rightarrow \begin{array}{r} 28_{10} \\ + \ 19_{10} \\ \hline 47_{10} \end{array} \end{array}$$

(a) 01101₂ and 01110₂, (b) 1111₂ and 1100₂.

$$\begin{array}{lcl} \text{(a)} & \begin{array}{r} 0 \ 1 \ 1 \ 0 \ 1_2 \\ + \ 0 \ 1 \ 1 \ 1 \ 0_2 \\ \hline 1 \ 1 \ 0 \ 1 \ 1_2 \end{array} & \rightarrow \begin{array}{r} 13_{10} \\ + \ 14_{10} \\ \hline 27_{10} \end{array} \\ \text{(b)} & \begin{array}{r} 1 \ 1 \ 1 \ 1 \ 1_2 \\ + \ 1 \ 1 \ 0 \ 0_2 \\ \hline 1 \ 1 \ 0 \ 1 \ 1_2 \end{array} & \rightarrow \begin{array}{r} 15_{10} \\ + 12_{10} \\ \hline 27_{10} \end{array} \end{array}$$

There are four basic rules for subtracting binary numbers. When subtracting binary numbers, we sometimes have to *borrow* from the next higher column. A borrow is required in binary only when we try to subtract a 1 from a 0. In this case when a 1 is borrowed from the next higher column, a 10 is created in the *column being subtracted*, and the last of the basic rules must be applied.

$$\left\{ \begin{array}{l} 0 - 0 = 0 \\ 1 - 0 = 1 \\ 1 - 1 = 0 \\ 10_2 - 1 = 1 \end{array} \right\}$$

(a) 10_2 from 11_2 (b) 011_2 from 101_2

$$\begin{array}{lcl} \text{(a)} & \begin{array}{r} 11_2 \\ -10_2 \\ \hline 1_2 \end{array} & \rightarrow \begin{array}{r} 3_{10} \\ 2_{10} \\ \hline 1_{10} \end{array} \\ \text{(b)} & \begin{array}{r} 101_2 \\ -011_2 \\ \hline 10_2 \end{array} & \rightarrow \begin{array}{r} 5_{10} \\ -3_{10} \\ \hline 2_{10} \end{array} \end{array}$$

2.20 Subtract the following numbers. Also show the equivalent decimal subtraction:

(a) 01011_2 from 11011_2 (b) 011_2 from 101_2

Solution:

$$\begin{array}{r} \text{(a)} \quad 11011_2 \rightarrow 27_{10} \\ - 01011_2 \quad 11_{10} \\ \hline 10000_2 \quad 16_{10} \end{array} \quad \begin{array}{r} \text{(b)} \quad 101_2 \rightarrow 5_{10} \\ - 011_2 \quad 3_{10} \\ \hline 10_2 \rightarrow 2_{10} \end{array}$$

2.21 What are the basic rules for multiplying binary numbers?

Solution:

There are four basic rules for multiplying binary numbers. Multiplication involves forming the *partial products*, shifting each partial product left one place and then adding all the partial products.

$$\left\{ \begin{array}{l} 0 \times 0 = 0 \\ 0 \times 1 = 0 \\ 1 \times 0 = 0 \\ 1 \times 1 = 1 \end{array} \right.$$

2.22 Carry out the following multiplications. Also show the equivalent decimal multiplication.

(a) $111_2 \times 101_2$ (b) $1011_2 \times 1001_2$

Solution:

$$\begin{array}{r} \text{(a)} \quad 111_2 \rightarrow 7_{10} \\ \times 101_2 \quad \times 5_{10} \\ \hline 111_2 \quad 35_{10} \\ 000_2 \\ 111_2 \\ \hline 100011_2 \end{array}$$

$$\begin{array}{r} \text{(b)} \quad 1011_2 \rightarrow 11_{10} \\ \times 1001_2 \quad \times 9_{10} \\ \hline 1011_2 \quad 99_{10} \\ 0000_2 \\ 0000_2 \\ 1011_2 \\ \hline 1100011_2 \end{array}$$

2.23 Carry out the following multiplications. Also show the equivalent decimal multiplication.

(a) $10101_2 \times 101_2$ (b) $11111_2 \times 10011_2$

Solution:

$$\begin{array}{r} \text{(a)} \quad 10101_2 \rightarrow 21_{10} \\ \times 101_2 \quad \times 5_{10} \\ \hline 10101_2 \quad 105_{10} \\ 00000_2 \\ 10101_2 \\ \hline 1101001_2 \end{array}$$

$$\begin{array}{r} \text{(b)} \quad 11111_2 \rightarrow 31_{10} \\ \times 10011_2 \quad 19_{10} \\ \hline 11111_2 \quad 279_{10} \\ 00000_2 \quad 31_{10} \\ 00000_2 \quad 589_{10} \\ 11111_2 \\ \hline 1001001101_2 \end{array}$$

2.24 What are the basic rules for division in binary?

Solution:

Division in binary follows the same procedure as division in decimal.

2.25 Carry out the following divisions. Also show the equivalent decimal division.

(a) $110_2 \div 11_2$ (b) $1111_2 \div 110_2$

Solution:

$$\begin{array}{r} \text{(a)} \quad 10_2 \rightarrow 2_{10} \\ 11_2 \overline{)110_2} \quad 3_{10} \overline{)6_{10}} \\ \underline{11_2} \quad \underline{6_{10}} \\ 000_2 \quad 0_{10} \end{array}$$

$$\begin{array}{r} \text{(b)} \quad 10.1 \rightarrow 2.5_{10} \\ 110_2 \overline{)1111.0_2} \quad 6_{10} \overline{)15_{10}} \\ \underline{110_2} \quad \underline{12_{10}} \\ 110_2 \quad 30_{10} \\ \underline{110_2} \quad \underline{30_{10}} \\ 000_2 \quad 00_{10} \end{array}$$

2.26 Carry out the following divisions. Also show their equivalent decimal division.

(a) $11111111_2 \div 110011_2$ (b) $110000_2 \div 100_2$

Solution:

$$\begin{array}{r} \text{(a)} \quad 101_2 \rightarrow 5_{10} \\ 110011_2 \overline{)11111111_2} \quad 51_{10} \overline{)255_{10}} \\ \underline{110011_2} \quad \underline{255_{10}} \\ 110011_2 \quad 000_{10} \\ \underline{110011_2} \\ 000000_2 \end{array}$$

$$\begin{array}{r} \text{(b)} \quad 1100_2 \rightarrow 12_{10} \\ 100_2 \overline{)110000_2} \quad 4_{10} \overline{)48_{10}} \\ \underline{100_2} \quad \underline{4} \\ 100_2 \quad 08_{10} \\ \underline{100_2} \quad \underline{8_{10}} \\ 000_2 \quad 0_{10} \end{array}$$

COMPLEMENTS IN NUMBER SYSTEMS

The complement number system was invented to make addition and subtraction faster and easier to implement by omitting the need for sign and magnitude comparison. Instead it requires *complementation* which is performed quite efficiently on binary numbers.

The *true complement* of a binary number is formed by subtracting each digit of the number from radix-minus-one of the number system and then adding 1 to the least significant digit. The true complement of a number in the decimal system is referred to as the *10's complement* and in the binary system it is referred to as the *2's complement*.

The *radix-minus-one complement* in each system is formed by subtracting each digit of the number from the radix-minus-one. It is 9 for the decimal system and 1 for the binary system.

The *1's complement* and *2's complement* of a binary number are important because they permit the representation of *negative numbers*. The method of 2's complement arithmetic is commonly used in computers to handle negative numbers.

2.27 How will you carry out 9's complement subtraction?

Solution:

Subtraction of a *smaller* decimal number from a *larger* decimal number is accomplished by adding 9's complement of the subtrahend (in this case the smaller number) to the minuend and then adding the carry to the result. This is called *end-around-carry*.

When subtracting a *larger* decimal number from a *smaller* decimal number, there is no carry, and the result is in 9's complement form and negative. This procedure has a distinct advantage in certain types of arithmetic logic.

2.28 Find the 9's complement of

- (a) 28 (b) 562 (c) 3497

Solution:

(a)	99	(b)	999	(c)	9999
	$\begin{array}{r} 99 \\ -28 \\ \hline 71 \end{array}$		$\begin{array}{r} 999 \\ -562 \\ \hline 437 \end{array}$		$\begin{array}{r} 9999 \\ -3497 \\ \hline 6502 \end{array}$

2.29 Perform the following subtractions using the 9's complement method.

- (a) 13 – 7 (b) 15 – 28

Solution:

	Regular subtraction	9's complement subtraction
(a)	$\begin{array}{r} 13 \\ -07 \\ \hline 06 \end{array}$	$\begin{array}{r} 13 \\ +92 \\ \hline 05 \end{array}$ 9's complement of 07 $\begin{array}{r} \textcircled{1} \quad 05 \\ \rightarrow +1 \\ \hline 06 \end{array}$ Add carry to result

(b)	$\begin{array}{r} 15 \\ -28 \\ \hline -13 \end{array}$	$\begin{array}{r} 15 \\ +71 \\ \hline 86 \\ -13 \\ \hline \end{array}$	9's complement of 28 9's complement of result <i>No carry indicates that the answer is negative and in complement form.</i>
-----	--	--	---

2.30 How will you carry out 10's complement subtraction?

Solution:

The 10's complement of a decimal number can be used to perform subtraction by adding the minuend to the 10's complement of the subtrahend and *dropping the carry*.

2.31 Convert the following decimal numbers to their 10's complement form:

- (a) 52 (b) 428

Solution:

(a)	$\begin{array}{r} 99 \\ -52 \\ \hline 47 \end{array}$	(b)	$\begin{array}{r} 999 \\ -428 \\ \hline 571 \end{array}$	9's complement form
	$\begin{array}{r} \oplus 1 \\ \hline \rightarrow 48 \end{array}$		$\begin{array}{r} \oplus 1 \\ \hline \rightarrow 572 \end{array}$	Add 1 10's complement

2.32 Perform the following subtractions by using the 10's complement method.

- (a) 54 – 21 (b) 196 – 155

Solution:

	Regular subtraction	10's complement subtraction
(a)	$\begin{array}{r} 54 \\ -21 \\ \hline 33 \end{array}$	$\begin{array}{r} 54 \\ +79 \\ \hline 133 \end{array}$ 10's complement of 21 $\begin{array}{r} \cancel{1} 33 \end{array}$ Drop carry
(b)	$\begin{array}{r} 196 \\ -155 \\ \hline 41 \end{array}$	$\begin{array}{r} 196 \\ +845 \\ \hline 1041 \end{array}$ 10's complement of 155 $\begin{array}{r} \cancel{1} 041 \end{array}$ Drop carry

2.33 How will you carry out 1's complement subtraction?

Solution:

1's complement method allows us to subtract using only addition.

To subtract a *smaller* number from a *larger* one, the following steps are involved:

1. Determine the 1's complement of the smaller number.
2. Add the 1's complement to the larger number.
3. *Remove the carry and add it to the result* (end-around carry).

To subtract a *larger* number from a *smaller* number the following steps are involved:

1. Determine the 1's complement of the larger number.
2. Add the 1's complement to the smaller number.
3. The answer has an opposite sign and is the 1's complement of the result. *There is no carry.*

2.34 Find the 1's complement of the following binary numbers:

(a) 101_2 (b) 1101_2 (c) 11100_2

Solution:

	Binary number	1's complement
(a)	101_2	010_2
(b)	1101_2	0010_2
(c)	11100_2	00011_2

2.35 Subtract 10011_2 from 11001_2 using the 1's complement method. Also show direct subtraction for comparison.

Solution:

Direct subtraction	1's complement subtraction	
11001_2	11001_2	
-10011_2	$+01100_2$	1's complement
<u> </u>	<u> </u>	
00110_2	$\textcircled{1}00101_2$	
	$\downarrow +1$	Add end-around carry
	<u> </u>	
	00110_2	

2.36 Subtract 1101_2 from 1001_2 using the 1's complement method.

Also show direct subtraction for comparison.

Solution:

Direct subtraction	1's complement subtraction	
1001_2	1001_2	
-1101_2	$+0010_2$	1's complement
<u> </u>	<u> </u>	
-0100_2	1011_2	Answer in 1's complement form and opposite in sign
	\downarrow	
	-0100	Final answer.

2.37 How will you obtain the 1's complement of a binary number with a digital circuit?

Solution:

The simplest way to obtain the 1's complement of a binary number with a digital circuit is to use parallel inverters (NOT circuits) as shown in Fig. 2.6 for an 8-bit binary number. *An inverter negates the input.*

2.38 How will you obtain the 2's complement of a negative binary number with a digital circuit?

Solution:

The 2's complement of a negative binary number can be realised using inverters and an adder as shown in

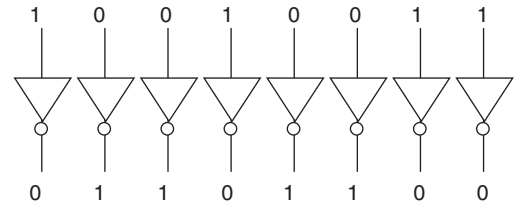


Fig. 2.6 Example of Inverters to Obtain 1's Complement of a Binary Number.

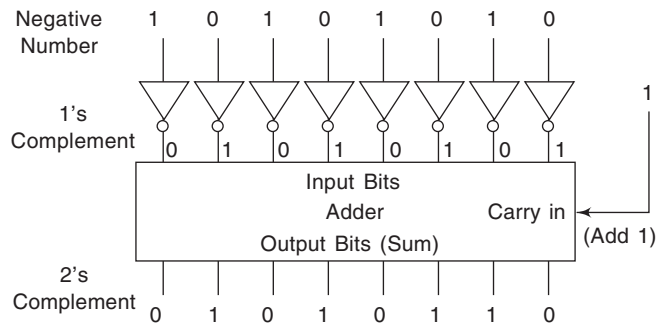


Fig. 2.7 Obtaining the 2's Complement of a Negative Binary Number.

Fig. 2.7. Invert each bit to take 1's complement and then add 1 to the 1's complement with an adder circuit.

2.39 How will you carry out 2's complement subtraction?

Solution:

To subtract a smaller number from a larger number, the following steps are involved:

1. Determine the 2's complement of the smaller number.
2. Add the 2's complement to the larger number.
3. Discard the carry (*there is always a carry in this case*).

To subtract a larger number from a smaller number the following steps are involved:

1. Determine the 2's complement of the larger number.
2. Add the 2's complement to the smaller number.
3. *There is no carry. The result is in 2's complement form and is negative.*
4. To get the answer in true form, take the 2's complement and change the sign.

2.40 Find the 2's complement of (a) 110_2 and (b) 10101_2

Solution:

	Binary number	2's complement	
(a)	110_2	001_2	1's complement of 110_2
		$+ 1_2$	Add 1
		<u> </u>	
		010_2	2's complement of 110_2

$$\begin{array}{rcl}
 \text{(b)} & 10101_2 & 01010_2 \quad \text{1's complement of } 10101_2 \\
 & & + 1_2 \\
 & & \hline
 & & 01011_2 \quad \text{2's complement of } 10101_2
 \end{array}$$

2.41 Find the 2's complement of 1101011101000.*Solution:*

$$\begin{array}{rcl}
 1101011101000 & & \\
 \updownarrow \quad \updownarrow & & \left\{ \begin{array}{l} \text{First 1 going right-to-left.} \\ \text{These bits remain as} \\ \text{they were.} \end{array} \right. \\
 0010100011000 & & \text{2's complement}
 \end{array}$$

1's complement of original bits.

2.42 Subtract 1011_2 from 1100_2 using the 2's complement method. Also show direct subtraction for comparison.*Solution:*

Direct subtraction	2's complement method	
1100_2	1100_2	
-1011_2	$+0101_2$	2's complement
$\hline 0001_2$	$\cancel{1}0001_2$	Discard carry
	$\rightarrow 0001_2$	

2.43 Subtract 11100_2 from 10011_2 using the 2's complement method. Also show direct subtraction for comparison.*Solution:*

Direct subtraction	2's complement method	
10011_2	10011_2	
-11100_2	$+00100_2$	2's complement
$\hline -01001_2$	$\rightarrow 10111_2$	No carry 2's complement of answer.
	$\rightarrow -01001_2$	

DECIMAL-TO-BINARY CONVERSION

Decimal-to-binary conversion is a more lengthy process. One method is the *repeated subtraction of powers of two* until there is no remainder or the remainder that is left is sufficiently small for the desired conversion. This method works best for *whole numbers*. The second method, *two-part method*, is used to convert a number containing a *fraction*. First the whole number is converted by successively dividing the number by two. Then the decimal part is converted by repeatedly multiplying by two until the decimal part of the number becomes exactly zero or until the desired accuracy is obtained.

2.44 How will you perform a decimal-to-binary conversion using the subtraction method?*Solution:*

1. Write down the powers of *upto and including* the one closest to the number being converted.
2. *Sequentially subtract* each of these powers of two from the decimal number.
3. If the powers can be subtracted, *write a binary 1* in the bit position corresponding to that power of two.
4. If the power is larger than the decimal number, *write a 0* in that position and try the next lower power of two.

2.45 Convert 50_{10} to binary using the subtraction method.*Solution:*

	32	16	8	4	2	1
	1	1	0	0	1	0
50	↑	↑			↑	
-32						
$\hline 18$						
-16						
$\hline 2$						
-2						
$\hline 0$						

$50_{10} = 110010_2$

2.46 How will you perform a decimal-to-binary conversion using the two-part method?*Solution:*

First convert the *whole part* of the decimal number by repeatedly dividing by two.

1. If there is no remainder (it divides evenly or is *even*) write down a 0.
2. If there is a remainder (answer is odd) write down a 1.
3. The first division represents the *least significant digit*.
4. The last division represents the *most significant digit*.

This method does not require writing down the powers of two. The fractional part is converted by repeatedly multiplying by two.

1. If the multiplication produces a whole number equal to one plus some fractional part, write a 1.
2. Multiplications by two are repeated until the fractional part exactly equals zero or until the desired accuracy is obtained.

2.47 Convert 53_{10} to binary by two-part method.*Solution:*

Division	Remainder	
$53 \div 2 = 26$	1	→ LSD
$26 \div 2 = 13$	0	
$13 \div 2 = 6$	1	
$6 \div 2 = 3$	0	
$3 \div 2 = 1$	1	
$1 \div 2 = 0$	1	→ MSD

$53_{10} = 110101$

2.48 Convert 0.3125_{10} to binary.*Solution:*

Multiplication	Whole Number Part	
0.3125		
$\times 2$		
0.6250	0	→ MSD
$\times 2$		
1.2500	1	
$\times 2$		
0.5000	0	
$\times 2$		
1.0000	1	→ LSD

$0.3125_{10} = 0.0101_2$

SIGNED BINARY NUMBERS

Binary numbers that carry identification as to their polarity are called signed binary numbers. Plus and minus signs for positive and negative numbers can be represented in a digital format. Further, if a binary number is negative, there are several convenient ways of representing that number. Each representation has its own features. The three major *signed binary notations* are: sign magnitude notation, 1's complement notation, and 2's complement notation.

2.49 Describe the sign magnitude notation for representing signed binary numbers.*Solution:*

The most straight forward method is to add a 0 or 1 to the most significant digit of the overall number. This notation is called *sign magnitude* because the *sign* bit is given first, and then the positive *magnitude* of the number follows. A *sign bit* of 0 indicates the number is positive, while a *sign bit* of 1 indicates the number is negative. All other bits of the number indicate the magnitude of the number just as for unsigned binary numbers.

Sign magnitude notation is *very easy to read*, but it is *not easy to use* when adding or subtracting binary numbers.

2.50 Express the decimal number 13 as a signed magnitude number.*Solution:*

$$13_{10} = 1101_2$$

Sign

$$+13_{10} = 01101_2$$

$$-13_{10} = 11101_2$$

2.51 Describe the 1's complement notation for representing signed binary numbers.*Solution:*

Another way to represent a signed binary number is to attach a *sign bit*, just as with sign magnitude notation, and *invert all of the bits if the number is negative*. 1's complement numbers are easy to form, but there are two representations of zero. Also, when 1's complement numbers are added or subtracted, a process known as *end-around-carry* is necessary to obtain the correct answer.

2.52 Express the decimal number 13 as signed magnitude number in 1's complement notation.*Solution:*

$$13_{10} = 1101_2$$

$$+13_{10} = 01101_2$$

$$-13_{10} = 10010_2$$

↑
Sign bit

same as in problem 2.50
invert all the bits if the number is negative.

This number representation is called 1's complement notation.

2.53 Describe the 2's complement notation of expressing signed magnitude numbers.*Solution:*

The most common representation for signed binary numbers is 2's complement notation. The 2's complement is generated by *inverting the bits* as in 1's complement, *then adding 1 to the least significant digit*. 2's complement is a little more difficult to generate, but it simplifies addition and subtraction. Further, there is only one representation for zero in 2's complement.

2.54 Express the decimal number 13 in 2's complement notation.*Solution:*

$$13_{10} = 1101_2$$

$$+13_{10} = 01101_2$$

$$-13_{10} = 10011_2$$

same as in problem 2.52

This number representation is called 2's complement notation.

OCTAL NUMBERS

The octal number system is *composed of eight digits*, which are

$$0, 1, 2, 3, 4, 5, 6, 7$$

To count above 7, we begin another column and start over.

$$10, 11, 12, 13, 14, 15, 16, 17$$

$$20, 21, 22, 23, 24, 25, 26, 27$$

$$30, 31, 32, 33, 34, 35, 36, 37$$

Counting in octal is the same as counting in decimal, except that any number with an 8 or 9 is omitted. We use the *subscript 8* to indicate on octal number.

OCTAL-TO-DECIMAL CONVERSION

Since the octal number system has a *base of 8*, each successive digit position is an *increasing power of 8*, beginning in the right-most column with 8^0 . The evaluation of an octal number in terms of its decimal equivalent is accomplished by multiplying each digit by its weight and summing the products.

2.55 Convert 2374_8 to its decimal equivalent.

Solution:

Weight	8^3	8^2	8^1	8^0
Decimal value	512	64	8	1
Octal number	2	3	7	4

$$\begin{aligned}
 2374_8 &= 2 \times 8^3 + 3 \times 8^2 + 7 \times 8^1 + 4 \times 8^0 \\
 &= 2 \times 512 + 3 \times 64 + 7 \times 8 + 4 \times 1 \\
 &= 1024 + 192 + 56 + 4 \\
 &= 1276_{10}
 \end{aligned}$$

DECIMAL-TO-OCTAL CONVERSION

Whole octal numbers are represented by digits to the left of the octal point. The *column weights* are

$$8^n \dots 8^6 \cdot 8^5 \cdot 8^4 \cdot 8^3 \cdot 8^2 \cdot 8^1 \cdot 8^0$$

A method of converting a decimal number into an octal number is the *repeated-division-by-8 method*, which is similar to the method used for conversion of decimal to binary. Each successive division by 8 yields a remainder that becomes a digit in the equivalent octal number. The first remainder generated is the *least significant digit (LSD)*.

2.56 Convert 359 decimal to octal.

Solution:

44	Remainder	
8 $\overline{) 359}$		
32		
39		
32		
7	→ 7	(LSD)
5		
8 $\overline{) 44}$		
40		
4	→ 4	
0		
8 $\overline{) 5}$		
0		
5	→ 5	(MSD)
		547_8

FRACTIONAL OCTAL NUMBERS

Fractional octal numbers are represented by digits to the right of octal point. The column weights are:

$$8^0 \cdot 8^{-1} \cdot 8^{-2} \cdot 8^{-3} \cdot 8^{-4} \cdot 8^{-5} \cdot 8^{-6} \dots 8^{-n}$$

↓
octal point

All digits to the left of octal point have weights that are *positive powers of eight*. All digits to the right of octal point have weights that are *negative powers of eight*.

To convert fractional octal numbers to decimal numbers determine the weights of each digit and *sum the weight times the digit*.

2.57 Determine the decimal value of the octal fraction 0.325_8 .

Solution:

First determine the weights of each digit and then sum the weight times the digit.

Octal weight:	8^{-1}	8^{-2}	8^{-3}
Decimal value:	0.125	0.015625	0.001953
Octal number:	0.3	2	5

$$\begin{aligned}
 0.325_8 &= 3(0.125) + 2(0.015625) + 5(0.001953) \\
 &= 0.375 + 0.03125 + 0.009765 \\
 &= 0.416015_{10}
 \end{aligned}$$

OCTAL-TO-BINARY CONVERSION

The conversion from octal to binary is performed by converting each octal digit to its *3-bit binary equivalent*.

Table 2.1 Octal Numerals

Octal Digit	0	1	2	3	4	5	6	7
Binary Equivalent	000	001	010	011	100	101	110	111

Using these conversions, any octal number is converted to binary by *individually* converting each digit.

2.58 Convert 472_8 to binary.

Solution:

4	7	2_8
100	111	010

Hence octal $472 = 100111010_2$

2.59 Convert 5431_8 to binary.

Solution:

5	4	3	1
101	100	011	001

Thus, $5431_8 = 101100011001_2$

BINARY-TO-OCTAL CONVERSION

Converting from binary to octal is the *reverse* of octal-to-binary conversion. The bits of binary numbers are arranged into *groups of three bits* starting at the least

significant bit. Then each group is converted to its equivalent, as shown in Table 2.1.

2.60 Convert 100111010₂ to octal.

Solution:

$$\begin{array}{ccc} \underbrace{100} & \underbrace{111} & \underbrace{010} \\ 4 & 7 & 2_8 \\ & = 472_8 \end{array}$$

2.61 Convert 11010110₂ to octal.

Solution:

$$\begin{array}{ccc} \underbrace{011} & \underbrace{010} & \underbrace{110} \\ 3 & 2 & 6_8 \\ & = 326_8 \end{array}$$

2.62 Why do we need the octal system?

Solution:

When dealing with a large quantity of binary numbers of many bits, it is *convenient and more efficient* for us to write the numbers in octal rather than binary. *Digital circuits and systems work strictly in binary; we are using octal only as a convenience for the operators of the system.*

2.63 Discuss the usefulness of the octal system.

Solution:

The octal system is attractive as *a shorthand means of expressing large binary numbers*. In computer work, binary numbers with up to 64 bits are not uncommon. These binary numbers do not always represent a *numerical* quantity but are always some type of *code* that conveys *non-numerical* information. In computers, binary numbers might represent:

1. actual non-numerical data;
2. numbers corresponding to a location (address) in memory;
3. an instruction code;
4. a code representing alphabetic and other non-numerical characters; or
5. a group of bits representing the status of devices internal or external to the computer.

HEXADECIMAL NUMBERS

Hexadecimal (*hex*) uses sixteen different digits. Because hex digits must be represented by a single character; *letters are chosen to represent values greater than 9*. The sixteen allowable hex digits are:

0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F.

To signify a hex number, a subscript 16 or the letter H is used (that is A7₁₆ or A7H, for example). Two hex digits are used to represent a *byte*. Four bits (one hex digit) are sometimes called a *nibble*.

2.64 What is the *practical application* of hexadecimal number system?

Solution:

Binary numbers are long. *The hexadecimal system was born out of the need to express numbers concisely, and is by far the most commonly used number system in computer literature.*

The hexadecimal number system, like the octal system, is *a method of grouping bits to simplify entering and reading the instructions or data present in digital computer systems*. Hexadecimal uses *4-bit groupings*, therefore, instructions or data used in 8-16-, or 32-bit computer systems can be represented as a two-, four-, or eight-, digit hexadecimal code instead of using a long string of binary digits.

2.65 How do we count in hex?

Solution:

Once we get to F, simply start over with another column and continue as follows:

10, 11, 12, 13, 14, 15, 16, 17, 18, 19,
1A, 1B, 1C, 1D, 1E, 1F
20, 21, 22, 23, 24, 25, 26, 27, 28, 29,
2A, 2B, 2C, 2D, 2E, 2F
30, 31, 32, 33, 34, 35, 36, 37, 38, 39,
3A, 3B, 3C, 3D, 3E, 3F
40, 41, and so forth.

2.66 Tabulate and *compare* binary, hexadecimal, and decimal numbers.

Solution:

Table 2.2 Binary, Hex, and Decimal Numbers—a Comparison

Binary	Hexadecimal	Decimal
0000	0	0
0001	1	1
0010	2	2
0011	3	3
0100	4	4
0101	5	5
0110	6	6
0111	7	7
1000	8	8
1001	9	9
1010	A	10
1011	B	11
1100	C	12
1101	D	13
1110	E	14
1111	F	15
1 0000	10	16

2.67 What is the maximum count of a two-digit, three-digit, and four-digit hex number?

Solution:

With two hexadecimal digits, FF₁₆, we can count up to 255₁₀(16² – 1). To count beyond this, three hexadeci-

mal digits are required. The maximum three-digit hexadecimal number is FFF_{16} , or 4095_{10} ($16^3 - 1$). The maximum four-digit hexadecimal number is FFFF_{16} , or $65,535_{10}$ ($16^4 - 1$).

HEXADECIMAL-TO-DECIMAL CONVERSION

One method to evaluate a hexadecimal number in terms of its decimal equivalent is to first convert it to *binary* and then convert from *binary* to *decimal*.

Another method of converting a hexadecimal number to its decimal equivalent is by multiplying each hexadecimal digit by its *weight* and then taking the sum of these products. *The weights of a hexadecimal number are increasing powers of 16.*

2.68 Convert $2\text{A}6_{16}$ to decimal.

Solution:

$$\begin{array}{ccc} 2 & \text{A} & 6 \\ 0010 & 1010 & 0110 \\ 001010100110 = 2^1 + 2^2 + 2^5 + 2^7 + 2^9 \\ = 2 + 4 + 32 + 128 + 512 \\ = 678_{10} \end{array}$$

DECIMAL-TO-HEXADECIMAL CONVERSION

Repeated division of a decimal number by 16 will produce the equivalent hexadecimal number formed by the remainders of each division. This is analogous to repeated division-by-2 for decimal-to-binary conversion and repeated division-by-8 for decimal-to-octal conversion.

2.69 Convert $\text{B}2\text{F}8_{16}$ to decimal.

Solution:

$$\begin{aligned} \text{B}2\text{F}8_{16} &= \text{B} \times 16^3 + 2 \times 16^2 + \text{F} \times 16^1 + 8 \times 16^0 \\ &= 11 \times 4096 + 2 \times 256 + 15 \times 16 + 8 \times 1 \\ &= 45,816_{10} \end{aligned}$$

2.70 Convert 498_{10} to hexadecimal.

$$\begin{array}{llll} 498 \div 16 = 31 & \text{remainder } 2 & & (\text{LSD}) \\ 31 \div 16 = 1 & \text{remainder } 15 (= \text{F}) & & \\ 1 \div 16 = 0 & \text{remainder } 1 & & (\text{MSD}) \\ 498_{10} = 1\text{F}2_{16} & & & \end{array}$$

HEXADECIMAL-TO-BINARY CONVERSION

To convert from hex to binary *replace each hex symbol with the appropriate four bits.*

2.71 Convert $\text{A}9_{16}$ to binary.

Solution:

$$\begin{array}{cc} \text{A} & 9 \\ 1010 & 1001 \\ 10101001_{16} & \end{array}$$

BINARY-TO-HEXADECIMAL CONVERSION

To convert a binary number to hex, simply *break the binary number into four-bit groups, starting at the binary point, and replace each group with the equivalent hex symbol.*

2.72 Convert 01101101_2 to hex.

Solution:

$$\begin{array}{cc} 0110 & 1101 \\ 6 & \text{D} \\ & 6\text{D}_{16} \end{array}$$

HEXADECIMAL ARITHMETIC

Hexadecimal *addition* can be done directly with hex numbers remembering that:

1. *hex* digits 0 through 9 are equivalent to *decimal* digits 0 through 9, and
2. the *hex* digits A through F are equivalent to *decimal* digits 10 through 15.

Since a hexadecimal number can be used to represent a binary number, it can also be used to represent the 2's complement of a binary number. *The 2's complement allows us to subtract by adding binary numbers.* We can also use this method for hexadecimal subtraction.

2.73 What are the rules for hexadecimal addition?

Solution:

When adding two hexadecimal numbers, the following rules apply:

1. In any given column of an addition problem, think of the two hexadecimal digits in terms of their decimal value.
2. If the sum of these two digits is 15_{10} or less, bring down the corresponding hexadecimal digit.
3. If the sum of these two digits is greater than 15_{10} , bring down the amount of the sum that exceeds 16_{10} and *carry* a one to the next column.

2.74 Add the following hexadecimal numbers:

$$\begin{array}{ll} \text{(a) } 58_{16} + 22_{16} & \text{(b) } 2\text{B}_{16} + 84_{16} \\ \text{(c) } \text{DF}_{16} + \text{AC}_{16} & \end{array}$$

Solution:

$$\begin{array}{ll} \text{(a)} & \begin{array}{r} 58_{16} \\ + 22_{16} \\ \hline 7\text{A}_{16} \end{array} & \begin{array}{l} \text{right column: } 8_{16} + 2_{16} \\ = 8_{10} + 2_{10} = 10_{10} = \text{A}_{16} \\ \text{left column: } 5_{16} + 2_{16} \\ = 5_{10} + 2_{10} = 7_{10} = 7_{16} \end{array} \\ \text{(b)} & \begin{array}{r} 2\text{B}_{16} \\ + 84_{16} \\ \hline \text{AF}_{16} \end{array} & \begin{array}{l} \text{right column: } \text{B}_{16} + 4_{16} \\ = 11_{10} + 4_{10} = 15_{10} = \text{F}_{16} \\ \text{left column: } 2_{16} + 8_{16} \\ = 2_{10} + 8_{10} = 10_{10} = \text{A}_{16} \end{array} \end{array}$$

(c) DF_{16} $+AC_{16}$ <hr style="width: 50px; margin-left: 0;"/> $18B_{16}$	right column: $F_{16} + C_{16}$ $= 15_{10} + 12_{10} = 27_{10}$ $27_{10} - 16_{10} = 11_{10} = B_{16}$ <i>with a 1 carry</i> left column: $D_{16} + A_{16} + 1_{16}$ $= 13_{10} + 10_{10} + 1_{10} = 24_{10}$ $24_{10} - 16_{10} = 8_{10} = 8_{16}$ <i>with a 1 carry</i>
--	--

2.75 Subtract the following hex numbers:

(a) $84_{16} - 2A_{16}$ (b) $C3_{16} - 0B_{16}$

Solution:

(a) $2A_{16} = 00101010_2$
 2's complement of $2A_{16} = 11010110 = D6_{16}$

84_{16} $+D6_{16}$ <hr style="width: 50px; margin-left: 0;"/> $15A_{16}$ $5A_{16}$	Drop carry as in 2's complement addition.
---	--

(b) $0B_{16} = 00001011_2$
 2's complement of $0B_{16} = 11110101 = F5_{16}$

$C3_{16}$ $+F5_{16}$ <hr style="width: 50px; margin-left: 0;"/> $1B8_{16}$	Drop carry
--	------------

CONVERSION ALGORITHMS

An *algorithm* is a method of arriving at an answer that always works. The first algorithm presented in this chapter is:

$$Y = d_n r^n + d_{n-1} r^{n-1} + \dots + d_0 r^0 + d_{-1} r^{-1} + \dots + d_{-m} r^{-m}$$

The second algorithm requires successive division for integers. Both algorithms are universal. A *flow chart* is a graphical means of expressing an algorithm.

2.76 Illustrate and explain the conversion algorithm.

Solution:

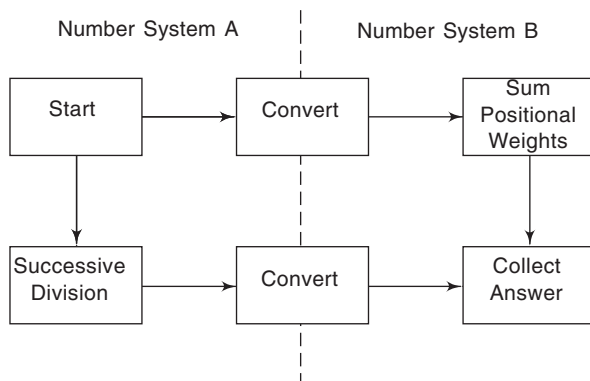


Fig. 2.8 Conversion Algorithm.

Numbers from number system A are first converted to those of the system B. Then calculation takes place within the system B. For example, to convert from the *binary system to the decimal system* each digit (a 1 or 0) is converted to decimal and then multiplied by the radix (expressed in the decimal system) raised to some decimal power. The operations could be described as: (a) *convert* (b) *arithmetic* and (c) *collect the answer*.

Decimal-to-octal conversion is usually done using *successive division*, because it is easier to do the arithmetic in the decimal system. If, however, the arithmetic is done in octal, the *positional weighting* method can be used.

Octal-to-decimal conversion is usually done using the *positional weighting* method. However, if the arithmetic is done in octal, the same answer can be found using *successive division*.

2.77 Draw the flowchart for repeated-division method of decimal-to-binary conversion of integers.

Solution:

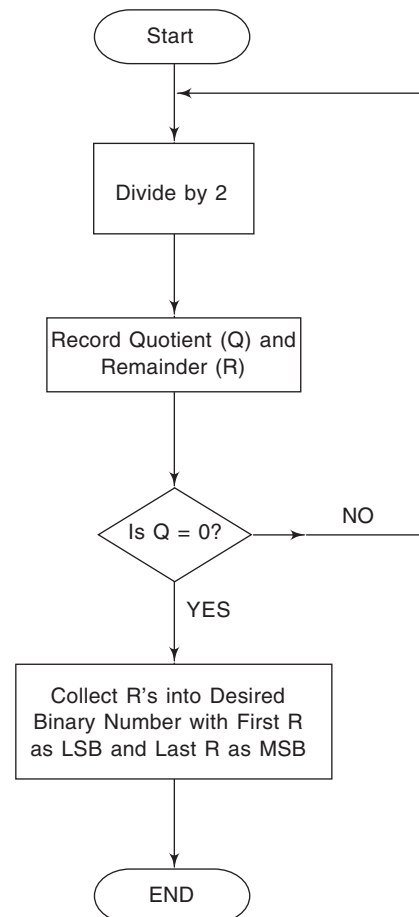


Fig. 2.9 Flowchart for Repeated-division Method of Decimal-to-binary Conversion of Integers. The Same Process can be used to Convert a Decimal Integer to any Other System.

FLOATING-POINT NUMBERS

In *decimal* very small and very large numbers are expressed in *scientific notation* (powers of 10). 2.7×10^9 and 1.56×10^{-12} are typical examples. *Binary numbers* can also be expressed in the same notation by stating a number (*mantissa*) and an *exponent of 2*. There are many *formats* of the floating-point number, each computer having its own. On some machines, the programmer can *select* from various formats, depending on the *accuracy* desired. Some use *excess-notation* for the exponent; some use *2's complement*; some use *signed magnitude* for both the mantissa and the exponent.

2.78 Differentiate between a fixed-point number and floating-point number.

Solution:

Any number with a *fixed location of the radix point* is called a fixed-point number, e.g. 01101.111_2 , 66.575_{10} , and $9AF. AB_{16}$. The *range* of numbers that can be represented in fixed-point notation is *limited*.

Floating-point notation is a method of expressing very small and very large numbers in (scientific notation) the form $Y = N r^p$, where N is the *mantissa*, r the *base* of the number system, and p is the *exponent* or power to which r is raised. $Y = 57.75 = 0.5775 \times 10^2$ is a typical example where $N = 0.5775$, $r = 10$, and $p = 2$.

2.79 What are the limitations of fixed point numbers?

Solution:

The first drawback of this scheme is the need of the user *to remember and keep track of the decimal point location*. The second drawback of this scheme is that the *range of numbers* which can be represented, using this scheme is limited to 999.999.

2.80 Describe the different formats with reference to binary floating-point numbers.

Solution:

Precision is the degree to which the *correctness* of a quantity is expressed. Single-precision, double-precision and extended-precision binary floating-point numbers have the same basic formats except for the number of bits. *Single-precision* floating-point numbers have 32 bits, *double-precision* numbers have 64 bits, and *extended-precision* numbers have 80 bits.

2.81 Illustrate and describe the floating-point format for one computer.

Solution:

The format for one computer is given in Fig. 2.10. In this machine the word consists of two parts: a 10-bit *mantissa*, and a 6-bit *exponent*. The mantissa is in *2's complement form*; the left-most bit is the *sign bit*. The *binary point* is to the right of this sign bit. The 6 bits of the exponent could represent 0 through 63. However, to express *negative exponents* the number 32 (100000_2) has been added to the desired exponent. This is a

common system used in floating-point formats. It represents *excess-32 notation*.

Mantissa	Exponent
0 1 0 0 1 0 0 0 0 0	1 0 0 0 1 1

Fig. 2.10 Floating-point Format.

2.82 What is the floating-point number in Fig. 2.10?

Solution:

Excess-32 notation is given in Table 2.3.

The mantissa portion is $+0.100100000$

The exponent portion is 100011

Subtracting $100000(32_{10})$ 000011

The entire number is $+0.1001_2 \times 2^3$

$N = 100.1_2$

$= 4.5_{10}$

Table 2.3 Excess-32 Notation

Desired Exponent	Binary Representation
-32	000000
-1	011111
0	100000
+6	100110
+31	111111

2.83 Illustrate and describe the format for single-precision binary floating-point numbers with the help of an example.

Solution:

In the standard format for a single-precision binary number, Fig. 2.11, the *sign bit* (S) is the left-most bit, the *exponent* (E) includes the next eight bits and the *mantissa* or the *fractional part* (F) includes the remaining 23 bits.

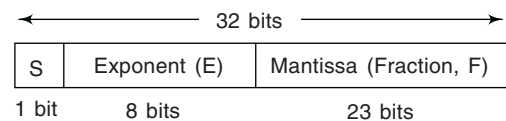


Fig. 2.11 Single-precision Binary Floating Point Numbers.

In the mantissa, the *binary point* is understood to be to the left of the 23 bits. Effectively, there are 24 bits in the mantissa because in any binary number the left-most (most significant) bit is always a 1. Therefore, this 1 is understood to be there although it does not occupy an actual bit position.

The eight bits in the exponent represent a *biased exponent*, which is obtained by adding 127 to the actual exponent. The purpose of the *bias* is to allow very large or very small numbers without requiring a separate sign bit for the exponents. The biased exponent allows a range of actual exponent values from -126 to $+128$. Let's use 1011010010001 as an example.

It can be expressed as 1 plus a fractional binary number by moving the binary point 12 places to the left and then multiplying by the appropriate power of 2.

$$1011010010001 = 1.011010010001 \times 2^{12}$$

Assuming that this is a positive number, the *sign bit* (S) is 0. The exponent, 12 is expressed as a *biased exponent* by adding it to 127 ($12 + 127 = 139$). The biased exponent (E) is expressed as the binary number 10001011. The *mantissa* is the fractional part (F) of the binary number 0.011010010001. Because there is always 1 to the left of the binary point in the power-of-two expression, *it is not included in the mantissa*. The complete floating-point number is

S	E	F
0	10001011	011010010001000000000000

Fig. 2.12 The Format for 1011010010001.

2.84 When is a floating-point number said to be *normalized*?

Solution:

In a 16-bit format (see Fig. 2.10), the *mantissa* is 10 bits long and the *exponent* 6-bits. The system is, therefore, capable of 9-bit *accuracy* (allowing 1 bit for the sign). *The exponent bits add, nothing to accuracy, only to magnitude*. Fixed-point 2's complement numbers expressed in 16 bits are accurate to 15 bits. Thus, although floating-point numbers can be much larger or smaller than equivalent length fixed-point numbers they are *less accurate*.

To keep the full 10 bits of accuracy in the floating-point number *the most significant bit must be placed next to the sign bit*. Under these conditions the floating-point number is said to be *normalized*. Most computers normalize the result of floating-point operations.

SUMMARY

- Digital systems require the use of decimal, binary, octal, and hexadecimal number systems.
- The binary system is used by the hardware; the octal and hexadecimal systems are used as a shorthand language for representing binary numbers.
- A binary number is a weighted number. The weight of each whole number digit is a positive power of two. The weight of each fractional digit is a negative power of two. The whole number weights increase from right to left—from least significant digit to most significant digit.
- A binary number can be converted to a decimal number by summing the decimal values of the weights of all the 1's in the binary number.
- A decimal whole number can be converted to binary by using the sum-of-weights or by repeated division-by-2 method.
- A decimal fraction can be converted to binary by using the sum-of-weights or by repeated multiplication-by-2 method.
- The basic rules for binary addition are:

$$0 + 0 = 0 \quad 0 + 1 = 1 \quad 1 + 0 = 1 \quad 1 + 1 = 10$$
- The basic rules for binary subtraction are:

$$0 - 0 = 0 \quad 1 - 1 = 0 \quad 1 - 0 = 1 \quad 10 - 1 = 1$$
- The 1's complement of a binary number is derived by changing 1's to 0's and 0's to 1's.
- The 2's complement of a binary number can be derived by adding 1 to the 1's complement.
- Binary subtraction can be accomplished with addition by using the 1's or 2's complement method.
- A positive binary number is represented by a 0 sign bit.
- A negative binary number is represented by a 1 sign bit.
- For arithmetic operations, negative binary numbers are represented in 1's complement or 2's complement form.
- The hexadecimal number system consists of 16 digits and characters, 0 through 9 followed by A through F.
- One hexadecimal digit represents a 4-bit binary number.
- A decimal number can be converted to hexadecimal by the repeated division-by-16 method.
- The octal number consists of eight digits, 0 through 7.
- A decimal number can be converted to octal by the repeated division-by-8 method.
- Octal-to-binary conversion is accomplished by simply replacing each octal digit by its 3-bit binary equivalent. The process is reversed for binary-to-octal conversion.
- Each of the systems uses the two basic conversion algorithms, and each can perform all the arithmetic operations.
- Floating point numbers provide the widest range of binary numbers.

Test your
understanding

REVIEW QUESTIONS

1. What is the largest decimal number that can be represented in binary with eight bits?
2. What weight does the digit 7 have in 58.72?
3. How many bits are required to count up to decimal 1 million?
4. Draw the table of hexadecimal and binary equivalences for 0 through 16_{10} .
5. Convert decimal 23 to binary.
6. Convert 1011101001_2 to decimal.
7. Tabulate binary and decimal positional weighting.
8. Write the next three numbers in the octal counting sequence: 624, 625, 626.
9. What range of decimal values can be represented by a four-digit octal number?
10. Convert 614_8 to decimal.
11. Convert 146_{10} to octal, then from octal to binary.
12. What range of decimal values can be represented by a four-digit hex number?
13. What are the advantages of using the hexadecimal system over binary, octal, or decimal system?
14. Convert $24CE_{16}$ to decimal.
15. Convert 3117_{10} to hex, then from hex to binary.
16. Write the next four numbers in this hex counting sequence: E9A, E9B, E9C, E9D.
17. Solve the following equation for x :

$$x_{16} = 1111 \quad 1111 \quad 1111 \quad 1111_2$$
18. Convert hexadecimal 7E to its decimal equivalent.
19. Convert decimal 141 to hexadecimal.
20. Add 1011_2 and 110_2 .
21. Subtract 1 from 100_2 .

Test your
understanding

SUPPLEMENTARY PROBLEMS

22. Express each of the following numbers as a sum of the products of each digit and its appropriate weight:
 (a) 51_{10} (b) 173_{10} (c) 1492_{10} (d) 10.658_{10}
23. How are the weighting factors determined for each decimal position in a base 10 number?
24. Convert 1101.0110_2 to decimal.
25. What is the weight of the MSB of a 16-bit number?
26. What is the decimal value of 110110_2 ?
27. Derive the 9's complement representation of
 (a) 61_{10} (b) 235_{10}
28. Evaluate $9_{10} - 6_{10}$ using the binary 1's complement system.
29. Draw the diagram to show all the 4-bit binary combinations in the 2's complement system to represent positive and negative numbers from 0 to 8.
30. Obtain the 1's and 2's complements of the following numbers:
 (a) 101101_2 (b) 101100_2
31. Convert 41_{10} to binary.
32. Convert 170_{10} to binary.
33. Convert 614_8 to decimal.
34. Convert 146_{10} to octal, then from octal to binary.
35. Convert 975_{10} to binary by first converting to octal.
36. Convert binary 1010111011_2 to decimal by first converting to octal.
37. Convert 11110011_2 to hexadecimal system.
38. Convert $F80B_{16}$ to binary.
39. Add $18_{16} + 34_{16}$.
40. Subtract $5C_{16}$ from 94_{16} .
41. How will you evaluate a binary number that is already in floating-point format?
42. Convert the decimal number 3.248×10^4 to a single-precision floating-point binary number.

Test your
understanding

OBJECTIVE TYPE QUESTIONS

Fill in the Blanks

43. Large numbers in the binary system become _____ in length.
44. There are various number systems in use which conveniently _____ with the binary system.
45. It is not necessary to design _____ circuits to perform binary arithmetic.
46. If an application calls for extensive mathematics, it is the domain of a _____.
47. Any number of any magnitude can be expressed by using the system of _____ weighting.
48. The position of a digit with reference to the _____ point determines its weight.
49. Digital computers function in the _____ number system.
50. Any number assigning process is called _____.
51. The highest weighting value is on the _____ left.
52. Position weighting values increase from _____ to _____.
53. The zeros to the right of the highest valued 1 serve as _____ to retain the 1's in their correct positions.
54. The least significant digit is the _____ right digit.
55. The most significant digit is multiplied by the highest _____ value.
56. With n bits we can count up to _____.
57. Binary digits are referred to as _____.
58. The radix in any number system equals the number of _____ in the system.
59. Three number systems are used in computer work: binary, _____, and hexadecimal.
60. The binary number system is sometimes called the _____ system.
61. The 1 in the binary number 1000 has a place value of _____ in decimal.
62. The binary number 1010 equals _____ in decimal.
63. The number 2^7 equals _____ in decimal.
64. When subtracting numbers, we sometimes have to _____ from the next higher column.
65. The 1's complement method of subtraction is particularly useful in _____ circuits.
66. The decimal number 39 equals _____ in binary.
67. The binary number 101010 equals _____ in decimal.
68. Signed binary numbers carry _____ as to their polarity.
69. There are _____ representations of zero in 1's complement notation.
70. The octal number system is composed of _____ digits.
71. The most common representation for signed binary numbers is _____ notation.
72. There is only one representation for _____ in 2's complement notation.
73. The octal system is a _____ means of expressing large binary numbers.
74. Hexadecimal uses _____ groupings.
75. In hexadecimal system _____ are chosen to represent values greater than 9.
76. Four bits (one hex digit) are sometimes called a _____.
77. Any number system is only a set of _____ symbols.
78. The weights of a hexadecimal number are increasing powers of _____.
79. The 2' complement allows us to subtract by _____ binary numbers.
80. Write the next four numbers in this hex counting sequence:
E9A, E9B, E9C, E9D, _____, _____, _____, _____.
81. In decimal, very small and very large numbers are expressed by _____ notation.
82. An algorithm is a method of arriving at an answer that _____ works.
83. Conversion from decimal-to-octal is usually done by _____ division.
84. Conversion from octal-to-decimal is usually done using the _____ weightage.
85. There are many _____ for floating-point numbers.
86. Floating point numbers have the advantage of expressing _____, _____ and _____, _____ numbers.
87. Floating-point numbers are less accurate than _____ numbers.

True/False Questions

State whether the following statements are True or False.

88. Binary number are to the base 2.
89. A radix of 10 gives 10 different states.
90. If an application calls for extensive mathematics, it is the domain of an encoder.
91. The most significant digit is the right-most digit.
92. We have names for the weighting of digits in their various places of significance.
93. Larger numbers cannot be expressed by positional weightage.
94. The position of a digit with reference to radix point determines its weight.
95. A raised finger can represent a 1 and a 0 can be represented by a lowered finger.
96. Octal numbers are to base 16.
97. $1 + 1 + 1 = 11_2 \rightarrow 1$ with a carry of 1.
98. When subtracting numbers, we sometimes have to borrow from the next higher column.
99. The complement number system was invented to make addition and subtraction faster and easier.
100. The 2' complement of a binary number is obtained by subtracting 1 from the 1's complement.
101. The method of repeated subtraction of powers of two works best for fractional binary numbers.
102. In sign magnitude notation, a sign bit of 1 indicates that the number is positive.
103. Sign magnitude notation is easy to use but not easy to read when adding and subtracting binary numbers.
104. There is only one representation of zero in 2's complement notation.
105. The primary advantage of hex number system is the ease with which conversions can be made.
106. In a hex system letters are chosen to represent values greater than 9.
107. An algorithm is a method of arriving at an answer that sometimes works.

Multiple Choice Questions

108. The number system with radix 2 is known as
 - (a) binary number system
 - (b) octal number system
 - (c) decimal number system
 - (d) hexadecimal number system
109. A group of 8 bits is known as
 - (a) a nibble
 - (b) a byte
 - (c) a bit
 - (d) an octal number
110. Knowledge of binary number system is required by the designers of computers and other digital systems because
 - (a) it is easy to learn binary number system
 - (b) it is easy to learn Boolean algebra
 - (c) it is easy to use binary codes
 - (d) the devices used in these systems operate in a binary manner
111. One's complement of binary number 10001011 is
 - (a) 01110101
 - (b) 01110111
 - (c) 01110100
 - (d) 11110100
112. One's complement of 1's complement of the binary number 11000101 is
 - (a) 00111010
 - (b) 10111010
 - (c) 00111011
 - (d) 11000101
113. Two's complement of the binary number 10010100 is
 - (a) 01101011
 - (b) 01101100
 - (c) 11101100
 - (d) 10001011
114. Two's complement of 2's complement of the binary number 01101100 is
 - (a) 10010100
 - (b) 01101100
 - (c) 10010011
 - (d) 11101100
115. The radix of hexadecimal system is
 - (a) 2
 - (b) 4
 - (c) 8
 - (d) 16
116. Hexadecimal number system is in digital computers and digital systems to
 - (a) perform arithmetic operations
 - (b) input binary data into the system
 - (c) perform logical operations
 - (d) perform arithmetic and logic operations
117. The hexadecimal equivalent of the binary number 11101101111010 is
 - (a) EDEB
 - (b) 35572
 - (c) FB7A
 - (d) 3B7A

118. The binary equivalent of the hexadecimal number AOB5 is
 (a) 0101110100001010 (b) 0101111101001010
 (c) 1010000010110101 (d) 1011000011000101
119. The binary equivalent of octal number 362 is
 (a) 011110010 (b) 101101010
 (c) 001101100010 (d) 100001101
120. The octal equivalent of the binary number 11010111 is
 (a) 656 (b) 327 (c) 653 (d) D7
121. In a digital computer binary subtraction is performed
 (a) in the same way as we perform subtraction in decimal number system
 (b) using two's complement method
 (c) using 9's complement method
 (d) using 10's complement method
122. The binary equivalent of decimal number 0.0625 is
 (a) 1001110001 (b) 0.1001110001
 (c) 0.0110001110 (d) 0.0001
123. The maximum positive and negative numbers which can be represented in 2's complement form using n bits are respectively
 (a) $+(2^{n-1} - 1), -(2^{n-1} - 1)$ (b) $+(2^{n-1} - 1), -2^{n-1}$
 (c) $+2^{n-1}, -2^{n-1}$ (d) $+2^{n-1}, -(2^{n-1} + 1)$
124. When two n -bit binary numbers are added, the sum will contain at the most
 (a) n bits (b) $n + 1$ bits
 (c) $n + 2$ bits (d) $n + n$ bits
125. The minimum number of bits required to represent negative numbers in the range of -1 to -9 using 2's complement representation is
 (a) 2 (b) 3 (c) 4 (d) 5
126. The minimum number of bits required to represent positive numbers in the range 1–31 using 2's complement representation is
 (a) 5 (b) 6 (c) 8 (d) 10
127. Two's complement representation of -8 is
 (a) 0111 (b) 1000 (c) 01000 (d) 10100
128. The number of bits required to encode 80 pieces of information is
 (a) 4 (b) 5 (c) 6 (d) 7

ANSWERS

1. $2^8 - 1 = 255$, 2. 0.1 3. 20 bits, 4. see problem 2.66,
 5. 10111, 6. 745_{10}
 7. **Table** Binary and Decimal Positional Weighting

(a) Binary	(b) Decimal
$1_2 = 1 \times 2^0 = 1_{10}$	$1_{10} = 1 \times 10^0 = 1_{10}$
$10_2 = 1 \times 2^1 = 2_{10}$	$10_{10} = 1 \times 10^1 = 10_{10}$
$100_2 = 1 \times 2^2 = 4_{10}$	$100_{10} = 1 \times 10^2 = 100_{10}$
$1000_2 = 1 \times 2^3 = 8_{10}$	$1000_{10} = 1 \times 10^3 = 1000_{10}$
$10000_2 = 1 \times 2^4 = 16_{10}$	$10000_{10} = 1 \times 10^4 = 10000_{10}$

8. 627, 630, 631 9. 0 to 4095 10. 396 11. 222, 010010010
 12. 0 to 65, 535
 13. It is most compact among the four systems and requires the fewest digit symbols,
 14. 9422 15. C2D, 110000101101 16. E9E, E9F, EA0, EA1
 17. $x = \text{FFFF}_{16}$ 18. 126 19. 8D 20. 10001_2
 21. 11_2

22. (a) $5 \times 10^1 + 1 \times 10^0$
 (b) $1 \times 10^2 + 7 \times 10^1 + 3 \times 10^0$
 (c) $1 \times 10^3 + 4 \times 10^2 + 9 \times 10^1 + 2 \times 10^0$
 (d) $1 \times 10^1 + 0 \times 10^0 + 6 \times 10^{-1} + 5 \times 10^{-2} + 8 \times 10^{-3}$
23. The form of every decimal number is
 $\dots 10^4 \ 10^3 \ 10^2 \ 10^1 \ 10^0 \ . \ 10^{-1} \ 10^{-2} \ 10^{-3} \ 10^{-4} \dots$

↓
decimal point

24. 13.375_{10} 25. 32768_{10} 26. 54_{10}
 27. (a) 38_{10} (b) 764_{10} 28. $0011_2; 3_{10}$

29. **Table Two's Complement Numbers**

Binary	Decimal
0111	+7
0110	+6
0101	+5
0100	+4
0011	+3
0010	+2
0001	+1
0000	0
1111	-1
1110	-2
1101	-3
1100	-4
1011	-5
1010	-6
1001	-7
1000	-8

30. 1's complement (a) 010010_2 (b) 010011_2
 2's complement (a) 010011_2 (b) 010100_2
31. 101001_2 32. 10101010_2 33. 396_{10} 34. $222_8; 10010010_2$
 35. $1717_8, 1010111011_2$ 36. $1273_8, 699_{10}$ 37. $F3_{16}$ 38. 1111100000001011_2
 39. $4C_{16}$ 40. 38_{16}
41. Number = $(-1)^S (1 + F)(2^{E-127})$

Consider the following floating-point binary numbers.

S	E	F
1	10010001	100011100010000000000000

The sign bit is 1. The biased exponent is $10010001 = 145$. Applying the formula we get

$$\begin{aligned} \text{Number} &= (-1)(1.10001110001)(2^{18}) = -11000111000100000000 \\ &= -407,688_{10} \end{aligned}$$

42. $3.248 \times 10^4 = 32480 = 11111011100000_2 = 1.1111011100000 \times 2^{14}$

The MSB will not occupy a bit position because it is always a 1. Therefore, the mantissa is the fractional 23-bit binary number 11110111000000000000000 and the biased exponent is

$$14 + 127 = 141 = 10001101_2$$

The complete floating-point number is

S	E	F
0	10001101	11110111000000000000000

43. unwieldy 44. interface 45. complicated 46. microprocessor
 47. positional 48. radix 49. binary 50. coding
 51. extreme 52. right to left 53. spacers 54. extreme
 55. place 56. $2^n - 1$ 57. bits 58. digits

- | | | | |
|-------------------------------|------------------------|----------------------|----------------|
| 59. octal | 60. base-2 | 61. 8 | 62. 10 |
| 63. 128 | 64. borrow | 65. arithmetic/logic | 66. 100111 |
| 67. 42 | 68. identification | 69. two | 70. eight |
| 71. sign magnitude | 72. zero | 73. shorthand | 74. four-bit |
| 75. letters | 76. nibble | 77. sequential | 78. sixteen |
| 79. adding | 80. E9E, E9F, EA0, EA1 | | 81. scientific |
| 82. always | 83. successive | 84. positional | 85. formats |
| 86. very small and very large | 87. fixed point | 88. True | 89. True |
| 90. False | 91. False | 92. True | 93. False |
| 94. True | 95. True | 96. False | 97. True |
| 98. True | 99. True | 100. False | 101. False |
| 102. False | 103. False | 104. True | 105. True |
| 106. True | 107. False | 108. (a) | 109. (b) |
| 110. (d) | 111. (c) | 112. (d) | 113. (b) |
| 114. (b) | 115. (d) | 116. (b) | 117. (d) |
| 118. (c) | 119. (a) | 120. (b) | 121. (b) |
| 122. (d) | 123. (b) | 124. (b) | 125. (d) |
| 126. (b) | 127. (b) | 128. (d) | |

Codes and Parity

INTRODUCTION

When numbers, letters, or words are represented by a special group of symbols we say that they are being *encoded*, and the group of symbols is called a *code*. Probably one of the most familiar codes is the *Morse Code*, where series of dots and dashes represent letters of the alphabet.

Any decimal number can be represented by an equivalent binary number. The group of 0's and 1's in the binary number can be thought of as a code representing the decimal number. When a decimal number is represented by its *equivalent* binary number, we call it *straight binary coding*.

All digit systems use some form of binary numbers for their internal operation, but the *external world is decimal in nature*. This means that conversions between decimal and binary numbers are being performed often. *Conversions* between decimal and binary can become long and complicated for large numbers. For this reason a means of encoding decimal numbers that combine some features of *both* the decimal and binary systems is used in certain situations. In many applications special codes are used for such *auxiliary functions* as error detection.

WEIGHTED BINARY CODES

In order to represent the 10 decimal digits, 0 through 9, it is necessary to use at least 4 binary digits ($0 = 0000$ and $9 = 1001$). Since there are 16 combinations of four binary digits, of which only 10 combinations are used, it is possible to form a very large number of distinct codes. Of particular importance is the class of *weighted codes*, whose main characteristic is that *each binary*

digit is assigned a weight, and for each group of four-bits, the sum of the weights of those binary digits whose value is 1 is equal to the decimal digit which they represent. In other words, if w_1, w_2, w_3 and w_4 are the weights of the binary digits and x_1, x_2, x_3 and x_4 are the corresponding digit values, then the decimal digit $N = w_4 x_4 + w_3 x_3 + w_2 x_2 + w_1 x_1$ is represented by the binary sequence $x_4 x_3 x_2 x_1$. A *sequence of binary digits which represents a decimal digit is called a code word*. Thus, the above sequence $x_4 x_3 x_2 x_1$ is the code word for N . A number of *weighted four-digit binary codes* are shown in Table 3.1.

Table 3.1 Examples of Weighted Binary Codes

Decimal digit	8	4	2	1	w_4 2	w_3 4	w_2 2	w_1 1	6	4	2	-3
0	0	0	0	0	0	0	0	0	0	0	0	0
1	0	0	0	1	0	0	0	1	0	1	0	1
2	0	0	1	0	0	0	1	0	0	0	1	0
3	0	0	1	1	0	0	1	1	1	0	0	1
4	0	1	0	0	0	1	0	0	0	1	0	0
5	0	1	0	1	1	0	1	1	1	0	1	1
6	0	1	1	0	1	1	0	0	0	1	1	0
7	0	1	1	1	1	1	0	1	1	1	0	1
8	1	0	0	0	1	1	1	0	1	0	1	0
9	1	0	0	1	1	1	1	1	1	1	1	1

3.1 Describe the binary-coded decimal (BCD) system. How will you form a BCD number?

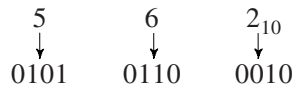
Solution:

Binary-coded decimal (BCD) means that each decimal digit is represented by a binary code of four-bits. This code is useful for outputting to displays that are always numeric (0 to 9), such as those found in digital clocks or digital voltmeters.

To form a BCD number, simply *convert* each decimal digit to its four-bit binary code. To convert BCD to decimal, just *reverse* the process. Start at the decimal point and *break the code into groups of four-bits*. Then write the decimal digit represented by each four-bit group.

3.2 Represent the number 562_{10} in 8421 code.

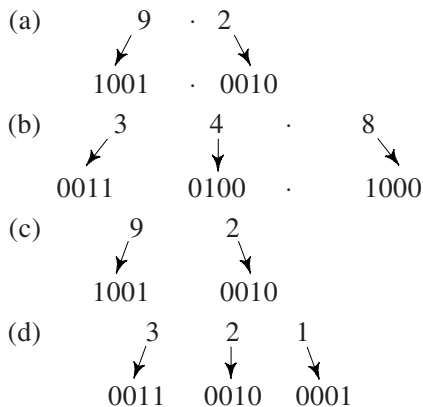
Solution:



3.3 Convert each of the following decimal numbers to BCD.

(a) 9.2 (b) 34.8 (c) 92 (d) 321

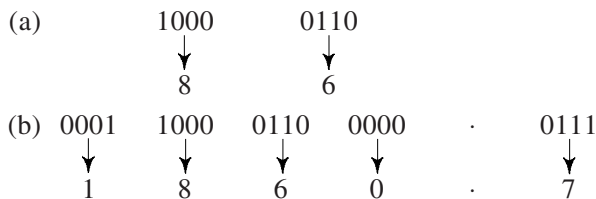
Solution:



3.4 Find the decimal numbers corresponding to the following BCD codes.

(a) 10000110 (b) 0001100001100000.0111

Solution:



BCD ADDITION

BCD is a *numerical code*. Many applications require that arithmetic operations be performed. *Addition* is the most important operation because the other three operations (subtraction, multiplication and division) can be accomplished using addition. To add two BCD numbers proceed as follows:

- (1) Add the two numbers using the rules for binary addition.
- (2) If a four-bit sum is equal to or less than 9, it is a *valid* BCD number.
- (3) If a four-bit sum is greater than 9, or if a carry-out of the sum is generated it is an invalid result. *Add 6(0110) to the four-bit sum to skip the six invalid*

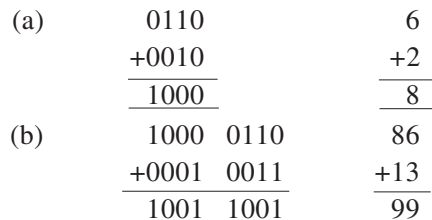
states and return the code to 8421. If a carry results when 6 is added, simply add the carry to the next four-bit group.

3.5 Add the following BCD numbers:

(a) **0110 and 0010**

(b) 1000 0110 and 0001 0011

Solution:



Note: (1) The decimal addition is shown for comparison.

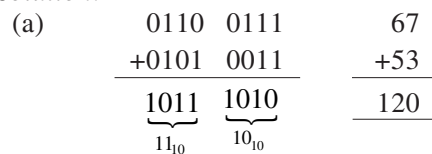
(2) The sum in any four-bit column does not exceed 9, and the results are *valid BCD numbers*.

3.6 Add the following BCD numbers:

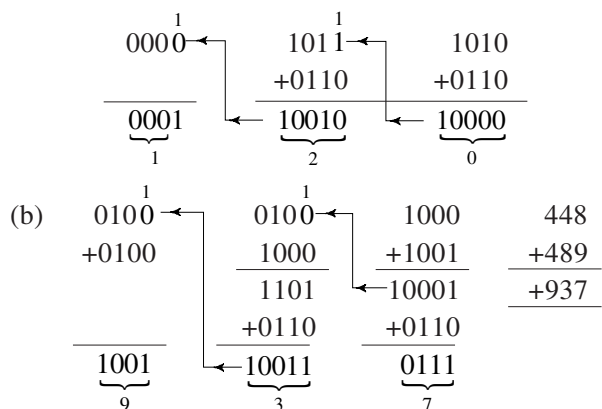
(a) 0110 0111 and 0101 0011

(b) 0100 0100 1000 and 0100 1000 1001

Solution:



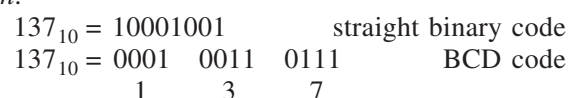
Both groups are *invalid* (>9). Add 6 to both groups.



This is a case of *invalid sum* (greater than 9 or a carry). Add 6 (0110) to skip the invalid states.

3.7 Express the number 137_{10} in BCD and binary. Explain any difference between the two results.

Solution:



- (1) A straight binary code takes the *complete decimal number* and expresses it in binary.

- (2) The BCD code converts each decimal digit to binary *individually*.
- (3) BCD requires more bits than straight binary to represent decimal numbers with more than one digit.

3.8 Compare BCD and binary.

Solution:

BCD is not another number system like binary, octal, decimal and hexadecimal. It is, in fact, the decimal system with each digit encoded in its binary equivalent. A *BCD number* is not the same as a straight binary number. A *straight binary code* takes the complete decimal number and represents it in binary. The BCD code converts each decimal digit to binary individually. BCD requires more bits than straight binary to represent decimal numbers with more than one digit. This is because BCD does not use all possible four-bit groups, and is therefore *somewhat inefficient*.

The main advantage of the BCD code is the relative ease of converting to and from decimal. Only the four-bit code groups for the decimal digits 0 through 9 need to be remembered. This *ease of conversion* is especially important from a hardware standpoint because in a digital system it is the logic circuits that perform conversions to and from decimal.

THE 8421 BCD CODE

The designation 8421 indicates the *binary weights* of the four-bits ($2^3, 2^2, 2^1, 2^0$). The *ease of conversion* between 8421 code numbers and the familiar decimal numbers is the main advantage of this code. The 8421 code is the *predominant BCD code*, and when we refer to BCD, we always mean the 8421 code unless otherwise stated.

In the 8421 code only 10 of the 16 combinations are used. The six combinations that are not used—1010, 1011, 1100, 1101, 1110 and 1111—are *invalid* in the 8421 BCD code.

The BCD code is not self-complementing. A *necessary condition for a weighted code to be self-complementing is that the sum of the weights must equal 9*.

NON-WEIGHTED CODES

Non-weighted codes are *codes that are not positionally weighted*. That is, each position within the binary number is not assigned a fixed value; For example: Excess-3 and Gray code.

Excess-3 Code: Excess-3, also called XS3, is a non-weighted code used to express decimal numbers. Used in some old computers, the code derives its name from the fact that *each binary code is the corresponding 8421 code plus 3 (011)*.

3.9 What is the key feature of the Excess-3 code?

Solution:

The key feature of the Excess-3 code is that it is *self-complementing*. That is, the 1's complement of an Ex-

cess-3 number is the Excess-3 code for the 9's complement of the corresponding decimal number. For example, the Excess-3 code for decimal 4 is 0111. The 1's complement of 0111 is 1000, which is the Excess-3 code for decimal 5, and 5 is the 9's complement of 4.

The 1's complement is easily produced with logic circuits by simple *inverting* each bit. The self-complementing property makes the Excess-3 code useful in some arithmetic operations, because subtraction can be performed using the *9's complement method*.

EXCESS-3 ADDITION

To add in Excess-3, add the binary numbers. If there is no carry out of the four-bit group *subtract 011*; if there is a carry out *add 011*.

3.10 Convert the following numbers to Excess-3 code.

(a) 87

(b) 159

Solution:

(a) 8 +3 ——— 11 ↓ 1011	7 +3 ——— 10 ↓ 1010
(b) 1 5 9 +3 +3 +3 ——— ——— ——— 4 8 12 ↓ ↓ ↓ 0100 1000 1100	

3.11 Add 3 and 2 in Excess-3.

Solution:

3 = 0011 binary +0011 ——— 0110 Excess-3 ↓ 0110 +0101 ——— 1011 -0011 ——— 5 = 1000	2 = 0010 binary +0011 ——— 0101 Excess-3 ↓ 0101 +1011 ——— 1011 -0011 There is no carry out. Subtract 0011. ——— 5 = 1000
---	--

3.12 Add 6 and 8 in Excess-3.

Solution:

6 = 0110 binary +0011 ——— 1001 Excess-3 ↓ 1001 +1011 ——— 10100 There is a carry out. +0011 Add 0011 ——— 14 = 10111	8 = 1000 binary +0011 ——— 1011 Excess-3 ↓ 1011 +1011 ——— 10111
---	--

Note: The answer includes six invalid combinations also (14 + 6 + 3)

GRAY CODE

The Gray code belongs to a class of codes called *minimum change code*. Successive coded characters never differ in more than one-bit. The Gray code is an *unweighted code*. Because of this the Gray code is not suitable for arithmetic operations but finds application in input/output devices, some types of analog-to-digital converters, and designation of rows and columns in a Karnaugh map.

3.13 How will you generate Gray code?

Solution:

The basic Gray code configuration is shown in Fig. 3.1 (a). A three 3-bit Gray code may be obtained by merely *reflecting* the two-bit code about an axis at the end of the code and assigning a third-bit as 0 above the axis and as 1 below the axis. This is illustrated in Fig. 3.1(b). By reflecting the three-bit code, a four-bit code may be obtained as in Fig. 3.1(c).

0	0	0	0	0	0	0	0	0	0
0	1	0	0	1	0	0	0	1	0
1	1	0	1	1	0	0	1	1	1
1	0	0	1	0	0	0	1	0	0
					1	1	0	0	0
					1	1	1	1	1
					1	0	1	0	1
					1	0	0	0	0
					1	1	0	0	1
					1	1	1	1	1
					1	1	1	0	0
					1	0	1	0	1
					1	0	1	1	1
					1	0	0	1	0
					1	0	0	0	0

(a)

(b)

(c)

Fig. 3.1 Generating Gray Code.

3.14 What is a cyclic code?

Solution:

In many practical applications, e.g. analog-to-digital conversion, it is desirable to use codes in which all successive code words differ in only one digit. Codes that have such a property are referred to as *cyclic codes*. A particularly important cyclic code is the Gray code. The feature that makes this cyclic code useful is the simplicity of the procedure for converting from the binary number system into the Gray code.

3.15 Explain the term resolution.

Solution:

The Gray code, also referred to as the *reflected code*, can have as many bits as necessary, and *the more the bits, the more the possible combinations of output codes*

(number of combinations = 2^n). A four-bit Gray code, for example, has $2^4 = 16$ different representations giving a *resolution* of 1 out of 16 possible angular positions at 22.5° each

$$2^4 = 16 \quad \text{and} \quad 360^\circ/16 = 22.5^\circ$$

3.16 Explain the relevance of Gray code in reducing errors in position indicators.

Solution:

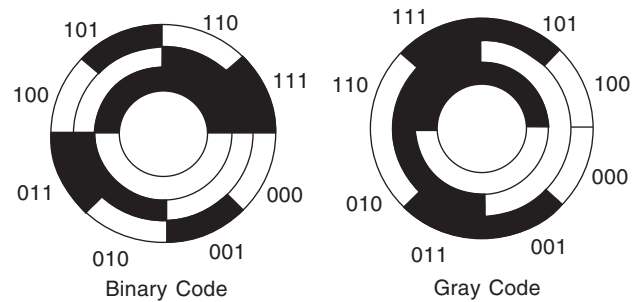


Fig. 3.2 Positional Indicating Codes.

Consider a rotating disk that must provide an output of the position in three-bit binary (Fig. 3.2). When the brushes are on the black part, they output a 1, when on a white part, they output a 0. However, consider what happens when the brushes are on the 111 sector and almost ready to enter the 000 sector. If one brush were slightly ahead of the other, say the 4's brush, the position would be indicated by a 011 instead of a 111 or 000. Therefore, a 180-degree ($^\circ$) error in disk position would result. Since it is physically impossible to have all the brushes precisely aligned, *some error will always be present at the edges of the sectors*.

The Gray code was introduced to reduce this error. It assures that *only one-bit will change each time the decimal number is incremented*. Whereas the binary system requires four-bits to change when going from 7 (0111) to 8 (1000), the Gray code requires only one-bit to change (0100 to 1100).

BINARY-TO-GRAY CONVERSION

To convert from binary-to-Gray code the following rules apply:

- (1) The MSD (left-most digit) in the Gray code is *the same as* the corresponding digit in the binary number.
- (2) Going from left to right, add each pair of binary digits to get the next Gray code digit. *Disregard carries*.

3.17 Convert the binary number 10110_2 to Gray code.

Solution:

1	0	1	1	0 ₂	binary
1	1	1	0	1	Gray

Note: Disregard carry. $1 + 1 = 0$ and not 10

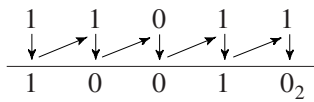
GRAY-TO-BINARY CONVERSION

To convert from Gray to binary, a similar procedure is adopted, but there are some differences. The following rules apply:

- (1) The MSD in the binary code is *the same as* the corresponding digit in the Gray code.
- (2) Add each binary digit generated to the Gray code digit in the *next adjacent* position. *Disregard carries.*

3.18 Convert Gray code number 11011 to binary code.

Solution:



ALPHANUMERIC CODES

To get information into and out of a computer, we need more than just *numeric* representations; we also have to take care of *letters and symbols* used in day-to-day processing. Information such as names, addresses, and item descriptions must be input and output in a readable format. But a digital system can deal only with 1's and 0's. Therefore, *we need a special code to represent all alphanumeric data* (letters, symbols, and numbers).

In the strictest sense, codes that represent numbers and alphabetic characters (letters) are called *alphanumeric codes*. Most of these codes, however, also represent symbols and various instructions necessary for conveying *intelligible information*.

3.19 What are the requirements of an alphanumeric code?

Solution:

At a minimum, an alphanumeric code must represent 10 decimal digits and 26 letters of the alphabet, for a total of 36 items. This requires 6 bits in each code combination, because five bits are insufficient ($2^5 = 32$). There are 64 total combinations of 6 bits, so we have 28 unused code combinations. Obviously, in many applications, symbols other than just numbers and letters are necessary to communicate completely. We need spaces to separate words, periods to mark the end of sentences or for decimal points, instructions to tell the receiving system what to do with the information, and more. So, with codes that are 6 bits long, we can handle decimal numbers, the alphabet, and 28 other symbols. This gives an idea of *the requirements of a basic alphanumeric code*.

Several coding systems have been invented that represent alphanumeric information as a series of 1's and 0's. These systems vary in complexity from the Morse code used in *telegraph work* to the Hollerith code used with *punched cards*.

THE ASCII CODE

Most industry has settled on an input/output (I/O) code called the *American Standard Code for Information Interchange*. The ASCII (pronounced "as-kee") code uses 7 bits to represent all the alphanumeric data used in computer I/O. Seven bits will yield 128 different code combinations, as listed in Table 3.2. ASCII is basically a *7-bit code*. An 8th bit is usually added and is (a) always set to a 1, (b) always set to a 0, or (c) used as a parity bit.

Table 3.2 American Standard Code for Information Interchange

	000	001	010	011	100	101	110	111
0000	NUL	DLE	SP	0	@	P	'	p
0001	SOH	DC ₁	!	1	A	Q	a	q
0010	STX	DC ₂	"	2	B	R	b	r
0011	ETX	DC ₃	#	3	C	S	c	s
0100	EOT	DC ₄	\$	4	D	T	d	t
0101	ENQ	NAK	%	5	E	U	e	u
0110	ACK	SYN	&	6	F	V	f	v
0111	BEL	ETB	'	7	G	W	g	w
1000	BS	CAN	(8	H	X	h	x
1001	HT	EM)	9	I	Y	i	y
1010	LF	SUB	*	:	J	Z	j	z
1011	VI	ESC	+	;	K	[k	{
1100	FF	FS	,	<	L	\	l	
1101	CR	GS	-	=	M]	m	}
1110	SO	RS	.	>	N	^	n	~
1111	SI	US	/	?	O	—	o	DEL

Definitions of Control Abbreviations:

ACK	Acknowledge	FS	Form separator
BEL	Bell	GS	Group separator
BS	Backspace	HT	Horizontal tab
CAN	Cancel	LF	Line feed
CR	Carriage return	NAK	Negative acknowledge
DC ₁ -DC ₄	Direct control	NUL	Null
DEL	Delete idle	RS	Record separator
DLE	Data link escape	SI	Shift in
EM	End of Medium	SO	Shift out
ENQ	Enquiry	SOH	Start of heading
EOT	End of transmission	STX	Start of text
ESC	Escape	SUB	Substitute
ETB	End of transmission block	SYN	Synchronous idle
ETX	End text	US	Unit separator
FF	Form feed	VT	Vertical tab

3.20 How will you use the ASCII Code?

Solution:

Each time a key is depressed on the ASCII keyboard, that key is converted into its *ASCII code* and processed by the computer. Then, before outputting the computer connects to a display terminal or printer, all information is converted from ASCII into standard English.

To use this table, place the *4-bit group* in the least significant positions and the *3-bit group* in the most significant positions.

3.21 (a) What is ASCII code for G?

(b) Using Table 3.2 determine the ASCII code for *p*.

Solution:

(a) ASCII code for G is

$\overbrace{100}^{3\text{-bit group}} \quad \overbrace{0111}^{4\text{-bit group}}$

(b) ASCII code for lower case letter *p* is

1 1 1 0 0 0 0

3.22 Write your name in ASCII.

Solution:

S . P . BALI
 1010011 0101110 1010000 0101110 1000010 1000001
 S . P . B A
 1001100 1001001
 L I

EBCDIC CODE

Another alphanumeric code frequently encountered is called *Extended Binary Coded Decimal Interchange Code*. EBCDIC (pronounced “eb-si-dic”) is an 8 bit code in which the decimal digits are represented by the 8421 BCD code preceded by 1111. Both lower case and upper case letters are represented in addition to various other symbols and commands.

THE HOLLERITH CODE

The Hollerith code is the code used in *punched cards*. Each card has 80 columns oriented vertically, and 12 rows oriented horizontally. The *rows* are numbered 0 through 9, 11 and 12. The *columns* are numbered 1 through 80, each one containing one character. Each character is uniquely identified by the rows punched in that column. The letter A is 12-1 punch (A punch in the 12-row and a punch in the 1-column), for example.

3.23 What is the practical application of Hollerith code?

Solution:

The Hollerith code is BCD, rather than natural binary oriented. Thus, translation to and from EBCDIC is fairly

simple. Since most large computers use punched cards, they use Hollerith for their card readers and punches and EBCDIC within the computer itself.

3.24 What is the need for error detection and correction codes?

Solution:

The codes for *error detection and correction* are used whenever we need to transmit data from one processor to another over noisy channels or whenever errors are likely to result from unstable environmental conditions. Such codes require *special encoding and decoding hardware*, which increases the cost and decreases the performance of a digital system, thus forcing the designer who uses these codes to trade cost and performance for *data security*.

Most modern digital equipment is designed to be relatively *error-free*, and the probability of errors is very low. However, we must realize that digital systems often transmit thousands, even millions of bits per second so that even a very low *rate of occurrence of errors* can produce an occasional error that might prove to be bothersome, if not disastrous. For this reason many digital systems employ some method of *detection* (and sometimes *correction*) of errors. *One of the simplest and most widely used schemes for error detection is the parity method.*

3.25 Explain how parity testing is done?

Solution:

Parity testing is done by adding a small amount of redundant information to each word of data to allow it to be checked. The extra information takes the form of a *parity bit* which is added at the end of each data bit. The *polarity* of the added bits is chosen so that the total number of 1's within the data word (including the added parity bit) is either always even (*even parity*) or always odd (*odd parity*). An even parity system is shown in Fig. 3.3.

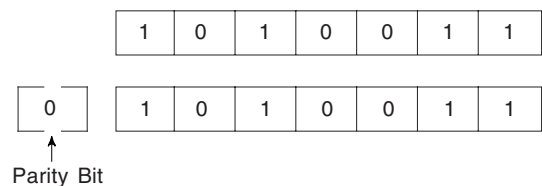


Fig. 3.3 An Even-parity System.

If the parity is incorrect on reception, an error has been detected and the system must take appropriate action. Although this technique indicates that an error has occurred, it cannot determine which bit or bits are incorrect. If two errors are present, the errors will not be detected. This simple error detecting technique will either detect an even, or an odd, number of errors. Random numbers thus have a 50% chance of passing the parity test.

Parity testing is often used in communication channels where it is used to give confidence that the line is working correctly. Although the *reliability* of testing any one word is low, when applied to a large number of words it is sure to detect errors if the line is unreliable.

When the parity method is being used, the transmitter and the receiver must be *in agreement*¹, *in advance*¹, as to whether odd or even parity is being used. There is no specific advantage of one over the other, although even parity seems to be used more often. The transmitter must attach an appropriate parity bit to each unit of information that it transmits.

3.26 Give three examples of (a) odd parity and (b) even parity.

Solution:

(a) Odd parity

Parity	Data							Total 1's
1	1	0	1	0	1	1	0	5
0	1	0	1	0	1	1	1	5
1	0	0	1	0	0	1	0	3

(b) Even parity

Parity	Data							Total 1's
1	1	0	1	0	1	1	1	6
1	0	0	0	0	1	1	1	4
0	1	0	0	0	1	0	0	2

CHECK SUM

An alternative method of checking the correctness of data is to use a checksum. *This provides a test of integrity of a block of data rather than of individual words.* When a group of words is to be transmitted, the words are summed at the transmitter and the sum is transmitted after the data. At the receiver, the words are again summed and compared with the sum produced by the transmitter. If both the results agree, the data is probably correct. If they don't, an error has probably been detected. As with the parity check, *the test gives no indication as to the location of the error but simply indicates that an error has occurred.* The action depends on the nature of the system. It might involve sending the data again or sounding an alarm to warn the operator.

HAMMING CODE

The parity and check sum techniques both send a small amount of redundant information to allow the integrity of the data to be tested. If one is prepared to send additional redundant information, it is possible to construct codes that not only detect the *presence* of errors, but also indicate their *location* within a word, allowing them to be corrected. An example of this technique is the well-known *Hamming code*.

The performance of these codes in terms of their ability to *detect and correct multiple errors* depends upon the amount of redundant information that can be tolerated. The more the redundancy that is incorporated, the greater is the rate at which data must be sent and the more complicated the system. It is also not possible to construct a code that will allow an unlimited number of errors. This would imply that the system could produce the correct output with a random input, clearly an impossibility.

3.27 Explain the format for Hamming code.

Solution:

The Hamming code format for four data bits would be:

$$D_7 \ D_6 \ D_5 \ P_4 \ D_3 \ P_2 \ P_1 \rightarrow 7\text{-bit code}$$

where the D-bits are the *data bits* and the P-bits are the *parity bits*. P_1 is set so that it establishes *even parity* over bits 1, 3, 5 and 7 (P_1, D_3, D_5 and D_7). P_2 is set for even parity over bits 2, 3, 6 and 7 (P_2, D_3, D_6 and D_7). P_4 is set for even parity over bits 4, 5, 6, and 7 (P_4, D_5, D_6 and D_7).

The above concept can be extended to any number of bits. A *15-bit code*, for example, would have the following format:

$$D_{15}, D_{14}, D_{13}, D_{12}, D_{11}, D_{10}, D_9, P_8, D_7, D_6, D_5, P_4, D_3, P_2, P_1 \rightarrow 15\text{-bit code}$$

Note: Parity bits are inserted at each 2^n bit. This is true for *Hamming codes of any length*.

3.28 Data bits 1011 must be transmitted. Construct even-parity, 7-bit, Hamming code for this data.

Solution:

P_1 must be a 1 in order for bits 1, 3, 5, and 7 to be even parity.

P_2 must be a 0 in order for bits 2, 3, 6, and 7 to be even parity.

P_4 must be a 0 in order for bits 4, 5, 6, and 7 to be even parity.

Therefore, the final code is

$$\begin{array}{ccccccc} D_7 & D_6 & D_5 & P_4 & D_3 & P_2 & P_1 \\ 1 & 0 & 1 & 0 & 1 & 0 & 1 \end{array}$$

The Hamming code data are now ready for transmission and reception. At the receiving end, they are decoded to see if any errors have occurred. Bits 1, 3, 5 and 7; bits 2, 3, 6, and 7; and bits 4, 5, 6, and 7 are all checked for even-parity. *Should they check out, there is no error. However, should there be an error, the problem bit can be located by forming a 3-bit binary number out of the three parity checks.*

3.29 Encode data bits 0101 into a 7-bit even-parity Hamming code.

Solution:

$$\begin{array}{ccccccc} D_7 & D_6 & D_5 & P_4 & D_3 & P_2 & P_1 \\ 0 & 1 & 0 & 1 & 1 & 0 & 1 \end{array}$$

3.30 Assume that the data has been encoded in a 7-bit even-parity Hamming code and the number 1011011 is received. Find out the bit(s) in error. What will the *corrected code* be?

Solution:

1	0	1	1	0	1	1
D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁

- (1) Bits 1, 3, 5, and 7 associated with parity bit P₁ contain an error.
- (2) Bits 2, 3, 6, and 7 associated with parity bit P₂ contain no error.
- (3) Bits 4, 5, 6, and 7 associated with parity bit P₃ contain an error.

P ₄	P ₂	P ₁
Error word = 1	0	1 ₂

$= 5_{10}$

Therefore, bit 5 of the transmitted code is in error. The corrected code should read.

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
1	0	0	1	0	1	1

3.31 A 7-bit Hamming code is received as 1111101. What is the *corrected code*? Assume even parity.

Solution:

D ₇	D ₆	D ₅	P ₄	D ₃	P ₂	P ₁
1	1	1	1	1	0	1

- (1) Bits 4, 5, 6, and 7 → no error (0)
- (2) Bits 2, 3, 6, and 7 → error (1)
- (3) Bits 1, 3, 5, and 7 → error (0)

Error word = $0\ 10_2 = 2_{10}$

Bit 2 is in error, and the corrected code is
1111111

SUMMARY

- 8421 BCD code is the most *prominent* BCD code.
- A decimal number is *converted* to BCD by replacing each decimal digit with the appropriate 4-bit binary code.
- In weighted binary codes each binary digit is assigned a *weight*.
- In BCD addition if a 4-bit sum is equal to or less than 9, it is a *valid* BCD number.
- Non-weighted codes are *not positionally weighted*.
- Gray code is a *minimum change* code.
- Excess-3 code is obtained by *adding 3* to every decimal digit and then converting the result to 4-bit binary.
- In Excess-3 *addition*, if there is no carryout of the 4 bit group subtract 011; if there is a carryout add 011.
- *Alphanumeric* codes include binary codes for letters, numbers and symbols.
- ASCII and EBCDIC are the *most commonly used* alphanumeric codes.
- *Cyclic* codes are used to avoid errors when going from one binary number to the next.
- The ASCII is a 7-bit alphanumeric code that is widely used in computer systems for *input and output* of information.
- The Hollerith code is used in *punched cards*.
- Parity, check sums, and parity data codes are used for *error correction*.
- Hamming code is used of *error detection and correction*.

Test your
understanding

REVIEW QUESTIONS

1. Explain the difference between a *weighted* and a *non-weighted* code.
2. Give two examples of *weighted* codes.
3. Give two examples of *non-weighted* codes.
4. *Convert* 5429 to BCD.
5. Convert BCD 0100 0011 1001 1000 to decimal.
6. What is the main *advantage* of BCD?
7. What is the necessary condition for a weighted code to be *self-complementing*?
8. What is the *key feature* of the Excess-3 code?
9. What is the *key feature* of Gray code?

10. What is a *reflective code*?
11. Where is *reflectivity* desirable?
12. What is a *sequential code*?
13. Which code is used in *punched cards*?
14. What scheme is used for *error detection*?
15. What is the requirement of *parity method*?

Test your
understanding

SUPPLEMENTARY PROBLEMS

16. Represent the number 2048 in:
 - (a) binary
 - (b) BCD
 - (c) Excess-3 code
 - (d) Gray code
17. Encode the following decimal numbers in BCD code:
 - (a) 46
 - (b) 327.89
 - (c) 20.305
18. Encode the numbers in Problem 17 to Excess-3 code.
19. Encode the numbers in Problem 17 to Gray code.
20. Express the following decimal numbers in 8421 BCD code:
 - (a) 328
 - (b) 1497
 - (c) 9725
21. Express the following Excess-3 numbers as decimals:
 - (a) 0110 1011 1100 0111
 - (b) 0011 0101 1010 0100
22. Convert the number 9450 to BCD code and show that it is not the same as the straight binary equivalent.
23. Convert 11100001110110 BCD to decimal.
24. Convert 9_{10} and 4_{10} to BCD. Add the conversions.
25. Add 8_{10} and 9_{10} in BCD.
26. Convert the following Gray numbers to equivalent binary numbers:
 - (a) 111011
 - (b) 101110101
27. Write your full name in ASCII.
28. Attach an even-parity bit as MSB for ASCII code.
29. Repeat the above problem for odd parity.
30. Find the number of bits required to encode:
 - (a) 56 elements of information
 - (b) 130 elements of information
31. Represent the decimal numbers
 - (a) 27
 - (b) 396 and
 - (c) 4096 in binary form in
 - (a) binary code
 - (b) BCD code
 - (c) Excess-3 code
 - (d) Gray code
 - (e) Octal code
 - (f) Hexadecimal code
32. Represent the decimal number 2048 in
 - (a) binary
 - (b) BCD code
 - (c) Excess-3 code
 - (d) Gray code
33. 7-bit Hamming code is received as 1101101. Locate the error position and find the correct code.
34. The message below has been coded in the Hamming code and transmitted through a noisy channel. Decode the message assuming that a single error has occurred in each word code:

1001001 0111001 1110110 0011011
35. If odd parity is being used, what is the parity bit when the decimal number 43 is converted to binary?
36. The decimal number 6 is to be transmitted using the Hamming error correcting code. (a) What are the values of $P_1 P_2 P_3$ (b) What 7-digit binary number is transmitted? (c) If the binary number 1100111 is received, how can the location of the error be determined?
37. Typically, digital thermometers use BCD to drive their *digital displays*.
 - (a) How many *BCD bits* are required to drive a 3-digit thermometer display?
 - (b) What 12 *bits* are sent to the display for a temperature of 147° ?
38. Most PC-compatible computer systems use a 20-bit address code to identify each of over 1 million binary locations.
 - (a) How many *hexcharacters* are required to identify the address of each memory location?
 - (b) What is the 5-digit *hexaddress* of the 200th memory location?
39. If the part number 651-M is stored in ASCII in a computer memory. List the *contents* of its memory location.

Test your
understanding

OBJECTIVE TYPE QUESTIONS

Fill in the Blanks

40. When a decimal number is represented by its _____ binary number we call it straight binary coding.
41. The external world is _____ in nature.
42. Conversions between decimal and binary can become _____ and _____ for large numbers.
43. In many applications special codes are used for _____ functions.
44. Error detection and correction codes are used whenever we want to transmit data from one processor to another over _____ channels.
45. Error detection and correction codes require special _____ and _____ hardware.
46. In weighted binary codes each binary digit is assigned a _____.
47. A _____ of binary digits which represents a decimal digit is called a code word.
48. The _____ is the predominant BCD code.
49. The BCD code is not _____.
50. In a self-complementing code, the sum of the _____ must equal 9.
51. If a 4-bit sum is equal to or less than 9, it is a _____ BCD number.
52. A BCD number is not the _____ as a straight binary number.
53. BCD is not another _____ system.
54. BCD converts each decimal digit to binary _____.
55. The main advantage of BCD is _____ of conversion to and from decimal.
56. Because BCD does not use all possible 4-bit groups it is somewhat _____.
57. Non-weighted codes are codes that are not _____ weighted.
58. Excess-3 code is _____.
59. Gray code belongs to a class of codes called _____ change codes.
60. Gray code is not suited for _____ operations.
61. Gray code is employed in the designation of _____ and _____ in a Karnaugh map.
62. Gray code is also referred to as a _____ code.
63. The Gray code can be _____ to any number of bits.
64. The Morse code uses two _____ conditions.
65. The Hollerith code is _____ rather than natural binary oriented.
66. The ASCII code uses _____ to represent all the alphanumeric data used in computer I/O.
67. The _____ code is the code used in punched cards.
68. The EBCDIC is an _____ code.
69. The major cause of any transmission errors is _____.
70. One of the simplest and most widely used schemes for error detection is the _____ method.
71. Parity bit is added at the _____ of each data word.
72. The _____ method of correctness of data provides a test of integrity of a block of data rather than individual words.
73. The checksum test gives no indication as to the _____ of the error.
74. When the parity method is being used, the transmitter and the receiver must be in _____ in advance.
75. _____ parity seems to be used more often.

True/False Questions

State whether the following statements are True or False.

76. To form a BCD number, simply convert each decimal digit to its 8-bit binary code.
77. The designation 8421 indicates the binary weights of the 4 bits.
78. BCD is an alphanumeric code.
79. BCD is somewhat inefficient.
80. Non-weighted codes are not positionally weighted.
81. Gray code is a minimum change code.
82. ASCII is a numeric code.
83. ASCII is an 8-bit code.
84. Hollerith is a 7-bit code.

85. Parity testing is done by adding a small amount of redundant information.
 86. The checksum method simply indicates an error has occurred.
 87. Parity bits are inserted at each 2^n -bit.

Multiple Choice Questions

88. The parity of the binary number 1100110
 (a) is even (b) is not known
 (c) is odd (d) is same as the number of zeros
89. Decimal number 13 is represented in natural BCD as
 (a) 1101 (b) 00010011
 (c) 00001101 (d) 00011101
90. The code used in computer cards is
 (a) Gray code (b) natural BCD code
 (c) 12-bit Hollerith code (d) ASCII code
91. The decimal number 279 will be represented in Excess-3 code as
 (a) 001001111001 (b) 010110101100
 (c) 100010111 (d) 100011010
92. When representing in the following code the consecutive decimal numbers differ only in one-bit.
 (a) Excess-3 (b) Gray
 (c) BCD (d) Hexadecimal
93. The number 27 is represented by 01011010 in
 (a) straight binary (b) natural BCD
 (c) Gray code (d) Excess-3 code
94. The number 6_{10} in Excess-3 is written as
 (a) 0110 (b) 0011
 (c) 1101 (d) 1001
95. The Hamming code has a minimum distance d_m of
 (a) 3 (b) 4 (c) 5 (d) 6
96. The ASCII code is basically a
 (a) 7-bit code (b) 12-bit code
 (c) 4-bit code (d) 6-bit code
97. The Hollerith code is used in
 (a) floppy disk (b) hard disk
 (c) VDU (d) punched cards
98. A string of 8-bits is known as a
 (a) quad (b) octet
 (c) nibble (d) byte
99. The parity of the binary number 100110011 is
 (a) even (b) odd (c) 2 (d) 1
100. A necessary condition for a weighted code to be self-complementing is that the sum of its weights must be equal to
 (a) 9 (b) 8
 (c) an even number (d) an odd number
101. If each successive code differs from its preceding code by a single bit only, then this code is called
 (a) BCD code (b) Gray code
 (c) weighted code (d) binary code

ANSWERS

1. *Weighted codes* obey the positional weighting principles. *Non-weighted codes* are not positionally weighted.
2. The 8421 and the 2421 are *weighted codes*.
3. Excess-3 and Gray code are *non-weighted codes*.

- If you look at these memory locations in hex, they will read

36 35 31 2D 4D

- | | | | |
|------------------------|------------------------|------------------------|------------------|
| 40. equivalent | 41. decimal | 42. long, complicated | 43. auxiliary |
| 44. noisy | 45. encoding, decoding | | 46. weight |
| 47. sequence | 48. 8421 | 49. self-complementing | |
| 50. weights | 51. valid | 52. same | 53. number |
| 54. individually | 55. ease | 56. inefficient | 57. positionally |
| 58. self-complementing | 59. minimum | 60. arithmetic | |
| 61. rows, columns | 62. reflected | 63. expanded | 64. signaling |
| 65. BCD | 66. 7 bits | 67. Hollerith | 68. Alphanumeric |
| 69. electrical noise | 70. parity | 71. end | 72. checksum |
| 73. location | 74. agreement | 75. even | 76. False |
| 77. True | 78. False | 79. True | 80. True |
| 81. True | 82. False | 83. False | 84. False |
| 85. True | 86. True | 87. True | 88. (a) |
| 89. (b) | 90. (c) | 91. (b) | 92. (b) |
| 93. (d) | 94. (d) | 95. (c) | 96. (a) |
| 97. (d) | 98. (d) | 99. (b) | 100. (a) |
| 101. (b) | | | |

Logic Gates

INTRODUCTION

Logic gates are the *basic building blocks* for forming digital electronic circuitry. A logic gate has one output terminal and one or more input terminals. Its output will be HIGH (1) or LOW (0) depending on the *digital level(s)* at the input terminal(s). Through the use of *logic gates*, we can design digital systems that will evaluate digital input levels and produce a *specific output response* based on that particular logic circuit design. The seven logic gates are AND, OR, INVERTER, NAND, NOR, exclusive-OR, and exclusive-NOR.

A *logic system* may involve a great many gates, and each gate may have many components. Because of this, it would be confusing to show a logic system as an electronic circuit complete with all components. Instead, *each type of gate is represented by a particular graphic symbol*.

4.1 What is the difference between Boolean algebra and ordinary algebra?

Solution:

The algebra used to symbolically describe logic functions is *Boolean algebra*. As with ordinary algebra, the letters of the alphabet can be used to represent *variables*. The primary difference is that *Boolean algebra variables can have only the values 0 or 1*. Boolean algebra is ideally suited to dealing with the problems of binary arithmetic and electronic digital systems.

4.2 Differentiate between positive and negative logic.

Solution:

There are two values for logic-value assignment. Choosing the high-level *H* to represent logic 1, as shown in Fig. 4.1(a), defines a positive-logic system. Choosing

the low-level *L* to represent logic 1, as shown in Fig. 4.1(b), defines a negative-logic system. It is not signal polarity that determines the type of logic, but rather the assignment of logic values according to the relative amplitudes of the signals.

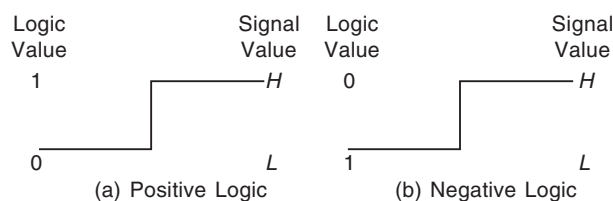


Fig. 4.1 Signal-amplitude Assignment and Type of Logic.

Logic Equation	$A = B$			
Graphic Representation	Point A	Wire	Point B	
Truth Table	A	B	or	A B
	0V	0V		0 0
	+V	+V		1 1
Logic Symbol				

Fig. 4.2 Function: Logical Equality. Symbol denotes a Non-inverting Amplifier or Buffer (Performs no Logic Function)

BUFFER

A buffer is a special solid-state device used to increase the drive current at the output. It is also used for isolation between output and input. A non-inverting buffer, Fig. 4.2, has no logical function.

4.3 What is the significance of a truth table? Draw the truth tables for two-input, three-input and four input circuits.

Solution:

A truth table simply shows all of the possible values for the inputs to a function, then shows the resultant outputs for each combination of inputs. The purpose of truth tables is not to determine whether or not circuits are 'lying', rather they verify 'truth' in the logical sense—that a given set of input conditions will produce a 'true' output, while some other set of input conditions will produce a 'false' output. A *true state* is indicated by the digit 1, and a *false state* is indicated by the digit 0.

Figure 4.3 shows samples of truth tables for two-, three- and four-input logic circuits. Each table lists *all possible combinations* of input logic levels on the left, with *resultant logic levels* for output X on the right. Of course, the actual value of X will depend on the *type* of logic circuit.

A Inputs B		Output X	
Input		Output	
A	B	X	
0	0	1	
0	1	0	
1	0	0	
1	1	0	

(a)

A	B	C	X
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

(b)

A	B	C	D	X
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	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	0
1	1	1	1	1

(c)

Fig. 4.3 Examples of Truth Tables for (a) Two-input, (b) Three-input and (c) Four-input Circuits.

- Note:* 1. The number of input combinations for an n -input truth table is 2^n .
 2. The list of all possible input combinations follows the binary counting sequence.

THE AND FUNCTION

If a situation which may be described with Boolean variables gives a desired result *only when all* of several external conditions are satisfied, then that situation is said to obey the Boolean *AND function*. AND gates may have any number of inputs (Fig. 4.4).

4.4 What voltage level is considered true (HIGH) and what level false (LOW)?

Solution:

Many logic gates that are on the market today define a 5 V level as true and a 0 V level as false. In actual

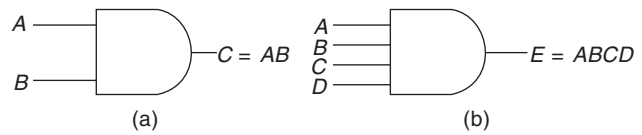


Fig. 4.4 (a) Two-input AND Gate. (b) Four-input AND Gate.

practice, any voltage *above* a certain level (for example 3 V) is considered true, and any voltage *below* a certain voltage level (for example 1V) is considered false.

4.5 Represent the AND function by switch analogy.

Solution:

The switch analogy of AND function is shown in Fig. 4.4. The lamp will light only if both *A and B* are closed. The lamp will not light if *either A or B* is closed.

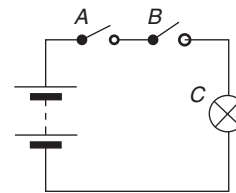


Fig. 4.5 Switch Analogy of AND Function.

4.6 Draw the truth table of a four-input AND gate.

Solution:

The truth table for a four-input AND gate is shown in Fig. 4.3(c).

PULSED OPERATION

In a majority of applications, the inputs to a gate are not stationary levels but are *voltages that change frequently between two logic levels* and can be classified as pulse waveforms. An AND gate obeys the truth table operation regardless of whether its inputs are constant levels or pulsed levels.

A useful means of the output response of a gate to varying input-level changes is by means of a timing diagram. A *timing diagram* illustrates graphically how the output levels change in response to input changes.

4.7 Illustrate and explain the pulsed operation of the AND gate.

Solution:

In examining the pulsed operation of the AND gate, we will look at the inputs *with respect to each other* in order to determine the output level at any given time. For example, in Fig. 4.6, the inputs are both HIGH (1) during the interval t_1 , making the output HIGH (1) during this interval. During interval t_2 , input A is LOW (0) and input B is HIGH (1), so the output is LOW (0). During interval t_3 , both inputs are HIGH (1) again, and therefore the output is HIGH (1). During interval t_4 , input A is HIGH (1) and input B is LOW (0), resulting

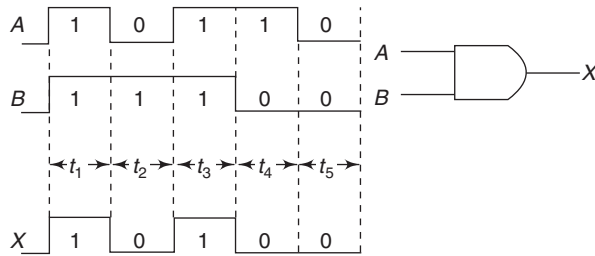


Fig. 4.6 Example of Pulsed AND Gate Operation.

in LOW (0) output. Finally, during interval t_5 , input A is LOW (0), input B is LOW (0), and the output is therefore LOW (0).

4.8 Determine the output waveform for the AND gate in Fig. 4.7.

Solution:

The output X will be HIGH only when both A AND B (AB) are HIGH at the same time. Using this fact the output waveform can be determined as shown in Fig. 4.6.

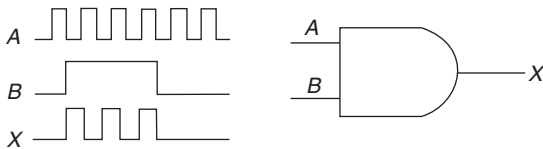


Fig. 4.7

4.9 What will happen to the X output waveform if the B input in 4.8 is kept at the 0 level?

Solution:

With B kept low, the X output will also stay LOW. This can be reasoned in two different ways. First, with $B = 0$ we have $X = A \cdot B = A \cdot 0 = 0$, since *anything multiplied (ANDed) by 0 will be 0*. Another way to look at it is that *an AND gate requires that all inputs be HIGH in order for the output to be HIGH* and this cannot happen if B is kept low.

4.10 Determine the output X for the three-input AND gate in Fig. 4.8.

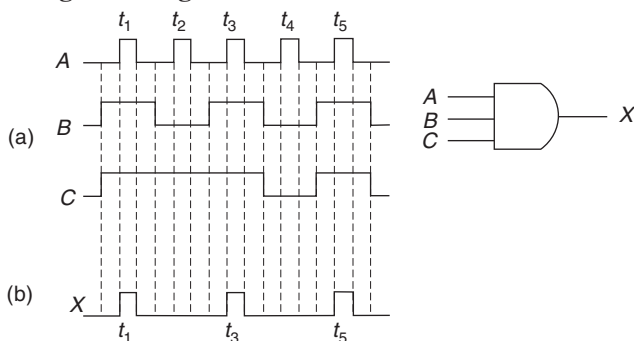


Fig. 4.8

Solution:

The output of a three-input AND gate will be HIGH only when all three inputs are HIGH as shown in Fig. 4.8.

4.11 Sketch the output waveform at X for the two-input AND gates shown in Fig. 4.9.

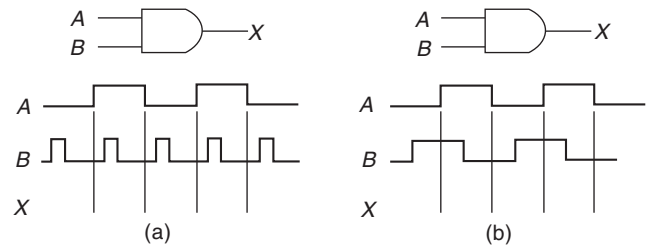


Fig. 4.9

Solution:

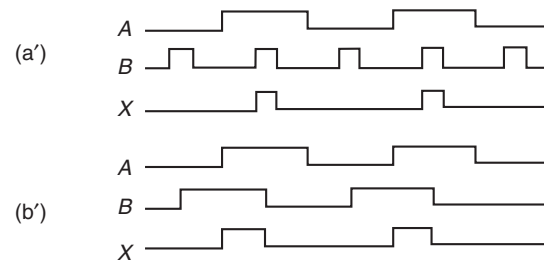


Fig. 4.9a

AND CIRCUITS

The pinout diagram of a 7408, TTL (transistor-transistor logic) Quad two-input AND gate is given in Fig. 4.10. This digital IC contains *four 2-input AND gates*. After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more AND gates to other TTL devices.

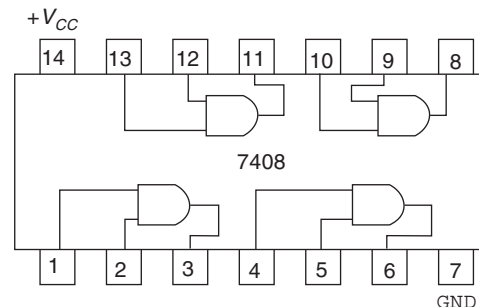


Fig. 4.10 Pinout Diagram of the 7408 Quad Two-input AND Gates.

4.12 Draw the circuit diagram of a relay AND circuit and explain its working.

Solution:

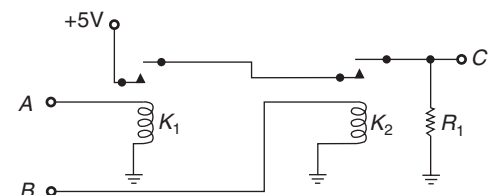


Fig. 4.11 Relay AND Circuit.

Input of +5 V applied to both points A AND B will cause relays K_1 and K_2 to energise, supplying +5 V to C, the output, via the closed relay contacts.

4.13 Draw the circuit diagram of a diode AND gate and explain its working.

Solution:

Point C will be +5 V if both inputs A AND B are at +5 V. If either A or B is at 0 V, the output will be 0 V (approximately). Assuming a silicon diode with a forward voltage drop of nominally 0.7 V, this has the effect of pulling the output C down to approximately 0.7 V. R_1 is called a common pull-up resistor tied to +5 V. It does not matter if more than one input is taken LOW, for in this situation the diodes with grounded cathodes simply share the current which is limited by the resistor. This is a simple logic gate. Most modern designs employ only positive logic.

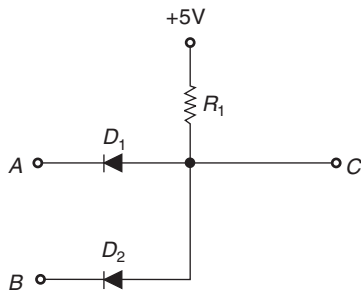


Fig. 4.12 Diode AND Circuit.

4.14 Draw the circuit diagram of a transistor AND gate and explain its working.

Solution:

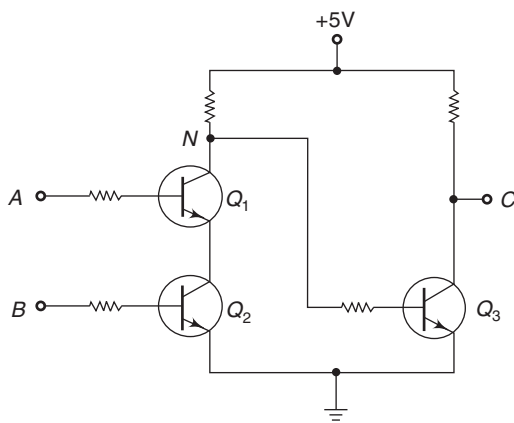


Fig. 4.13 *Transistor AND Gate.*

When both A AND B are at +5 V, transistors Q_1 and Q_2 conduct, moving point N to ground. This cuts-off Q_3 , driving point C to +5 V. However, if either A or B is at a ground level, either Q_1 or Q_2 will be cut-off sending

point N positive and providing base current to Q_3 . Thus, point C will go to ground. The introduction of transistors gives us an immense improvement, both logically and electrically.

4.15 Briefly describe an automobile's safety system incorporating an AND gate.

Solution:

The circuit in Fig. 4.14 shows an AND gate whose HIGH output activates a buzzer when three conditions are met on its inputs. When the ignition switch S_1 is ON, a HIGH is connected to the gate input A . When the seat belt is not properly buckled, switch S_2 is OFF and a HIGH is connected to the gate input B . At the instant the timer switch is turned ON, the timer is activated and produces a HIGH on gate input C . The resultant HIGH gate output activates the alarm. After a specified time, the timer's output goes LOW, disabling the AND gate and turning OFF the alarm. If the seat belt is buckled when the ignition is turned ON, a LOW is applied to input B , keeping the gate output LOW, thus preventing the alarm from sounding.

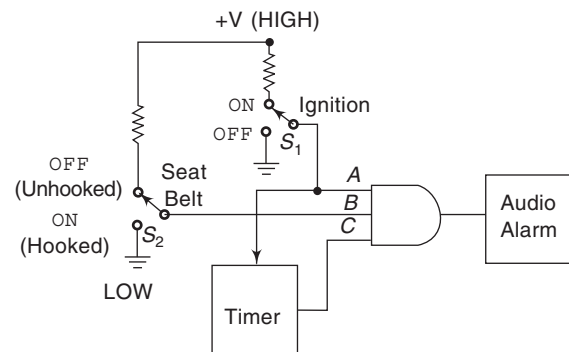


Fig. 4.14 *An Automobile's Safety System.*

4.16 Discuss a simple logic system, incorporating an AND gate, for control of an elevator motor.

Solution:

The circuit in Fig. 4.15 shows a simple logic system for the control of an elevator motor. The push-button switch

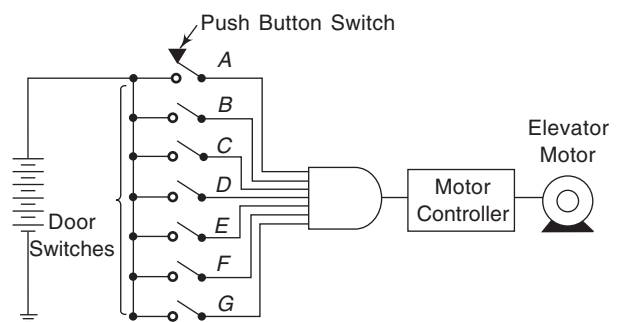


Fig. 4.15 *An Automobile's Safety System.*

starts the elevator motor only when all of the door switches are closed. When one or more door switches is open, HIGH levels are not present at every input of the AND gate, and the motor cannot be started. When all the door switches are closed, HIGH levels are applied to each input of the gate, with the exception of input A. Closing the push-button switch now provides a HIGH level at A. Thus, the AND gate output goes from LOW to HIGH (0 to 1), and the motor controller is energised to start the motor. If one or more of the door switches is open, the AND gate output remains LOW when the push-button switch is pressed, and the motor cannot be started.

ENABLE/INHIBIT FUNCTION OF AND GATE

A common application of the AND gate is to *enable* (allow) the passage of a signal (pulse waveform) from one point to another at certain times and to *inhibit* (prevent) the passage of the signal at other times.

4.17 How can a clock oscillator be enabled/disabled using an AND gate?

Solution:

The clock frequency of 1 MHz converts to 1 μ s for each clock period. To transmit the clock pulses, we have to provide an enable signal. The illustration (Fig. 4.16) shows the circuit and waveforms to enable four clock pulses. For the HIGH clock pulses to get through the AND gate to point X, the second input to AND gate (enable signal input) must be HIGH, otherwise the output of the AND gate will be LOW. Therefore, when the enable signal is HIGH for 4 μ s, four clock pulses pass through the AND gate. When the enable signal goes LOW, the AND gate disables (inhibits) any further clock pulses from reaching the receiving device.

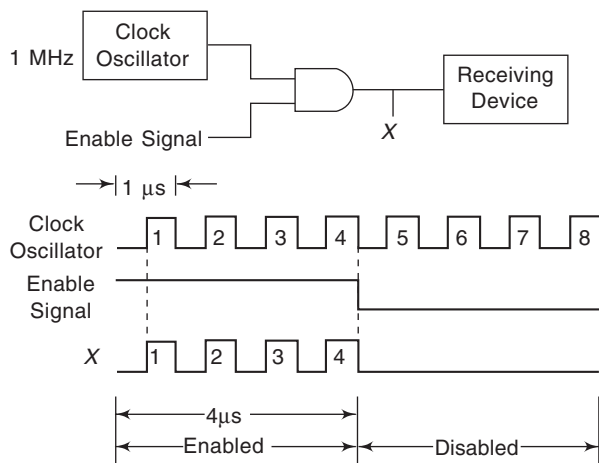


Fig. 4.16 Using an AND Gate to Enable/Disable a Clock Oscillator.

AND LAWS

There are three AND laws.

$$A \cdot 1 = A \quad A \cdot 0 = 0 \quad A \cdot A = A$$

All of these three AND laws can be verified by remembering what the AND symbol means.

4.18 Verify AND laws with the help of illustrations.

Solution:

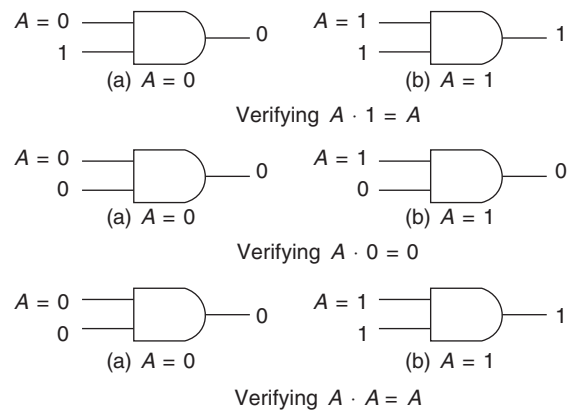


Fig. 4.17 Illustrating and Verifying AND Laws.

THE OR FUNCTION

If a situation which may be described with Boolean variables gives a desired result *only when any one or all* of several external conditions are satisfied, then that situation is said to obey the Boolean OR function. OR gates may have any number of inputs (Fig. 4.18).

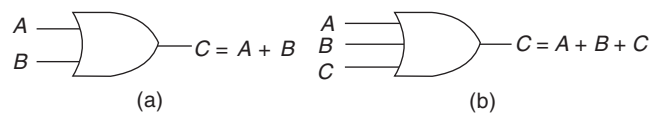


Fig. 4.18 (a) Two-input OR Gate. (b) Three-input OR Gate.

4.19 Represent the OR function by switch analogy.

Solution:

The switch analogy of OR function is shown in Fig. 4.19. The lamp will light if either A OR B is closed OR both A and B are closed.

Note: Compare Fig. 4.19 with Fig. 4.5.

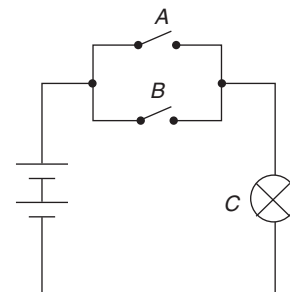


Fig. 4.19 Switch Analogy of OR Function.

4.20 Draw the truth table for a three-input OR gate.*Solution:*

The truth table for a three-input OR gate is given in Fig. 4.3(b).

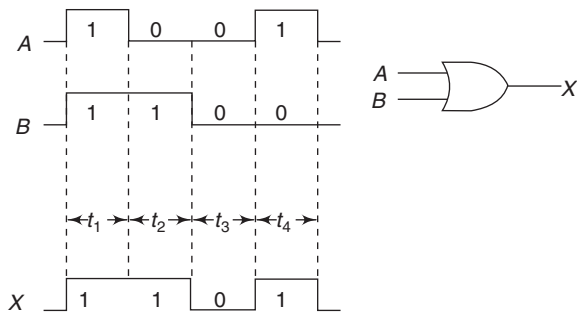
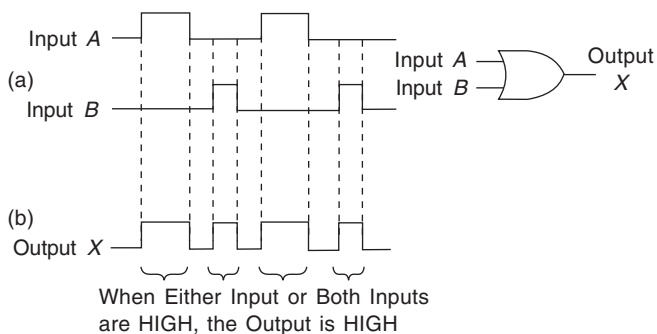
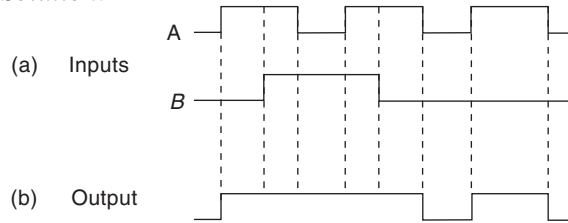
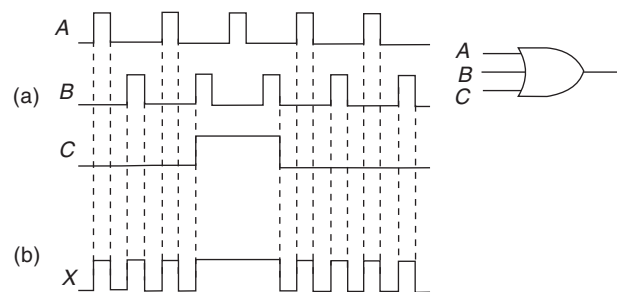
Note: Compare Fig. 4.3(b) with Fig. 4.3(c).

PULSED OPERATION

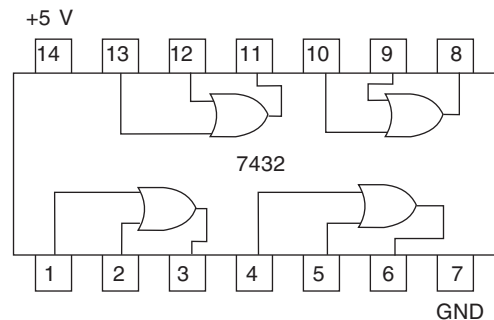
The important thing in pulsed operation is the mutual relationship of all the waveforms involved.

4.21 Analyze the pulsed operation of the OR gate in Fig. 4.20.*Solution:*

The input A and B are both '1' during interval t_1 , making the output '1'. During interval t_2 , input A is '0', but because input B is '1', the output is '1'. Both inputs are '0' during interval t_3 , the output is '0'. During time t_4 , the output is '1', because input A is '1'.

**Fig. 4.20** Pulsed Operation of OR Gate.**4.22 If the two waveforms A and B are applied to the OR gate in Fig. 4.21(a), what is the resulting output waveform?***Solution:***Fig. 4.21** The Output of a Two-input OR Gate is HIGH when Either or Both Inputs are HIGH.**4.23 For the two-input waveforms in Fig. 4.22(a), sketch the output waveform showing its proper relationship to the inputs for the two-input OR gate.***Solution:***Fig. 4.22** The Output is HIGH when any of the Inputs are HIGH.**4.24 For the three-input OR gate shown in Fig. 4.23, determine the output waveform in proper relationship to the inputs.***Solution:***Fig. 4.23** The Output is HIGH when any of the Inputs are HIGH.**OR CIRCUITS**

The pinout diagram of a 7432, TTL Quad 2-input OR gate is shown in Fig. 4.24. This digital IC contains 4 two-input OR gates inside a 14-pin dual-in-line package (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.

**Fig. 4.24** Pinout Diagram of the 7432 Quad 2-input OR Gates.**4.25 Draw the circuit diagram of a relay OR circuit and explain.***Solution:*

If either A OR B is +5 V, one of the relay contacts, which are wired in parallel, will close applying +5 V to point C . (See Fig. 4.25.)

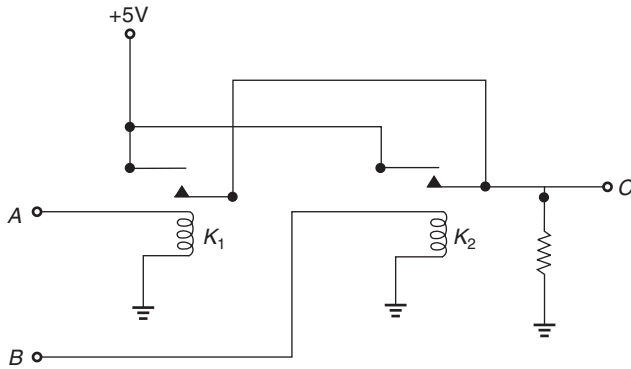


Fig. 4.25 Relay OR Circuit.

4.26 Draw the circuit diagram of a diode OR gate and explain its working.

Solution:

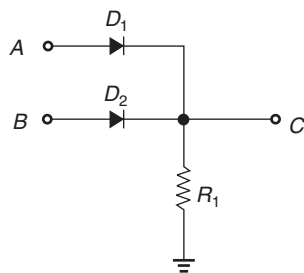


Fig. 4.26 Diode OR Circuit.

Applying +5 V to input A OR input B, OR both inputs A and B, will forward bias D_1 , OR D_2 , OR both D_1 and D_2 , causing point C to go to +5 V.

4.27 Draw the circuit of a transistor OR gate and explain its working.

Solution:

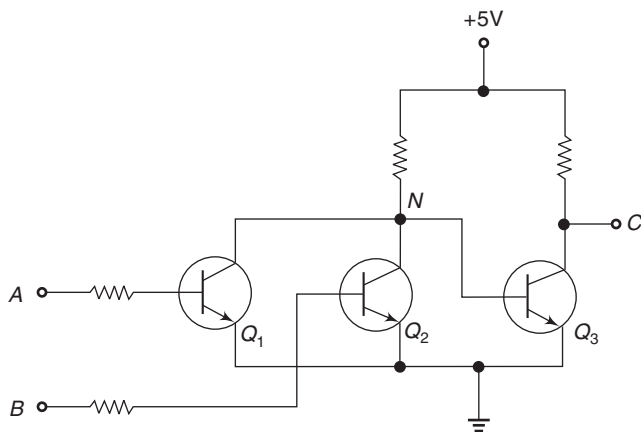


Fig. 4.27 Transistor OR Gate.

Applying +5 V to point A will cause Q_1 to conduct, causing point N to go to ground. This, in turn, will cut off Q_3 , causing point C to go to +5 V. Applying +5 V to point B will cause Q_2 to conduct, resulting in the output again going to +5 V. If both inputs are grounded

Q_1 and Q_2 will cut-off, causing point N to go positive, supplying current to the base of Q_3 . This results in input C going to ground. Therefore, this circuit satisfies the definition for the OR gate.

4.28 Draw the logic system for controlling a boiler used in a hot-water space heating system. The boiler is switched 'ON' when the air temperature falls below a predetermined level OR when the water temperature falls below a preset level.

Solution:

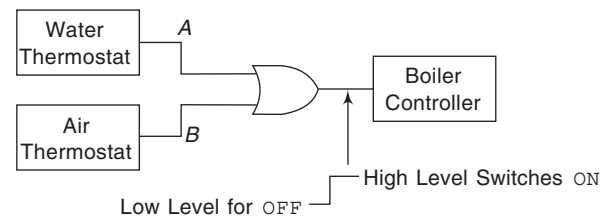


Fig. 4.28 Logic System for Controlling a Boiler used in a Water Heating System.

When the *air temperature* falls below a predetermined level, the air thermostat provides a HIGH output, which calls for the boiler to be switched 'ON'. Similarly, when the *water temperature* falls below a preset level, a HIGH output is produced by the water thermostat to switch the boiler 'ON'. The desired operation is achieved by the use of an OR gate, as illustrated, and a controller that switches the boiler 'ON' when the OR gate output is HIGH.

4.29 The boiler control circuit in problem 4.28 is to be modified to provide safety functions. The boiler is to switch 'OFF' if the water temperature exceeds a prescribed maximum. A second water thermostat is included to detect the upper temperature limit. This produces a HIGH output level while the water temperature is below the maximum. Also, the boiler is to switch 'OFF' if the quantity of water is below a minimum safe level. A water level transducer is included, and this produces a HIGH output while the water level remains above the minimum. Make the necessary circuit modification.

Solution:

The statement which describes the operation of the circuit is as follows:

The boiler starts when a HIGH input is provided at terminals A OR B.

When the two additional parts are included, the circuit statement must be modified as follows:

The boiler starts when a HIGH input is provided at terminals (A OR B) and at C AND D.

It is clear that an AND gate and an OR gate are required.

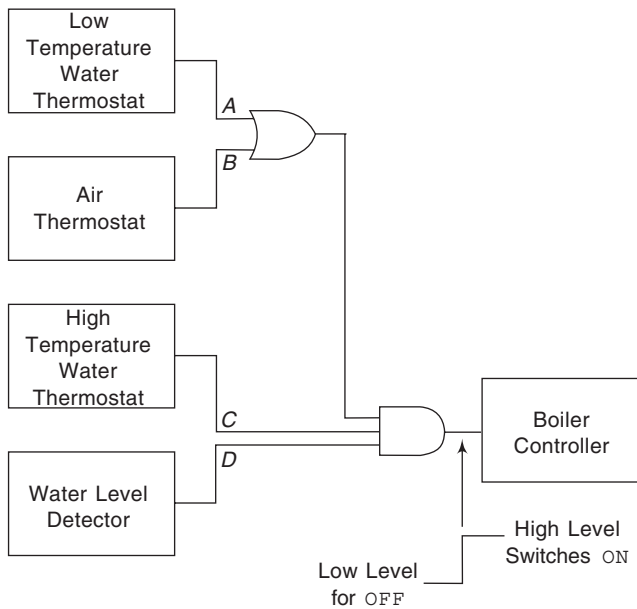


Fig. 4.29 Modification to the Boiler Control Circuit in Problem 4.29 to provide Safety Functions. The Boiler can be Switched 'ON' only if the Water Temperature remains below a prescribed Maximum and the Water Level remains above a Specified Minimum.

4.30 The elevator control circuit in problem 4.16 is to be modified to permit a maintenance technician to start the motor regardless of the condition of other switches. Determine the necessary modification.

Solution:

The motor starts when a HIGH input is provided at terminals (A AND B AND C AND D AND E AND F AND G) OR H. This statement shows that an AND gate and an OR gate are required. The modification is shown in Fig. 4.30.

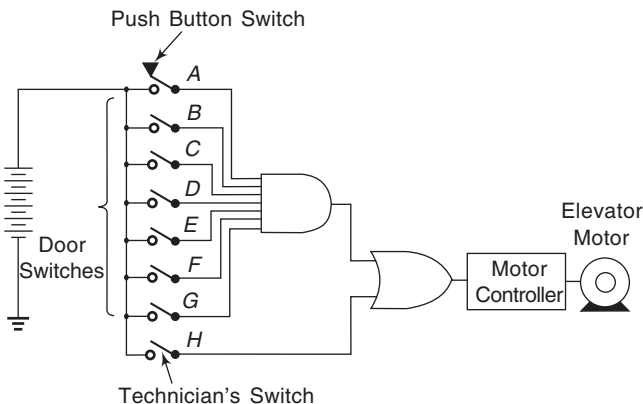


Fig. 4.30 Modification to the Motor Control Circuit in Problem 4.16 to Provide a Maintenance Technician Switch 'H' which Starts the Motor Regardless of the Condition of Other Switches.

Closing switch *H* provides a HIGH input to the OR gate, resulting in a HIGH input to the motor controller and the subsequent starting of the motor. The condition of the other switches has no effect on this operation. Also, with the switch left open, all of the other switches must be closed before the motor can start.

4.31 In a room with three doors, an indicator lamp must be turned 'ON' when any of the three doors is not completely closed.

Solution:

The sensors are switches that are open when a door is ajar, or open. This open switch creates the HIGH level for the OR gate input, as shown in Fig. 4.31. If any OR all of the doors are open, the gate output is HIGH. The HIGH level is then used to illuminate the indicator lamp. The gate is, of course, assumed to be capable of supplying sufficient current to the lamp.

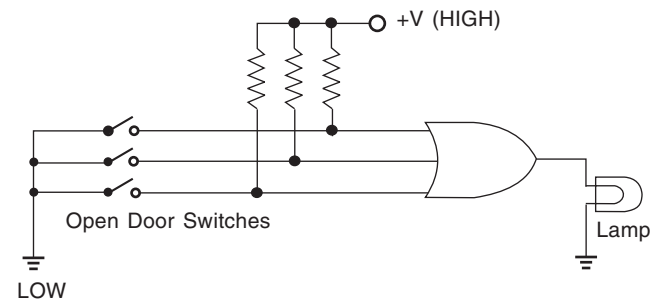


Fig. 4.31 An Example of an OR Gate Application.

ENABLE/INHIBIT FUNCTION OF OR GATE

An OR gate can also be used to *disable* a function. The only difference is that the enable input signal is made HIGH to disable and the output of the OR gate goes HIGH when it is disabled. This is shown in Fig. 4.32.

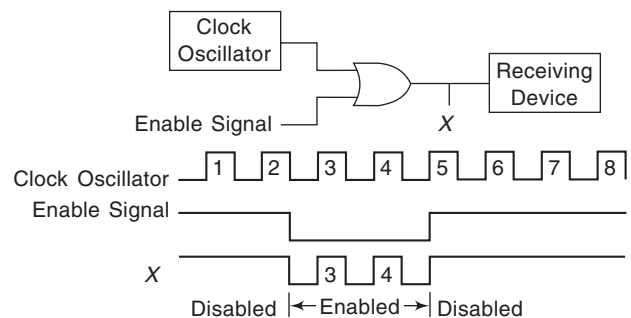


Fig. 4.32 Using an OR Gate to Enable/Disable a Clock Oscillator.

OR LAWS

There are three OR laws.

$$A + 1 = 1 \quad A + 0 = A \quad A + A = A$$

All of these OR laws can be verified remembering what the OR symbol means.

4.32 Verify OR laws with the help of illustrations.

Solution:

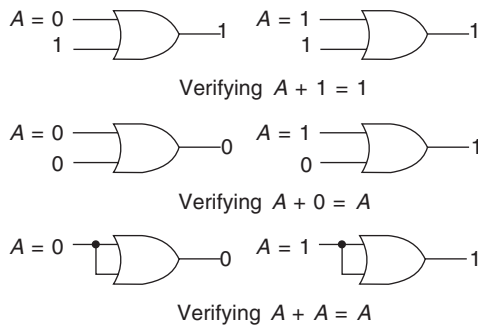


Fig. 4.33 Illustrating and Verifying OR Laws.

THE NOT FUNCTION

The NOT circuit performs a basic logic function called *negation* or *complementation*. It changes one logic level to the *opposite* logic level (negates the input). In terms of levels, it changes a HIGH level to a LOW level and a LOW level to a HIGH level. In terms of bits it changes a '1' to a '0' and a '0' to a '1'. Each statement is called the negation or *inverse* of the other. A logic gate that negates the input is called an inverter. Inversion is indicated by an overline \bar{A} (not A). The negation indicator is a bubble appearing on the *input* or *output* of a logic element. When appearing on the *input*, the bubble means that an external 0 produces an internal 1. When appearing on the *output*, the bubble means that an internal 1 produces an external 0. Typically, inputs are on the left of a logic symbol and outputs are on the right. The triangle, Fig. 4.34, represents an amplifier. When a single circuit is used for inversion alone the triangle is included with the symbol.

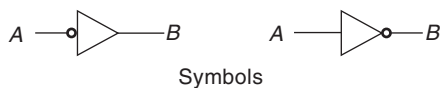


Fig. 4.34 The Inverter.

4.33 Discuss the significance of a polarity indicator.

Solution:

The polarity indicator is a triangle (Δ). When appearing on the input, it means that an external LOW level produces an internal HIGH level. When appearing on the output, it means that an internal HIGH level produces an external LOW level. The placement of the negation or polarity indicator does not imply a change in the way an inverter operates. Both indicators are equivalent and can be interchanged.



Fig. 4.35 The Polarity Indicator.

PULSED OPERATION

Inversion of a variable A gives \bar{A} . Conversely, inversion of \bar{A} gives A . A *double* inversion of a variable is thus equivalent to no inversion. This is shown in Fig. 4.36.

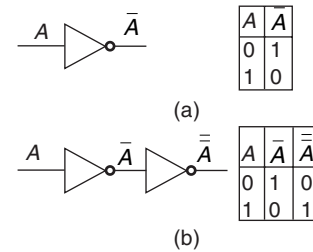


Fig. 4.36 (a) Single Inversion. (b) Double Inversion.

4.34 Illustrate the pulsed operation of an inverter.

Solution:

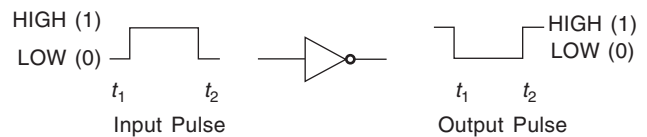


Fig. 4.37 Inverter with Pulsed Input.

4.35 Sketch the output waveform at X and Z if the timing waveform shown below is input at A.

Solution:

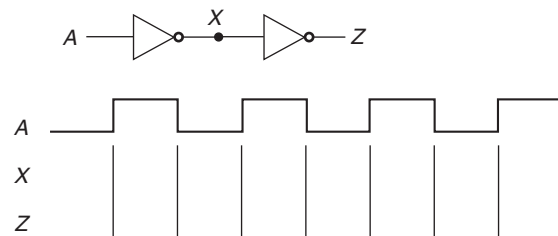


Fig. 4.38 Double Inversion.

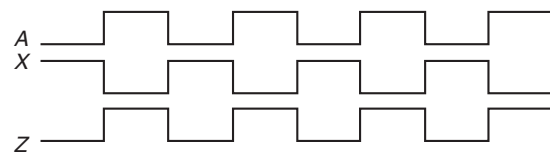


Fig. 4.39 Solution for 4.35.

INVERTER CIRCUITS

The 7404 is a TTL hex inverter. This integrated circuit (IC) contains six inverters. After applying +5 V to pin 14 and grounding pin 7, you can connect any or all of the inverters to other TTL devices. For instance, if you need only one inverter, you can connect an input signal to pin 1 and take the output signal from pin 2; the other five inverters can be left unconnected. (See Fig. 4.40)

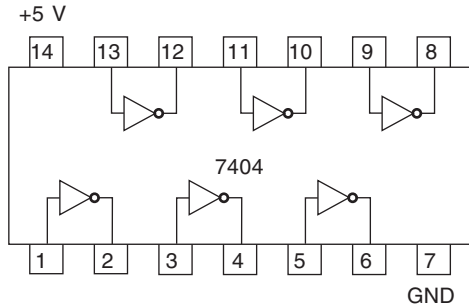


Fig. 4.40 Pinout Diagram of the 7404 Hex Inverter.

4.36 Draw the diagram of a relay inverter circuit. Explain.

Solution:

With +5 V applied to the relay circuit, the relay will energise, opening the *normally closed* contacts, causing 0 V to appear on the output lead B. Thus, a HIGH on the input is inverted (negated) to a LOW on the output.

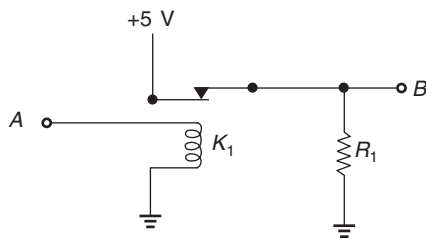


Fig. 4.41 Relay Inverter Circuit.

4.37 Draw the diagram of a transistor inverter circuit. Explain.

Solution:

In the transistor inverter circuit, +5 V applied to input A will turn the transistor 'ON', causing it to conduct, resulting in a ground level at point B. With a ground level applied to point A, the transistor will turn 'OFF', resulting in +5 V at point B.

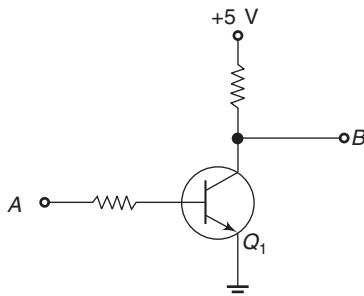


Fig. 4.42 Transistor Inverter Circuit.

NOT LAWS

There are several laws of Boolean algebra that become apparent when examining the inverter.

$$\begin{aligned} \bar{0} &= 1, \quad \bar{1} = 0, \quad \text{if } A = 0 \quad \text{then } \bar{A} = 1, \quad \text{if } A = 1 \\ \text{then } \bar{\bar{A}} &= 0, \quad \bar{\bar{\bar{A}}} = A \end{aligned}$$

4.38 Give the verbal statements of NOT laws.

Solution:

$\bar{0} = 1$ If a statement is not false, it must be true.

$\bar{1} = 0$ If a statement is not true, it must be false.

If $A = 0$ then $\bar{A} = 1$ If a statement is false, then the negation of that statement is true.

If $A = 1$ then $\bar{A} = 0$ If a statement is true, then the negation of that statement is false.

$\bar{\bar{A}} = A$ Double inversion of a variable is equivalent to no inversion.

4.39 Differentiate between even and odd number of negations.

Solution:

An *even* number of negations is equivalent to no negation.

An *odd* number of negations is equivalent to a single negation.

THE NAND FUNCTION

The term NAND is a contraction of *NOT-AND* and implies an AND function with a complemented (inverted) output.

The circle at the output acts just like an *inverter*. So a NAND gate can be drawn symbolically as an AND gate with an inverter connected to its output as shown in Fig. 4.43.

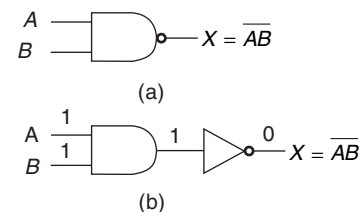


Fig. 4.43 (a) Symbol for NAND Gate.

(b) *AND-INVERT* Equivalent of a NAND Gate with $A = 1$ and $B = 1$.

4.40 Draw the truth table of NAND gate and write the Boolean equation for a two-input NAND gate.

Solution:

The Boolean equation for a NAND gate is written as $X = \overline{AB}$. The inversion bar is drawn over (A and B), meaning that the output of the NAND is the complement of (A AND B) i.e. [$\text{NOT } (A \text{ AND } B)$]. Because we are inverting the output, the truth table outputs will be the complements of the AND gate. Think of how an AND gate would respond to the inputs and then invert your answer (see Fig. 4.44). We can see that *the output is LOW when both inputs are HIGH* (just the opposite of AND gate). Also, *the output is HIGH when any or all inputs are LOW*.

A	B	X
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 4.44 Two-input NAND Gate Truth Table.

4.41 Draw the logic symbols of, and write the Boolean expressions for a three-input and an eight-input NAND gate.

Solution:

NAND gates can also have more than two inputs. Figure 4.45 shows the symbols and Boolean expressions for a three-input and an eight-input NAND gate.

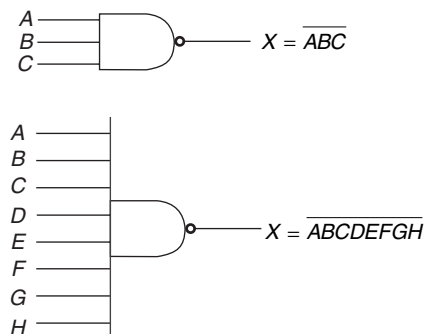


Fig. 4.45 Symbols and Boolean Expressions for a Three-input and an Eight-input NAND Gate.

4.42 A NAND gate is also referred to as a negative OR gate. Why?

Solution:

In Fig. 4.44, if A is LOW or B is low, or if both A and B are LOW then X is HIGH. Here we have an OR operation that requires one or more LOW inputs to produce a HIGH output. This is referred to as *negative OR*. When a NAND gate is looking for one or more LOWs on its inputs, rather than for all HIGHS, it is acting as a negative OR, Fig. 4.46(b). The two symbols in Fig. 4.46 represent the same gate, but they also serve to define its role in a particular application.

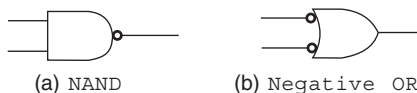


Fig. 4.46 Standard Symbols Representing the Two Equivalent Functions of the NAND Gate.

- Note:*
1. In a NAND gate, the LOW level is the active output level. The bubble on the output indicates that the output is active 0.
 2. The unique output from a NAND gate is LOW only when *all* inputs are HIGH.

3. The *universality* of the NAND gate means that logic systems incorporating many different functions may be designed with only a single type of gate.

4.43 Discuss the implementation of NOT, AND, or OR gates by NAND gates.

Solution:

The *NOT* operation is obtained from a one-input NAND gate, actually another symbol for an inverter circuit. The *AND* operation requires two NAND gates. The first NAND gate produces an inverted AND and the second NAND gate acts as an inverter to produce the normal output. The *OR* operation is achieved through a NAND gate with additional inverters in each input. All the three operations are shown in Fig. 4.47.

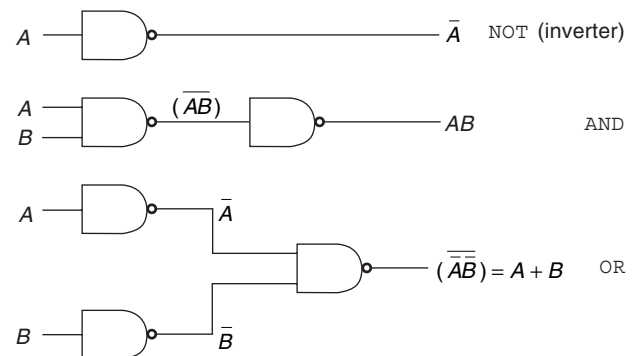


Fig. 4.47 Universality of the NAND Gate.

PULSED OPERATION

In a NAND gate, the only time a LOW output occurs is when all inputs are HIGH.

4.44 Sketch the output waveform at X for the NAND gate in Fig. 4.48.

Solution:

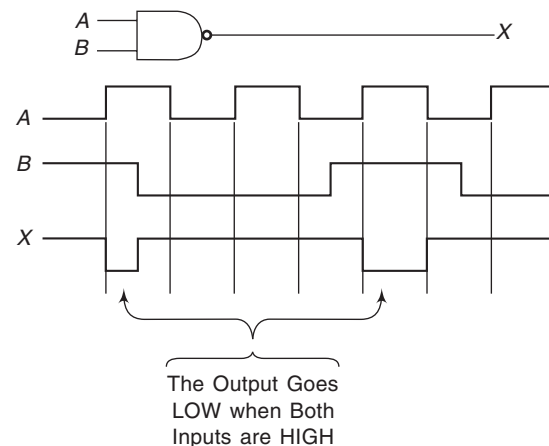


Fig. 4.48 Timing Analysis of a NAND Gate.

4.45 Sketch the output waveform at X for the NAND gate shown in Fig. 4.49(a) with the given input waveforms at A , B and control in Fig. 4.49(b).

Solution:

The control input waveform is used to enable/disable the NAND gate. When it is LOW, the output is stuck HIGH. When it is HIGH, the output will respond LOW when A and B go HIGH.

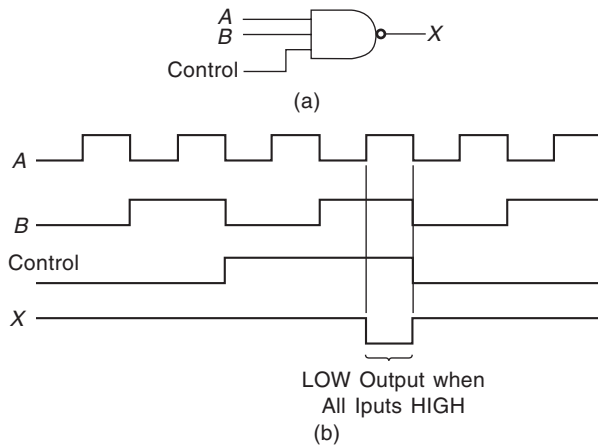


Fig. 4.49 Timing Analysis of a NAND Gate with a Control Input.

NAND CIRCUITS

The 7400 IC contains four 2-input NAND gates. After connecting a supply voltage of +5 V to pin 14 and a ground to pin 7, you can connect one or more NAND gates to other TTL devices.

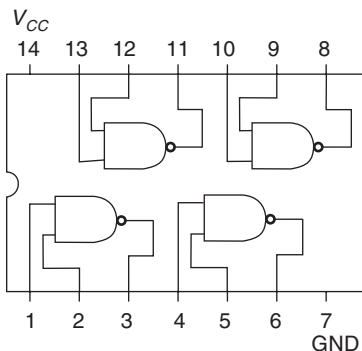


Fig. 4.50 Pinout Diagram of the 7400 Quad 2-input NAND Gates.

4.46 Draw the external connections to a 4011 CMOS IC to form the circuit shown in Fig. 4.51.

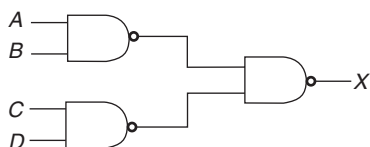


Fig. 4.51

Solution:

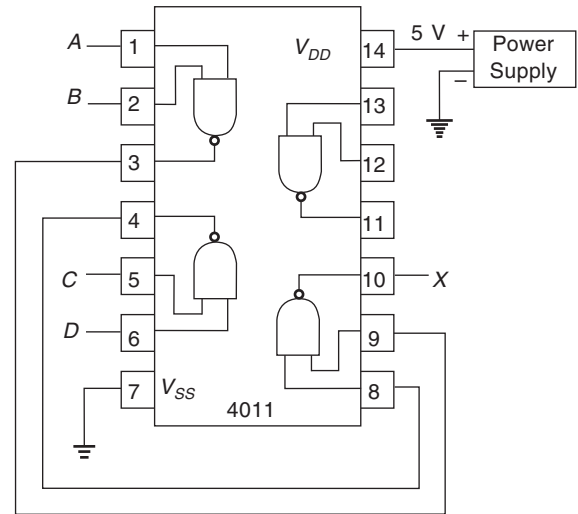
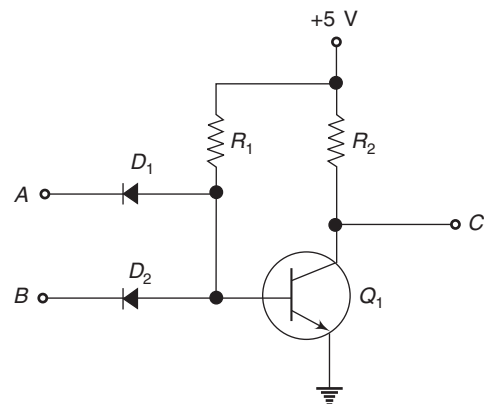


Fig. 4.52 Solution for 4.46. $+V_{DD}$ is connected to +5 V supply and V_{SS} to ground. According to CMOS manual, V_{DD} can be any Positive Voltage from +3 to +15 V with respect to V_{SS} (Usually Ground).

4.47 Draw a transistor NAND gate circuit and explain.

Solution:

The 1's indicate +5 V and the 0's ground. The output C is LOW only when A and B are both 1.



A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

Fig. 4.53 A Transistor NAND Gate Circuit Along with its Associated Truth Table.

Note: A NAND gate is a purchasable piece of hardware, usually an integrated circuit. This piece of hardware can perform all three functions: AND, OR, and Invert.

4.48 The simultaneous occurrence of two HIGH level voltages must be detected and indicated by a LOW level output that is used to illuminate a LED. Sketch the operation.

Solution:

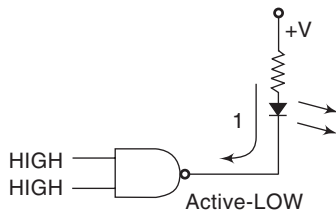


Fig. 4.54

The application requires a NAND function, since the output must be *active-LOW* in order to produce current through the LED when the two HIGHS occur on its outputs. The NAND symbol is, therefore, used to show the operation.

THE NOR FUNCTION

The term NOR is a contraction of *NOT-OR* and implies an OR function with a complemented (inverted) output. The circle at the output acts just like an *inverter*. So a NOR gate can be drawn symbolically as an OR gate with an inverter connected to its output as shown in Fig. 4.55.

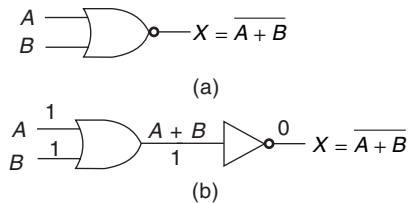


Fig. 4.55 (a) Symbol for NOR gate.
(b) OR-INVERT equivalent of a NOR gate with $A = 1$ and $B = 1$.

4.49 Draw the truth table of and write the Boolean equation for a two-input NOR gate.

Solution:

The Boolean equation for a NOR gate is written as $X = \overline{A + B}$. The inversion bar is drawn over $(A + B)$, meaning that the output of the NOR is the complement of $(A$ or $B)$ i.e. $[NOT (A + B)]$. Because we are inverting the output, the truth table output will be the complement of the OR gate truth table output. Think of how an OR gate would respond to the inputs and then invert your answer (see Fig. 4.56). We can see that *the output is HIGH when both inputs are LOW* (just the opposite of OR gate). Also, *the output is LOW when any or all inputs are HIGH*.

4.50 Draw the logic symbols of, and write the Boolean expressions for a three-input and an eight-input NOR gate.

A	B	X
0	0	1
0	1	0
1	0	0
1	1	0

Fig. 4.56 Two-input NOR Gate Truth Table.

Solution:

NOR gates can also have more than two inputs. Figure 4.57 shows the symbols and Boolean expressions for a three-input and eight-input NOR gate.

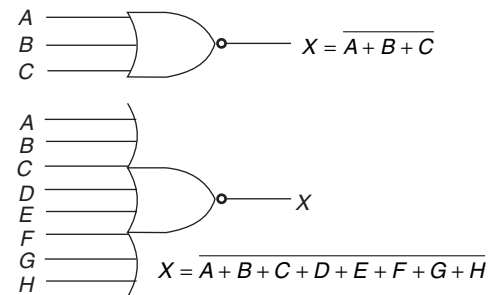


Fig. 4.57 Symbols and Boolean Expressions for a Three-input and Eight-input NOR Gate.

4.51 A NOR gate is also referred to as a 'Negative AND gate'. Why?

Solution:

In Fig. 4.56, if both A and B are LOW then X is HIGH. Here we have an AND operation that requires all LOW inputs to produce a HIGH output. This is referred to as *negative AND*. When a NOR gate is looking for all LOWs on its inputs, rather than one or more HIGHS, it is acting as a negative AND, Fig. 4.58(b). Also, the output is LOW when any of the inputs A and B are HIGH. The two symbols in Fig. 4.58 represent the same gate, but they also serve to define its role in a particular application.

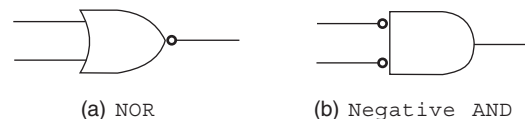


Fig. 4.58 Standard Symbols Representing the Two Equivalent Functions of the NOR Gate.

- Note:*
1. In a NOR gate, the low level is the active output level. The bubble on the output indicates that *the output is active 0*.
 2. The unique output from a NOR gate is HIGH only when *all* inputs are LOW.
 3. The *universality* of the NOR gate means that logic systems incorporating many different functions may be designed with only a single type of gate.

4.52 Discuss the implementation of NOT, OR, or AND gates by NOR gates.

Solution:

The *NOT* operation is obtained from a one-input NOR gate, yet another symbol for inverter circuit. The *OR* operation requires two NOR gates. The first NOR gate produces an inverted OR and the second acts as an inverter to obtain the normal output. The *AND* operation is achieved through a NOR gate with additional inverters at each input. All the three operations are shown in Fig. 4.59.

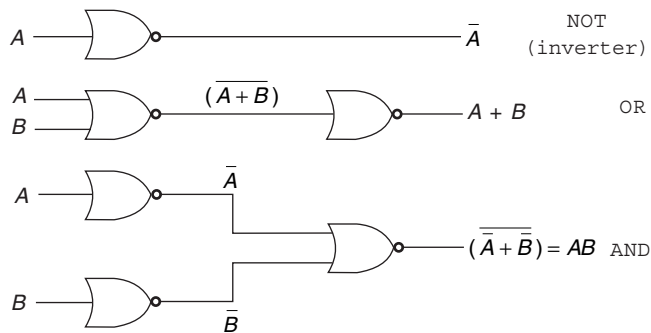


Fig. 4.59 Universality of the NOR Gate.

PULSED OPERATION

Again, as with other types of gates, we will simply follow the truth table operation to determine the waveforms in the proper time relationship to the inputs.

4.53 Sketch the waveforms at *X* and *Y* with the switches in the 'down' (0) position. Repeat the problem with the switches in the 'up' (1) position.

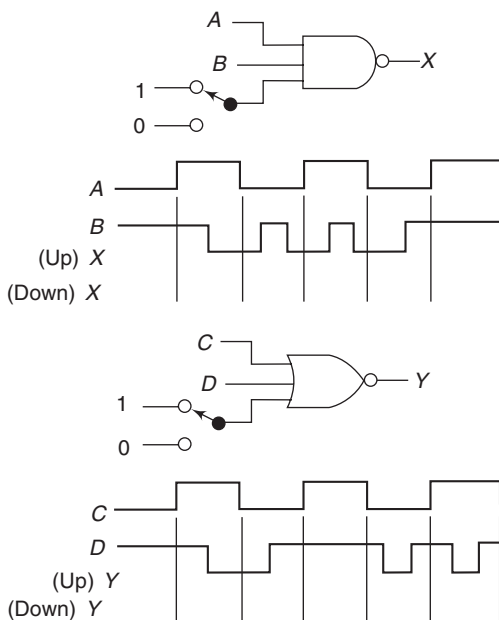


Fig. 4.60

Solution:

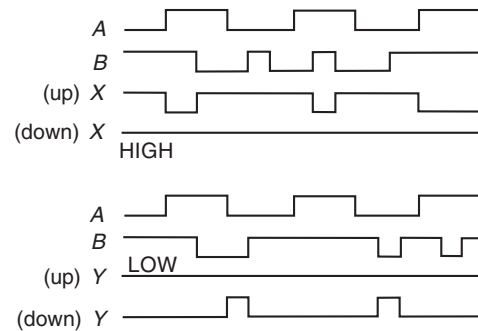


Fig. 4.61 Solution for 4.53.

4.54 Sketch the output at *X* and *Y* in Fig. 4.62, given the input waveforms.

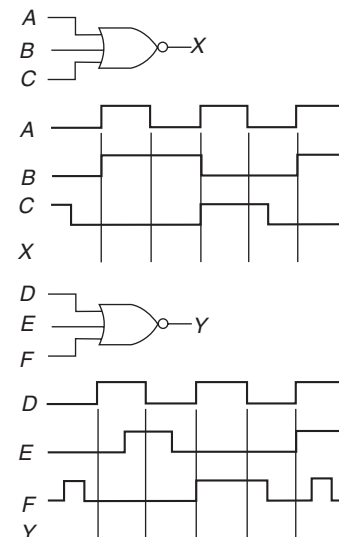


Fig. 4.62

Solution:

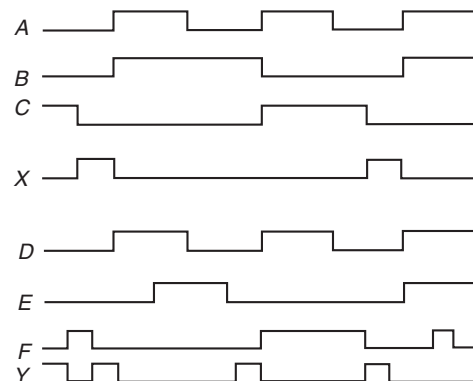


Fig. 4.63 Solution for 4.54.

NOR CIRCUITS

The most common ECL type is designated as the 10,000 series. The 10102 provides four 2-input NOR gates. An

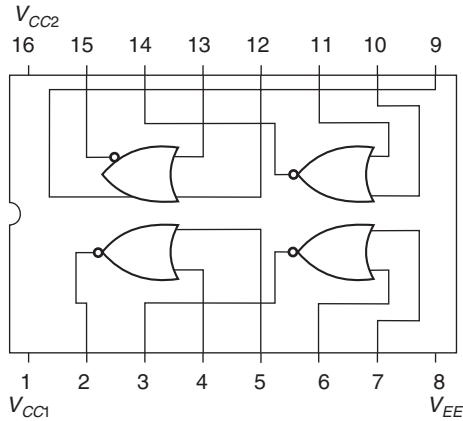


Fig. 4.64 Pinout Diagram of (Emitter-coupled Logic) (ECL) 10102 Quad 2-input NOR Gates.

ECL gate may have two outputs, one for the NOR function and another for the OR function (pin 9 of the 10102 IC).

4.55 Draw the pinout diagram of CMOS 4002. Explain.

Solution:

CMOS circuits of the 4002 can accommodate only two 4 input NOR gates because of pin limitations. The IC has two unused terminals marked NC (no connection). The terminal marked V_{DD} requires a power supply from 3 to 15 V, while V_{SS} is connected to ground.

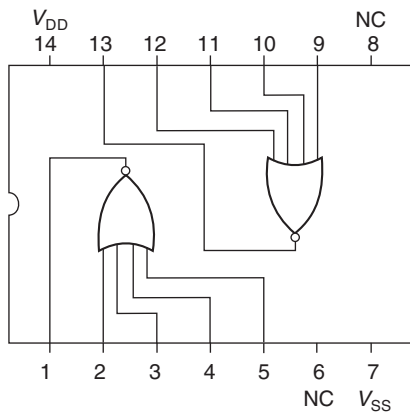


Fig. 4.65 Pinout Diagram of CMOS 4002.

4.56 Draw the circuit diagram of a diode-transistor NOR gate. Explain.

Solution:

A diode-transistor NOR gate is shown in Fig. 4.66. When all of the inputs are LOW, transistor Q_1 is 'OFF' and the output is HIGH. A HIGH input at terminal A OR B OR C biases the transistor 'ON' and produces a LOW output.

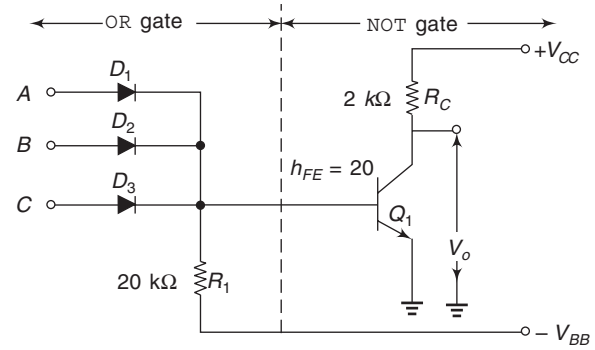


Fig. 4.66 Diode-transistor NOR Gate, consisting of an OR Gate and an Inverter Stage, or NOT Gate. A LOW Output is produced When HIGH Inputs are Present at Terminals A OR B OR C.

4.57 A gate is required to monitor two lines and to generate a HIGH level output used to activate an electric motor whenever either or both lines are LOW. Sketch the operation.

Solution:

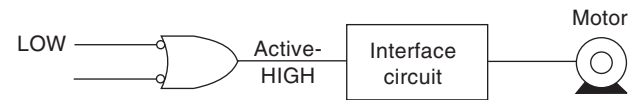


Fig. 4.67

This application requires an active-LOW input OR function because the output has to be active-HIGH in order to produce an indication of the occurrence of one or more LOW levels on its inputs. In this case, the gate functions as a negative-OR and is represented by the appropriate symbol shown in the diagram. A LOW on either input or both inputs causes an active-HIGH output to activate the motor through an appropriate interface circuit.

4.58 A certain application requires that two lines be monitored for the occurrence of a HIGH level voltage on either or both lines. Upon detection of a HIGH level, the circuit must provide a LOW voltage to energise a particular indicating device. Sketch the operation.

Solution:

The application requires a NOR function, since the output must be active-LOW in order to give an indication of at least one HIGH on its inputs. The NOR symbol is, therefore, used to represent the operation as shown in Fig. 4.68.

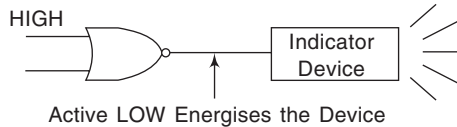


Fig. 4.68

4.59 What type of gate should be used to detect if all three landings gears are retracted after take off, assuming a LOW output is required to activate an LED display?

Solution:

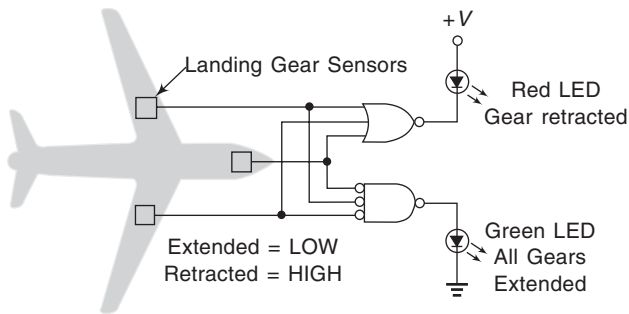


Fig. 4.69 A Part of an Aircraft's Functional Monitoring System.

Power is applied to the circuit only when the 'gear-down' switch is activated. Use a NOR gate for each of the two requirements as shown in Fig. 4.69. One NOR gate operates as a negative-AND to detect a LOW from each of the three landing gear sensors. When all three of the gate inputs are LOW, the three landing gears are properly extended and the resulting HIGH output from the negative-AND gate turns on the green LED display. The other NOR gate operates as a NOR to detect if one or more of the landing gears remain retracted when the 'gear down' switch is activated. When one or more of the landing gears remain retracted, the resulting HIGH from the sensor is detected by the NOR gate, which produces a LOW output to turn on red LED warning display.

THE EXCLUSIVE OR FUNCTION

The exclusive OR gate is sometimes referred to as the *any but not all gate*. The term 'exclusive OR gate' is often shortened to 'XOR gate'. The logic symbol for the XOR gate is shown in Fig. 4.70(a), the Boolean expression for the XOR function is shown in Fig. 4.70(b). The symbol \oplus means that the terms are XORed together. Notice that if any but not all of the inputs are 1, then the output will be a binary or logical 1.

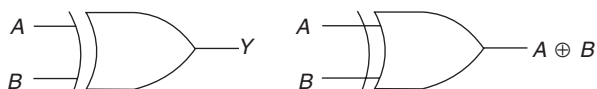


Fig. 4.70 (a) XOR Gate Symbol, (b) Boolean Expression.

4.60 Compare the truth tables for the OR gate and XOR gate.

Solution:

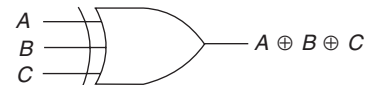
INPUTS		OUTPUT	
B	A	OR	XOR
0	0	0	0
0	1	1	1
1	0	1	1
1	1	1	0

Fig. 4.71 Comparison of Truth Tables for OR and XOR Gates.

4.61 What is the unique characteristic of the XOR gate?

Solution:

The unique characteristic of the XOR gate is that it produces a HIGH output only when an *odd* number of HIGH inputs are present. If an *even* number of HIGH inputs to the XOR gate are present the output will be LOW. This is shown in Fig. 4.72. The XOR gates are used in a variety of arithmetic circuits.



(a)

3-input XOR

INPUTS			OUTPUT
C	B	A	Y
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(b)

Fig. 4.72 (a) Three-input XOR Gate Symbol and Boolean Expression (b) Truth Table.

XOR CIRCUITS

ECL-10107 IC provides three XOR gates. There are two outputs from each gate; the other output gives the XNOR function equivalence. ECL gates have three terminals for power supply. V_{CC1} and V_{CC2} are usually connected to ground and V_{EE} to a -5.2 V supply. (See Fig. 4.73)

XOR and XNOR gates are available in both TTL and CMOS integrated-circuit packages. The 7486 is a TTL quad XOR and the 4077 is a CMOS quad XNOR.

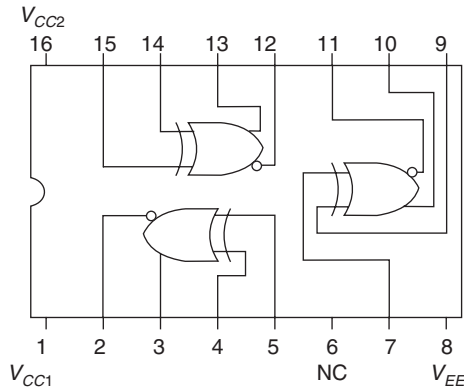


Fig. 4.73 Pinout of ECL-10107 Triple Exclusive OR-NOR Gate.

4.62 Draw the combination of AND, OR, and NAND gates to provide the XOR function.

Solution:

The combination of AND, OR, and NAND gates shown in Fig. 4.74 will reduce to the *one or other but not both*, XOR function.

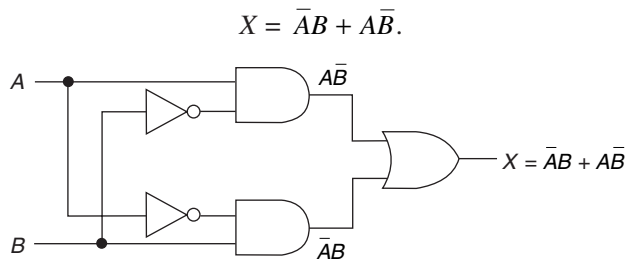


Fig. 4.74

4.63 Draw the switch analogy of the XOR function.

Solution:

Practically everyone who lives in a two-storeyed house uses an exclusive-OR gate every day. This consists of the switches at the top and bottom of the stairs that operate the *landing light*. When *both* switches are in the 'up' position, the lamp is not lit. If the switch at the *bottom* of the stairs is put on the 'down' position then the lamp will light. If the switch at the *top* of the stairs is also put in the 'down' position, the lamp will extinguish. Conversely, if *both* switches are in the 'down' position, the lamp will not be lit. *Putting one switch in the 'up' position and the other switch in the 'down' position will only light the lamp* (see Fig. 4.75)

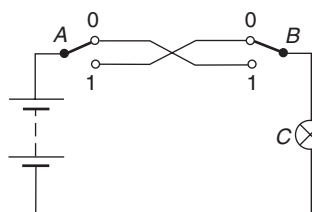


Fig. 4.75 Switch Analogy of the XOR Function.

This can be described by the following statement: the lamp is lit only when the switches are in *different positions* or *X is true if A is true or B is true but not both*. This can be rephrased as: *X is true if A is true and B is false OR if B is true and A is false*, which can be written as the Boolean equation: $X = \bar{A}B + A\bar{B}$.

4.64 Exclusive-OR circuits used in Digital Comparator.

Solution:

Unlike the radio receiver, a digital system generally exhibits no *natural symptoms*—such as hum, distortion, or erratic volume—to warn the operator that something has gone wrong. Instead, the system simply provides the *wrong answer*. Furthermore, component failure is not the only way by which information can be made erroneous or be lost. There is some distortion each time a pulse passes from one circuit to another. Added to this is the distortion caused by the inductive effects from nearby circuits. Thus, it should be understood that *a pulse itself can sometimes become so distorted that it may produce an error*.

One method of detecting errors, which may be called a *redundancy method*, involves running the problem through the computer *twice* and noting whether the solutions are identical. However, a computer's major function is to save time, and this type of check doubles the amount of time that is required to solve a problem. In addition, this check is useless if a component has failed; for *the faulty component will probably distort both solutions in exactly the same manner*.

A better solution is to provide *two sets of identical circuitry*, as shown in Fig. 4.76. When this is done, *the same problem is run simultaneously through both circuits*. Unless both circuits commit exactly the same error, obtaining identical solutions indicates that the answer is correct. Recall that *an exclusive-OR circuit produces an output only when its two inputs are different*.

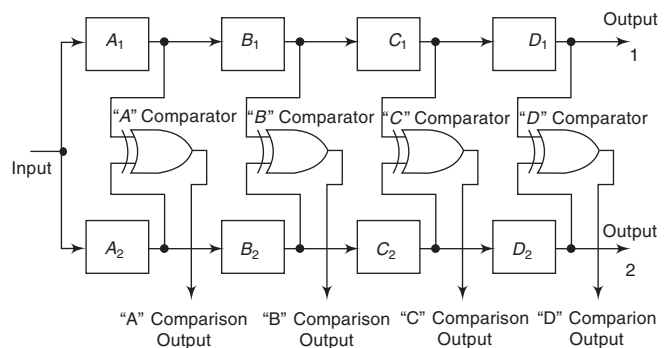


Fig. 4.76 Exclusive-OR Circuits used as Comparator in Error Detection.

When using this method, the inputs should always be the same. A signal at the output of any of the exclusive-OR circuits indicates that the two parallel

major section outputs are not identical and that one of them is in error. A signal at B , for example, means that either section B_1 or section B_2 is in error.

The *cost factor* must be considered. While parallel operation is an excellent technical concept and its use requires no extra time it does double the circuitry costs.

THE EXCLUSIVE NOR FUNCTION

The 'exclusive NOR gate' is often shortened to 'XNOR gate'. The logic symbol for the XNOR gate and the Boolean expression for the XNOR gate are given in Fig. 4.77. It is the XOR symbol with the added invert bubble on the output side. The bar over $A \oplus B$ expression tells us we have inverted the output of the XOR gate. When the two input logic levels are opposite, the output of the XNOR gate is LOW.

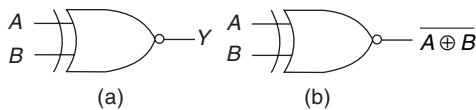


Fig. 4.77 (a) XNOR Gate Symbol (b) Boolean Expression.

4.65 Compare the truth tables of XOR and XNOR gates.

Solution:

$X = AB + \bar{A}\bar{B}$			$X = \bar{A}B + A\bar{B}$		
A	B	X	A	B	X
0	0	1	0	0	0
0	1	0	0	1	1
1	0	0	1	0	1
1	1	1	1	1	0

Exclusive-NOR Exclusive-OR

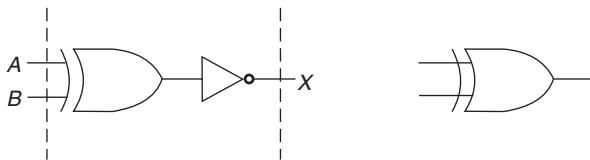


Fig. 4.78

Note: The output of the XNOR gate is the *complement* of the output of the XOR gate.

PULSED OPERATION

Under pulsed input conditions, we apply the truth table operation during each distinct time interval of the pulsed inputs.

4.66 Find the output X with the inputs A and B for the XOR gate in Fig. 4.79.

Solution:

Input waveforms A and B are at opposite levels during time intervals t_2 and t_4 . Therefore, the output X is HIGH

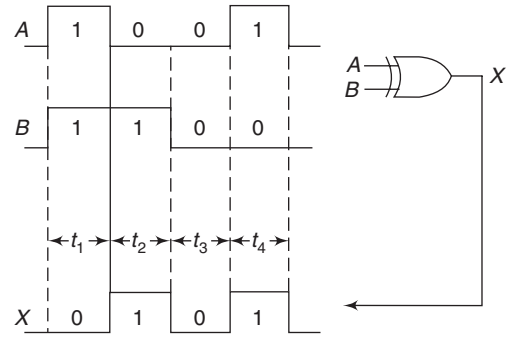


Fig. 4.79 Pulsed XOR Gate Operation.

during these two times. Since, both inputs are at the same level, either both HIGH or both LOW, during time intervals t_1 and t_3 , the output is LOW during those times. This is illustrated in Fig. 4.79.

4.67 Determine the output waveforms for the XOR gate and the XNOR gate, given the input waveforms, A and B , in Fig. 4.80.

Solution:

The output waveforms are shown in Fig. 4.80. The XOR output is HIGH only when both inputs are at *opposite* levels. The XNOR output is HIGH only when both inputs are the same.

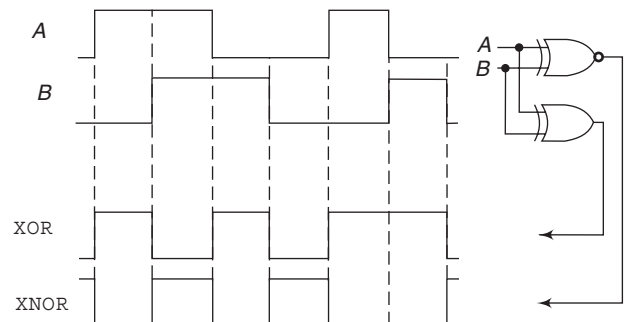


Fig. 4.80 Pulsed XOR and XNOR Gate Operation.

XNOR CIRCUITS

The XNOR gate provides a HIGH output for both inputs HIGH or both inputs LOW (Fig. 4.81).

$$X = AB + \bar{A}\bar{B}$$

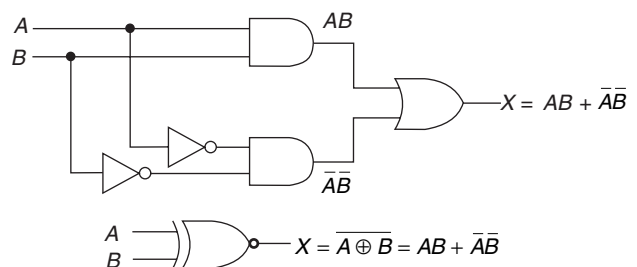


Fig. 4.81 XNOR Function Using AND, OR and NAND Gates.

4.68 How will you use an XOR gate as a two-bit adder?

Solution:

The output of the XOR gate is the *binary sum* of the two input bits. In the case where the inputs are both 1's, the output is the sum 0, but you lose the carry of 1. XOR gates are *combined* to make complete adding circuits.

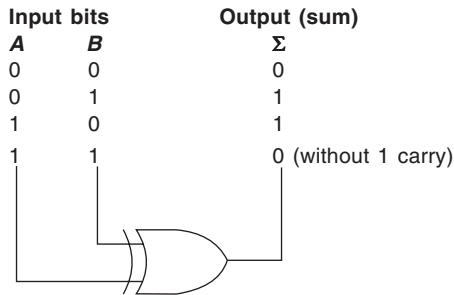


Fig. 4.82 An XOR Gate as a Two-bit Adder.

4.69 A certain system contains two identical circuits operating in parallel. As long as both are operating properly, the outputs of both circuits are always the same. If one of the circuits fails, the outputs will be at opposite levels at

sometime. Devise a way to detect that a failure has occurred in one of the circuits.

Solution:

The outputs of the circuits are connected to the inputs of an XOR gate as shown in Fig. 4.83. A failure in either one of the circuits produces differing outputs, which cause the XOR inputs to be at opposite levels. This condition produces a HIGH on the output of the XOR gate, indicating a failure in one of the circuits.

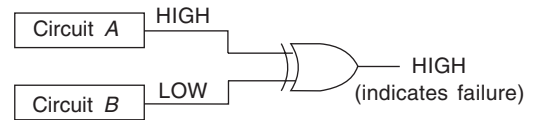


Fig. 4.83

4.70 Give a summary of the basic logic gates in tabular form.

Solution:

See Fig. 4.84.

4.71 How will you perform gate inversions using inverters. Illustrate.

Solution:

See Fig. 4.85.

Logic Function	Logic Symbol	Boolean Expression	Truth Table	
AND		$A \cdot B = Y$	Inputs	Output
			B A	Y
			0 0	0
			0 1	0
			1 0	0
OR		$A + B = Y$	1 1	1
			0 0	0
			0 1	1
Inverter		$A = \bar{A}$	1 0	1
			1 1	1
NAND		$\overline{A \cdot B} = Y$	0 0	1
			0 1	1
			1 0	1
			1 1	0
NOR		$\overline{A + B} = Y$	0 0	1
			0 1	0
			1 0	0
			1 1	0
XOR		$A \oplus B = Y$	0 0	0
			0 1	1
			1 0	1
			1 1	0
XNOR		$\overline{A \oplus B} = Y$	0 0	1
			0 1	0
			1 0	0
			1 1	1

Fig. 4.84 Summary of Basic Logic Gates.

Invert Outputs		$=$	
		$=$	
		$=$	
		$=$	
Invert Inputs		$=$	
		$=$	
		$=$	
		$=$	
Invert Inputs and Outputs		$=$	
		$=$	
		$=$	
		$=$	

Fig. 4.85 Gate Conversions using Inverters. The + Symbol here indicates Combining the Functions.

4.72 Briefly explain and compare traditional and IEEE logic gate symbols.

Solution:

See Fig. 4.86.

The *traditional logic gate symbols* are recognised by all workers in the electronics industry. These symbols are very useful in that they have *distinctive shapes*. Manufacturers' data manuals include traditional logic symbols and are recently including the newer *IEEE functional logic gate symbols*. These newer *IEEE* symbols are commonly referred to as *dependency notation*. For simple gating circuits, the traditional logic symbols are probably preferred but the IEEE standard symbols have advantages as ICs become more complicated. Most military contracts call for use of IEEE standard symbols.

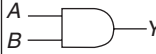
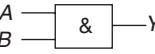
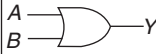
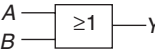
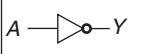
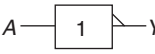
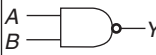
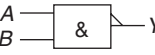
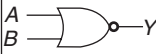
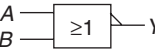
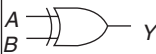
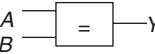
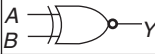
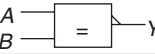
Logic Function	Traditional Logic Symbol	IEEE Logic symbol*
AND		
OR		
NOT		
NAND		
NOR		
XOR		
XNOR		

Fig. 4.86 Comparing Traditional and IEEE Logic Gate Symbols.

SUMMARY

- A logic gate is a digital circuit with one or more inputs but only one output. The output is HIGH only for certain combinations of input signals.
- Binary 0 for low voltage and binary 1 for high voltage is called positive logic.
- Binary 0 for high voltage and binary 1 for low voltage is called negative logic.
- A non-inverting buffer has no logical function.
- A truth table shows all of the input-output possibilities of a logic circuit.
- An AND gate produces a HIGH output only when all inputs are HIGH.
- A timing diagram illustrates graphically how the output levels change in response to input changes.
- The 7408 is a Quad 2-input AND gate.
- An AND gate enables the passage of signal at certain times and inhibits the passage of signal at other times.
- There are three AND laws: $A \cdot 1 = A$, $A \cdot 0 = 0$, and $A \cdot A = A$.
- An OR gate produces a HIGH output if any input is HIGH.
- The 7432 is a Quad 2-input OR gate.
- An OR gate can be used to disable a function.
- There are three OR laws: $A + 1 = 1$, $A + 0 = A$, and $A + A = A$.
- An inverter is a gate with only one input and a complemented output.
- The 7404 is a hex inverter.
- There are five NOT laws: $\overline{0} = 1$, $\overline{1} = 0$, if $A = 0$ the $\overline{A} = 1$, if $A = 1$ then $\overline{A} = 0$, and $\overline{\overline{A}} = A$
- The NAND gate represents an AND gate followed by an inverter.
- A NAND gate is also referred to as a negative OR gate.
- In a NAND gate, the LOW level is the active output level.
- The unique output from a NAND gate is LOW only when all its inputs are HIGH.
- A NAND gate is a purchasable piece of hardware, usually an integrated circuit.
- NOT, OR, or AND gates can be implemented using only NAND gates.
- The 7400 is a Quad 2-input NAND gate.
- The 4011 is a Quad 2-input NAND gate.
- The NOR gate represents an OR gate followed by an inverter.
- The NOR gate is also referred to as a negative AND gate.
- In a NOR gate, the LOW level is the active output level.
- The unique output from a NOR gate is HIGH only when all its inputs are LOW.
- A NOR gate is a purchasable piece of hardware, usually an integrated circuit.

- NOT, OR, or AND gates can be implemented using only NOR gates.
- The 10102 is a Quad 2-input NOR gate.
- The CMOS. 4002 can accommodate only two 4-input NOR gates.
- The XOR gate has a HIGH output only when an odd number of inputs is HIGH.
- The ECL-10107 is a triple exclusive OR-NOR gate.
- The 7486 is a TTL Quad XOR.
- The 4077 is a CMOS Quad XNOR.
- When the two logic levels are opposite, the output of the XNOR gate is LOW.



Test your
understanding

REVIEW QUESTIONS

1. How many table entries are needed for a five-input circuit?
2. When is the output of an AND gate HIGH?
3. When is the output of an AND gate LOW?
4. What is the only input combination that will produce a HIGH at the output of a five-input AND gate?
5. What logic level should be applied to the second input of a two-input AND gate if the logic signal at the first input is to be prevented from reaching the output?
6. Describe the truth table for a three-input AND gate?
7. When is the output of an OR gate HIGH?
8. When is the output of an OR gate LOW?
9. What is the only set of input conditions that will produce a LOW output for any OR gate?
10. Write the Boolean expressions for a six-input OR gate?
11. Describe the truth table for a two-input OR gate.
12. If a 1 is on the input of an inverter what is the output?
13. Write a logic equation for each of the following:
 - (a) $A = \text{NOT } B$,
 - (b) $A \text{ OR } B = F$,
 - (c) $A \text{ AND } B \text{ AND NOT } C = F$
14. What is the only set of input conditions that will produce a HIGH output from a three-input NOR gate.
15. Change the NOR gate of Fig. 4.87 to a NAND gate and change the NAND gate to a NOR. What is the new expression for X ?
16. When is the output of a NAND gate HIGH?
17. When is the output of a NAND gate LOW?
18. What is the functional difference between a NAND gate and a negative-OR gate? Do they both have the same truth table?
19. Write the output expression for NAND gate with inputs A , B , and C .
20. When is the output of a NOR gate HIGH?
21. When is the output of a NOR gate LOW?
22. Describe the functional difference between a NOR gate and a negative-AND gate? Do they both have the same truth table?
23. Write the output expression for a three input NOR with input variable A , B , and C .
24. When is the output of an XOR gate HIGH?
25. When is the output of XNOR gate HIGH?
26. How can you use an XOR gate to detect when two bits are different?
27. Describe the significance of a timing diagram.

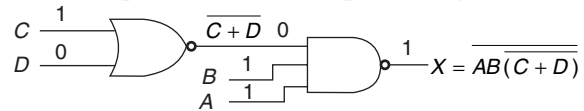


Fig. 4.87

SUPPLEMENTARY PROBLEMS

Test your
understanding

28. Pins 1 and 2 are the input and output pins of an inverter. With the waveform shown in Fig. 4.88 as input to pin 1, sketch the output waveform at pin 2.



Fig. 4.88

29. The input waveform shown below is applied to an inverter. Sketch the output waveform in proper relationship to the input.



Fig. 4.89

30. Sketch the output waveform at X for the three-input AND gate shown below.

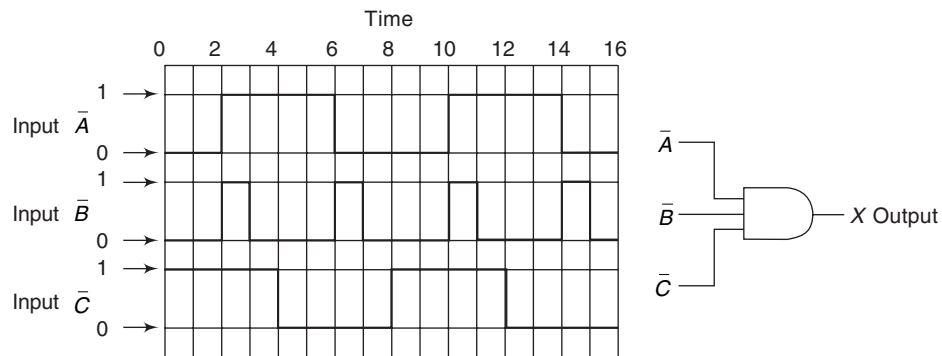


Fig. 4.90

31. Draw the truth table for the given diagram.

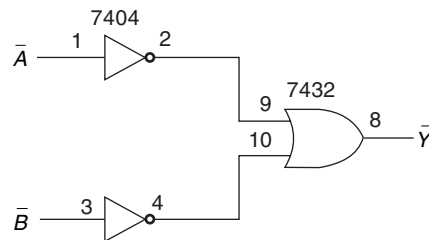


Fig. 4.91

32. Draw the truth table for the given diagram. (See Fig. 4.92)
 33. Draw the truth table and its binary equivalent for a three-input AND gate.
 34. Draw the timing diagram of a three-input NAND gate to show that the output is LOW only when all three inputs are HIGH.
 35. Sketch the output waveform at X for the NAND gate shown with given input waveforms. (See Fig. 4.93)
 36. Sketch the output waveform at X for the NOR gate shown with the given input waveforms. (See Fig. 4.94)

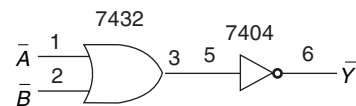


Fig. 4.92

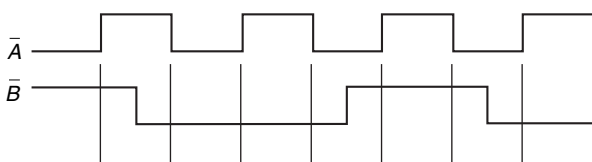


Fig. 4.93

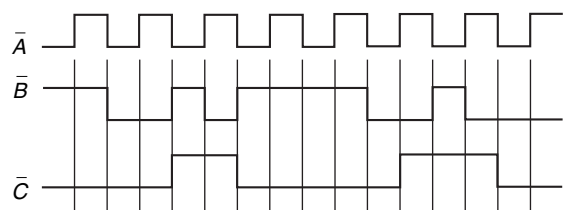
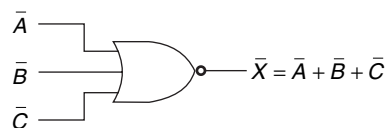


Fig. 4.94

37. Using the given diagram, sketch the waveform that will allow only the even pulses (2, 4, 6, 8 etc.) to get through. (See Fig. 4.95)
38. Sketch the output waveform at X for two-input NOR gate with the given input waveforms. (See Fig. 4.96)

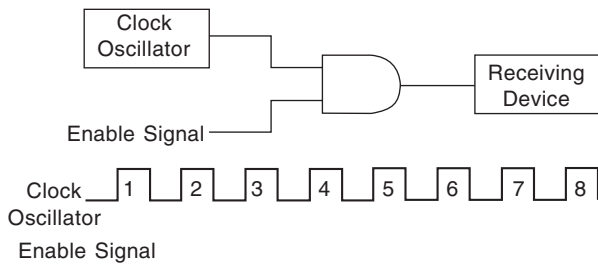


Fig. 4.95

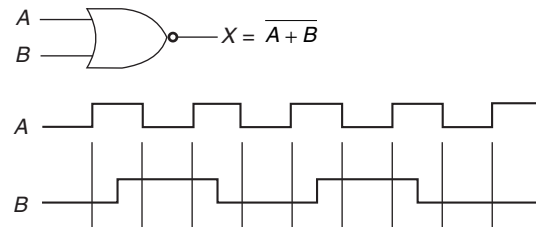


Fig. 4.96

39. Sketch the output waveform at X for the NAND gate shown with the given input waveforms at A , B , and control. (See Fig. 4.97)
40. Draw the truth table for a three-input NAND gate.

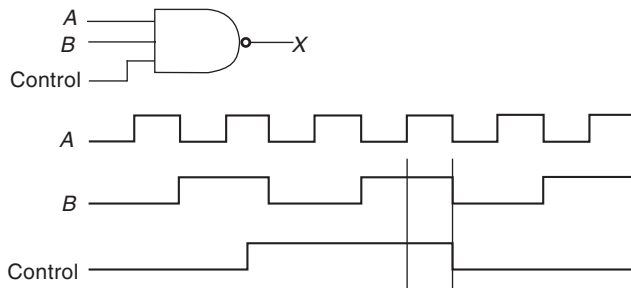


Fig. 4.97

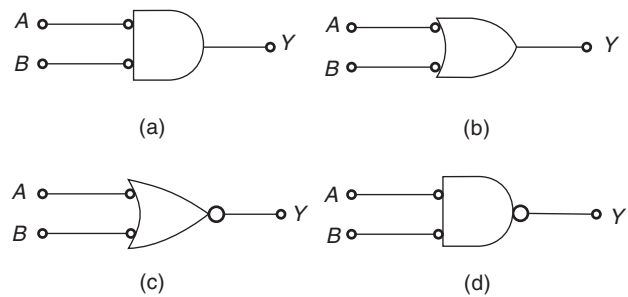


Fig. 4.98

41. Find the relationship between the inputs and the outputs for each of the gates shown in Fig. 4.98. Name the operation performed in each case.
42. The input waveform at A is given for the two-input OR gate in Fig. 4.99. Sketch the input waveform at B that will produce the output at X .

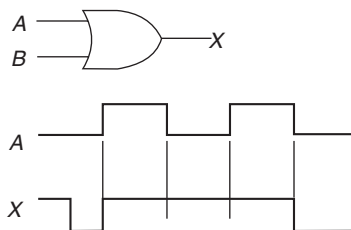


Fig. 4.99

OBJECTIVE TYPE QUESTIONS

Test your understanding

Fill in the Blanks

43. NOT A is written as _____.
44. The output of an AND gate, if one of its input terminals is connected to logic 0, is _____.
45. The output of an AND gate when inhibited is logic 0 _____ of all other inputs.
46. The logic signal required to _____ an OR gate is logic 1.

47. The output of a two-input NAND gate is _____ if $A = 1$, $B = 0$.
48. The output side of a NAND gate logic symbol is _____ (flat with an added invert bubble, pointed with an added invert bubble, round with an added invert bubble.)
49. The output of a two-input NOR gate is _____ if $A = 0$, $B = 1$.
50. The XOR gate may be called a (η) _____ gate.
51. The XNOR circuit operates completely opposite to the _____ circuit.
52. The XNOR circuit produces a HIGH output whenever the two inputs are at the _____ level.
53. The XNOR circuit has only _____ inputs.
54. If input A of a two input XNOR gate is LOW and input B is HIGH, the output of the XNOR gate will be _____.
55. Boolean multiplication is the same as _____.
56. A bar over a variable means _____.
57. Boolean addition is the same as _____.
58. A double bar over a variable means _____.
59. Complementation is the same as _____.
60. Boolean expressions can be _____ using algebraic methods.
61. If the higher of the two voltages represents a 0 and the lower voltage represents a 1, the system is called a _____ system.
62. The list of all possible input combinations follows _____ counting sequence.
63. The number of input entries will equal _____ for n -input truth table.
64. A logic _____ is a graphic way of indicating a particular logic function.
65. Venn diagrams may be used to provide a graphic illustration of _____ operations.
66. The intersection of two classes is identical to the _____ function.
67. The union of two classes is identical to the _____ function.
68. A _____ state is indicated by the digit 1.
69. The basic purpose of an AND gate is to determine when certain conditions are _____ true.
70. The _____ level is the active output level for the AND gate.
71. Any voltage above _____ is considered true.
72. Any voltage below _____ is considered false.
73. AND gates may have _____ of inputs.
74. An AND gate _____ the truth table operation regardless of whether its inputs are constant levels or pulsed levels.
75. An AND gate requires that _____ inputs be HIGH in order for the output to be HIGH.
76. AND and OR are called _____ connectives.
77. $A + 1 =$ _____ and $A \cdot 1 =$ _____.
78. An OR gate can be used to _____ a function.
79. The purpose of the inverter is to change one logic level to the _____ logic level.
80. The negation indicator is a _____ appearing on the input or output of a logic element.
81. Inputs are on the _____ of a logic symbol.
82. The placement of a bubble on the input or output of a logic element is determined by the _____ of the input signal.
83. The active state is the state when the signal is considered to be _____ on the input.
84. $\bar{\bar{1}} =$ _____, $\bar{\bar{1}} =$ _____.
85. An even number of negations is equal to _____.
86. An _____ number of negations is equal to a single negation.
87. The NAND function implies an AND function with a _____ output.
88. In a NAND gate, the output is _____ when any or all the inputs are LOW.
89. A NAND gate is equivalent to an active-LOW input _____ gate.
90. The _____ gate can perform an AND function and an OR function.
91. A NAND gate is a piece of _____.
92. The unique output from a NAND gate is a LOW only when all inputs are _____.
93. An _____ gate provides for the connection of additional diodes to increase the number of input terminals.
94. The NAND gate can be used as an _____ by connecting the input leads together.
95. A NOR gate is equivalent to a _____ AND gate.

96. A NOR gate is a universal _____ building block.
97. The NOR gate can be used as an _____ by connecting the input leads together.
98. The XOR gate produces a HIGH output only when an _____ number of HIGH inputs are present.
99. The XNOR gate is also called an _____ gate.

True/False Questions

State whether the following statements are True or False.

100. A logic gate has an input terminal and one or more output terminals.
101. Each logic gate is represented by a particular logic symbol.
102. Boolean algebra variables can have only the values 0 or 1.
103. An open switch is equivalent to 1 and a closed switch equivalent to 0.
104. The + (plus) sign in Boolean algebra refers to the AND function.
105. A double bar has no effect on the logical value.
106. The list of all possible input combinations in a truth table follows the octal counting sequence.
107. The basic function of the AND gate is to determine when certain conditions are simultaneously true.
108. An inclusive OR function includes the possibility of *A* being true, OR *B* being true, OR both being true.
109. An AND gate can be used to enable/disable a clock oscillator.
110. An OR gate can be used to disable a function.
111. A logic gate that negates the input is called a buffer.
112. A bubble on the input means a double inversion.
113. An AND gate with inverters on its inputs and output is called a NAND gate.
114. Logic gates can have only two inputs.
115. XOR and XNOR gates have only two inputs.
116. It is possible to expand the XOR gate.
117. The XNOR gate may be called a non-equivalence gate.

Multiple Choice Questions

118. The voltage levels for positive logic system
(a) must necessarily be positive. (b) must necessarily be positive or negative.
(c) may be positive or negative. (d) must necessarily be 0 or 5 V.
119. The digital operations such as AND, OR, NOT etc. can be performed by using
(a) switches (b) amplifiers
(c) rectifiers (d) oscillators
120. An inverter gate can be developed using
(a) two diodes (b) a transistor
(c) a resistance and a capacitance (d) an inductance and a capacitance
121. If an input *A* is given to an inverter, the output will be
(a) *A* (b) \bar{A}
(c) $1/A$ (d) 1
122. The output of a two-input OR gate is HIGH
(a) only if both the inputs are HIGH
(b) only if both the inputs are LOW
(c) only if one input is HIGH and the other is LOW
(d) only if at least one of the inputs is LOW.
123. For an AND gate
(a) All LOW inputs produce a HIGH output
(b) All HIGH inputs produce a LOW output
(c) All HIGH inputs produce a HIGH output
(d) All LOW inputs produce a LOW output
124. The output of a gate is LOW when at least one of its inputs is HIGH. This is true for
(a) AND (b) NAND
(c) OR (d) NOR
125. The output of a gate is LOW when at least one of its inputs is LOW. This is true of
(a) AND (b) NAND
(c) OR (d) NOR

126. The output of a two-input AND gate is HIGH
(a) only if both inputs are HIGH
(b) only if both inputs are LOW
(c) only if one input is HIGH and the other is LOW
(d) if at least one of the inputs is LOW
127. NAND gate means
(a) inversion followed by AND gate (b) AND gate followed by an inverter
(c) AND gate followed by an OR gate (d) OR gate followed by an AND gate
128. The output of a gate is HIGH when at least one of its inputs is LOW. It is true for
(a) XOR (b) NAND
(c) NOR (d) OR
129. The output of a gate is HIGH when at least one of its inputs is HIGH. It is true for
(a) NAND (b) AND
(c) OR (d) XOR
130. The output of a gate is HIGH when all of its inputs are HIGH. It is true for
(a) XOR (b) AND
(c) OR (d) NAND
131. The output of a gate is LOW if and only if all of its inputs are HIGH. It is true for
(a) AND (b) XNOR
(c) NOR (d) NAND
132. The output of a gate is HIGH if and only if all of its inputs are LOW. It is true for
(a) NOR (b) XNOR
(c) NAND (d) XOR
133. The output of a two-input NAND gate is HIGH
(a) only if both the inputs are HIGH
(b) only if both the inputs are LOW
(c) only if one input is HIGH and the other is LOW
(d) if at least one of the inputs is LOW.
134. A NOR gate means
(a) inversion followed by an OR gate (b) OR gate followed by an inverter
(c) NOT gate followed by an OR gate (d) NAND gate followed by an OR gate.
135. An XOR gate gives a HIGH output
(a) if there are odd number of 1's in the input
(b) if there are even number of 1's in the input
(c) if there are odd number of 0's in the input
(d) if there are even number of 0's in the input.
136. An XNOR gate is logically equal to
(a) inverter followed by XOR gate (b) NOT gate followed by an XOR gate
(c) XOR gate followed by an inverter (d) complement of a NOR gate.
137. The logic expression $AB + \bar{A}\bar{B}$ can be implemented by giving inputs A and B to a two-input
(a) NOR gate (b) XNOR gate
(c) XOR gate (d) NAND gate
138. The logic expression $A\bar{B} + \bar{A}B$ can be implemented by giving inputs A and B to a two input
(a) NAND gate (b) XOR gate
(c) XNOR gate (d) NOR gate
139. The gate ideally suited for bit comparison is a
(a) two-input XNOR gate (b) two-input XOR gate
(c) two-input NOR gate (d) two-input NAND gate
140. Two-input XNOR gate gives HIGH output
(a) when one input is HIGH and the other is LOW
(b) only when both the inputs are LOW
(c) when both the inputs are the same
(d) only when both the inputs are HIGH.

141. The output of a gate is LOW if and only if all its inputs are LOW. It is true for
(a) XOR (b) AND
(c) OR (d) NOR
142. The output of a two-input gate is 1 if and only if its inputs are unequal. It is true for
(a) OR (b) XOR
(c) NOR (d) XNOR
143. The output of a two-input gate is 0 if and only if its inputs unequal. It is true for
(a) XNOR (b) NOR
(c) AND (d) NAND
144. The output of a two-input gate is 1 if and only if its inputs are equal. It is true for
(a) XOR (b) XNOR
(c) AND (d) NAND
145. The output of a two-input gate is 0 if and only if its inputs are equal. It is true for
(a) AND (b) OR
(c) NOR (d) XOR
146. Which of the following gates can be used as an inverter?
(a) AND (b) OR
(c) NAND (d) NOR
147. A gate is enabled when its enable input is at logic 1. The gate is
(a) AND (b) OR
(c) NAND (d) NOR
148. A gate is enabled when its enable input is at logic 0. The gate is
(a) OR (b) AND
(c) NOR (d) NAND
149. A gate is inhibited when its inhibit input is at logic 1. The gate is
(a) AND (b) NAND
(c) OR (d) NOR
150. A gate is disabled when its disable input is at logic 0. The gate is
(a) NAND (b) NOR
(c) OR (d) AND
151. The output of a logic gate is 1 when all its inputs are at logic 1. The gate is either
(a) a NAND or a NOR (b) an AND or an OR
(c) an OR or an XOR (d) an AND or a NOR
152. The output of a logic gate is 1 when all its inputs are at logic 0. The gate is either
(a) an XOR or an XNOR (b) an OR or a NAND
(c) an AND or an XNOR (d) a NAND or a NOR
153. Which of the following gates is known as an universal gate?
(a) AND (b) NAND
(c) OR (d) NOT
154. Any logical expression can be realized by using only
(a) AND gates (b) AND and NOT gates
(c) OR and NOT gates (d) AND, OR and NOT gates
155. The NAND gate is known as a universal gate because
(a) it can be used as an inverter
(b) AND operation can be realized using NAND gates
(c) OR operation can be realized using NAND gates
(d) AND, OR and NOT operations can be performed using NAND gates.
156. Which of the following gates is known as an universal gate?
(a) OR (b) NOR
(c) AND (d) NOT

ANSWERS

1. 32

2. An AND gate output is HIGH when all inputs are HIGH.

3. An AND gate output is LOW when one or more inputs are LOW.

4. All five inputs = 1.

5. A LOW input will keep the output LOW.

6.

A	B	C	X
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

7. An OR gate output is HIGH when one or more inputs are HIGH.

8. An OR gate output is LOW when all inputs are LOW.

9. All inputs LOW.

10. $X = A + B + C + D + E + F$.

11.

A	B	X
0	0	0
0	1	1
1	0	1
1	1	1

12. When an inverter input is 1, the output is 0.

13. $A = \bar{B}$ (b) $A + B = F$ (c) $AB\bar{C} = F$.

14. All inputs LOW.

15. $X = \overline{A + B + CD}$

16. A NAND gate output is HIGH when one or more inputs are LOW.

17. A NAND gate output is LOW when all inputs are HIGH.

18. NAND: Active-LOW output for all HIGH inputs; Negative-OR: Active-HIGH output for one or more LOW inputs; Same truth table.

19. $X = \overline{ABC}$.

20. A NOR gate output is HIGH when all inputs are LOW.

21. A NOR gate output is LOW when one or more inputs are HIGH.

22. NOR: Active-LOW output for one or more HIGH inputs, Negative-And: Active-HIGH output for all LOW inputs. Same truth table.

23. $X = \overline{A + B + C}$.

24. An XOR gate output is HIGH when the inputs are at opposite levels.

25. An XNOR gate output is HIGH when all inputs are at the same level.

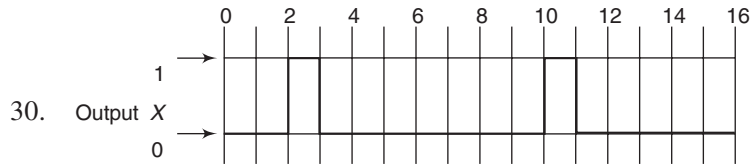
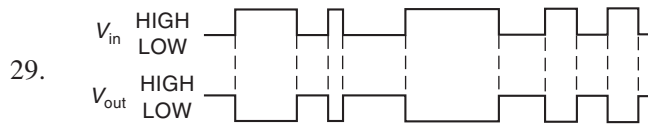
26. Apply the bits to the XOR gate inputs; when the output is HIGH, the bits are different.

27. A timing diagram is basically a graph that accurately displays the relationship of two or more waveforms with respect to each other on a time basis.

28. Pin 2 Output



A B C D E F G H I



31.

A	B	Y
L	L	H
L	H	H
H	L	H
H	H	L

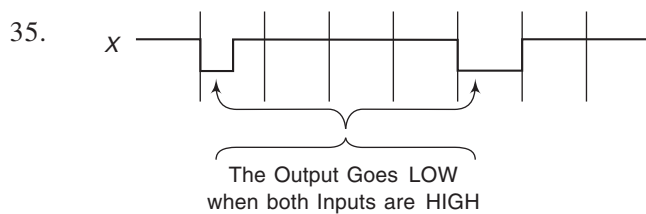
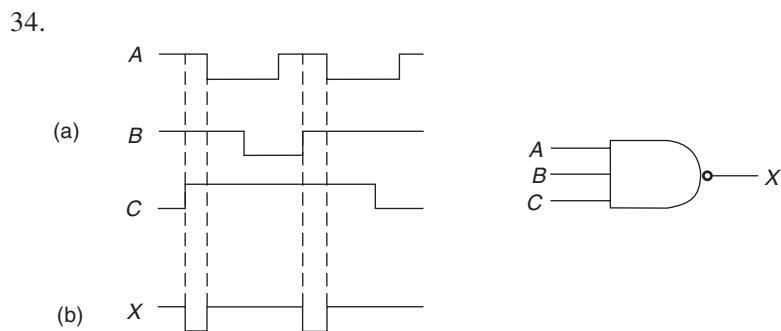
32.

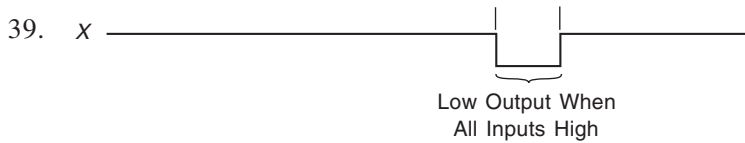
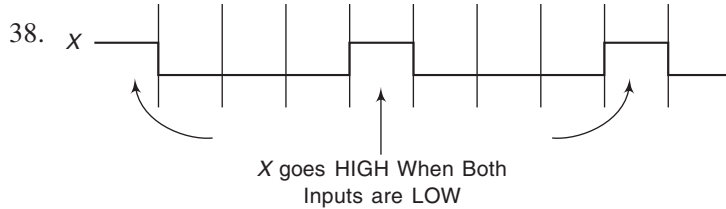
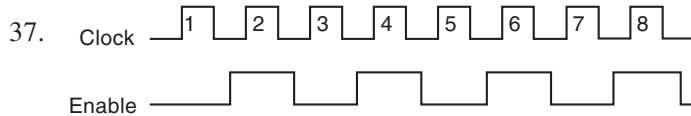
A	B	Y
L	L	H
L	H	L
H	L	L
H	H	L

33.

A	B	C	Y
L	L	L	L
L	L	H	L
L	H	L	L
L	H	H	L
H	L	L	L
H	L	H	L
H	H	L	L
H	H	H	H

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





40.

A	B	C	X
0	0	0	1
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	0

41.

Inputs			Output
A	B		Y
0	0		1
0	1		0
1	0		0
1	1		0

(a) Operation performed—NOR

Inputs			Output
A	B		Y
0	0		1
0	1		1
1	0		1
1	1		0

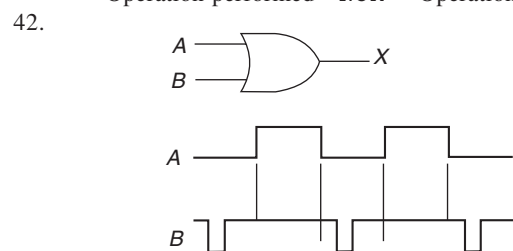
(b) Operation performed—NAND

Inputs			Output
A	B		Y
0	0		0
0	1		0
1	0		0
1	1		1

(c) Operation performed—AND

Inputs			Output
A	B		Y
0	0		0
0	1		1
1	0		1
1	1		1

(d) Operation performed—OR



- | | | | |
|---------------|---------------------------------------|--------------------|---------------------|
| 43. A | 44. logic 0 | 45. irrespective | 46. inhibit |
| 47. HIGH | 48. round with an added invert bubble | 49. LOW | 50. any but not all |
| 51. XOR | 52. same | 53. two | 54. LOW |
| 55. ANDing | 56. complementation | 57. ORing | 58. no inversion |
| 59. inversion | 60. simplified | 61. negative logic | 62. binary |
| 63. 2" | 64. symbol | 65. Boolean | 66. AND |
| 67. OR | 68. true | 69. simultaneously | 70. HIGH |
| 71. 3 V | 72. 1 V | 73. any number | 74. obeys |
| 75. all | 76. logical | 77. 1, A | 78. disable |
| 79. opposite | 80. bubble | 81. left | |

- | | | | |
|------------------|------------------|--------------|------------------|
| 82. active state | 83. present | 84. 0, 1 | 85. no inversion |
| 86. odd | 87. complemented | 88. HIGH | 89. OR |
| 90. NAND | 91. hardware | 92. HIGH | 93. expandable |
| 94. inverter | 95. negative | 96. hardware | 97. inverter |
| 98. odd | 99. equivalence | 100. False | 101. True |
| 102. True | 103. False | 104. False | 105. True |
| 106. False | 107. True | 108. True | 109. True |
| 110. True | 111. False | 112. False | 113. False |
| 114. False | 115. True | 116. True | 117. False |
| 118. (c) | 119. (a) | 120. (b) | 121. (b) |
| 122. (d) | 123. (c) | 124. (c) | 125. (a) |
| 126. (a) | 127. (b) | 128. (b) | 129. (c) |
| 130. (b) | 131. (d) | 132. (a) | 133. (d) |
| 134. (b) | 135. (a) | 136. (c) | 137. (b) |
| 138. (b) | 139. (a) | 140. (c) | 141. (c) |
| 142. (b) | 143. (a) | 144. (b) | 145. (d) |
| 146. (c) and (d) | 147. (a) | 148. (c) | 149. (c) |
| 150. (d) | 151. (b) | 152. (d) | 153. (b) |
| 154. (d) | 155. (d) | 156. (b) | |

Boolean Algebra

INTRODUCTION

Named after its inventor, George Boole (1854), Boolean algebra defines constants, variables and functions to describe binary systems. It then describes a number of theorems that can be used to manipulate logic expressions. Boolean *operators* are the codes for the basic logic gates. You can use them as a short hand notation for digital circuits.

Boolean *constants* consist of 0 and 1. Boolean *variables* are quantities that can take different values at different times. They may represent *input, output or intermediate signals* and are given names consisting of alphabetic characters such as A , B , C , X or Y . Boolean variables may only take the values 0 or 1.

Each of the elementary *logic functions* is represented in Boolean algebra by a unique *symbol* as shown below:

Function	Symbol	Example
AND	dot (\cdot)	$C = A \cdot B$
OR	plus ($+$)	$C = A + B$
NOT	overbar ($\bar{}$)	$C = \bar{A}$

Fig. 5.1 Logic Functions.

BOOLEAN THEOREMS

Boolean expressions are not unique. We therefore require some method of *manipulating* expressions into their simplest forms. The rules of Boolean algebra consist of a set of *identities* and a set of *laws*. These are summarised in Table 5.1. It is easier to see the sense of the rules if they are converted into concrete examples when their meaning becomes clear. The identities on AND, OR and NOT functions have already been dealt with in details in Chapter 4. The various laws can also be understood by linking them to concrete examples.

Table 5.1 Summary of Boolean Algebra Identities and Laws

AND function	Commutative law
(1) $0 \cdot 0 = 0$	(24) $AB = BA$
(2) $0 \cdot 1 = 0$	(25) $A + B = B + A$
(3) $1 \cdot 0 = 0$	Distributive law
(4) $1 \cdot 1 = 1$	(26) $A(B + C) = AB + AC$
(5) $A \cdot 0 = 0$	(27) $A + BC = (A + B)(A + C)$
(6) $0 \cdot A = 0$	Associative law
(7) $A \cdot 1 = A$	(28) $A(BC) = (AB)C$
(8) $1 \cdot A = A$	(29) $A + (B + C) = (A + B) + C$
(9) $A \cdot A = A$	Absorption law
(10) $A \cdot \bar{A} = 0$	(30) $A + AB = A$
OR function	(31) $A(A + B) = A$
(11) $0 + 0 = 0$	DeMorgan's law
(12) $0 + 1 = 1$	(32) $\overline{A + B} = \bar{A} \cdot \bar{B}$
(13) $1 + 0 = 1$	(33) $\overline{A \cdot B} = \bar{A} + \bar{B}$
(14) $1 + 1 = 1$	Also note
(15) $A + 0 = A$	(34) $A + \bar{A}B = A + B$
(16) $0 + A = A$	(35) $A(\bar{A} + B) = AB$
(17) $A + 1 = 1$	
(18) $1 + A = 1$	
(19) $A + A = A$	
(20) $A + \bar{A} = 1$	
NOT function	
(21) $\bar{0} = 1$	
(22) $\bar{1} = 0$	
(23) $\overline{\bar{A}} = A$	

COMMUTATIVE LAW

It states that the elements of a function can be arranged in *any* sequence provided the connective is the *same*. Commutative law can also be stated as: 'the *order* in which terms are ANDed or ORed together is unimportant.

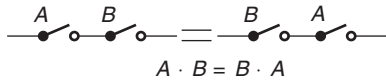


Fig. 5.2 Verifying Commutative Law by Switching Analogy.

5.1 Verify that the following operations are commutative:

- (a) AND (b) OR (c) XOR

Solution:

- (a) $A \cdot B = B \cdot A$ Therefore, the AND operation is commutative.
 (b) $A + B = B + A$ Therefore, the OR operation is commutative.
 (c) $A \oplus B = B \oplus A$ Therefore, the XOR operation is commutative.

DISTRIBUTIVE LAW

The distributive laws allow the *factoring or multiplying* of expressions. Two distributive laws will be considered.

- (a) $A(B + C) = AB + AC$ and
 (b) $A + BC = (A + B)(A + C)$

The first statement of the distributive law can be verified by switching analogy and truth table shown in Fig. 5.3. Both of them are self explanatory.

A	B	C	AB	AC	AB + AC	B + C	A(B + C)
0	0	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	1	0	0	0	0	1	0
1	1	0	1	0	1	1	1
0	0	1	0	0	0	1	0
1	0	1	0	1	1	1	1
0	1	1	0	0	0	1	0
1	1	1	1	1	1	1	1

$AB + AC = A(B + C)$

This Truth Table Illustrates the First Statement of the Distributive Law: Compare the 0's and 1's in Columns 6 and 8!

Fig. 5.3 Verifying the First Statement of the Distributive Law.

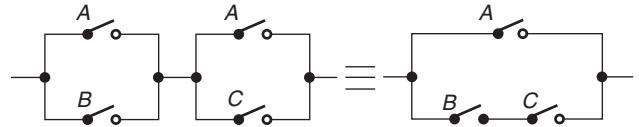
5.2 Verify the second statement of the distributive law:

$$A + BC = (A + B)(A + C)$$

Solution:

The second statement of the distributive law can be verified by switching analogy and truth table shown in Fig. 5.4. Both of them are self explanatory.

Note: This law is extremely useful in the *simplification* of functions and hence of logic circuits. For example in Figs 5.3 and 5.4 one switch contact can be saved by choosing the right-hand circuit in each case.



A	B	C	A + B	A + C	(A + B)(A + C)	BC	A + BC
0	0	0	0	0	0	0	0
1	0	0	1	1	1	0	1
0	1	0	1	0	0	0	0
1	1	0	1	1	1	0	1
0	0	1	0	1	0	0	0
1	0	1	1	1	1	0	1
0	1	1	1	1	1	1	1
1	1	1	1	1	1	1	1

$$(A + B)(A + C) = A + BC$$

Fig. 5.4 Verifying the Second Statement of the Distributive Law.

ASSOCIATIVE LAW

This law merely states that in any Boolean function containing elements (A, B, C etc) separated by the *same* connective, it does not matter if some of the elements are considered as a *group*.

- (a) $ABC = AB(C) = A(BC) = AC(B)$
 (b) $A + B + C = (A + B) + C = A + (B + C) = (A + C) + B$

If three switches are connected in series to perform an AND function, or in parallel to perform an OR function, it does not matter if two of them are considered as a group, the circuit is still the same. This is illustrated in Fig. 5.5.

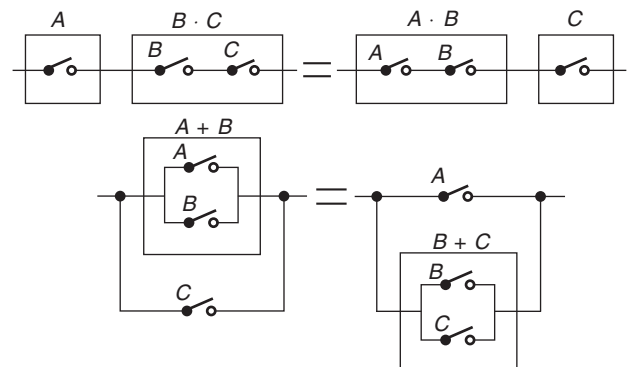


Fig. 5.5 Verifying Associative Law.

Note: The associative law does not hold good if elements are connected by *different* connectives.

5.3 Explain the physical significance of the associative law.

Solution:

When many conditions are to be ANDed or ORed together, the *order* in which the conditions are combined is unimportant. This is illustrated in Fig. 5.6.

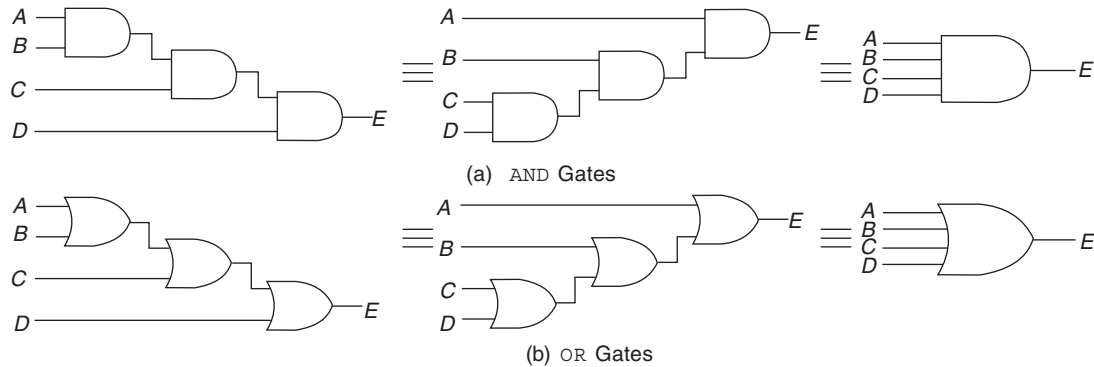


Fig. 5.6 Physical Significance of Associative Law.

5.4 Verify that the following operations are associative:

- (a) AND (b) OR (c) XOR

Solution:

- (a) If $A \cdot (B \cdot C) = (A \cdot B) \cdot C$, then the AND operation is associative. This can be proved by making the truth table as given in Table 5.2.

Table 5.2

A	B	C	$(A \cdot B) \cdot C$	$A \cdot (B \cdot C)$
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	0	0
1	1	0	0	0
1	1	1	1	1

- (b) The associative property requires that

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

This can be proved by making a truth table in a way similar to Table 5.2.

- (c) The associative property requires that

$$A \oplus (B \oplus C) = (A \oplus B) \oplus C$$

This can also be proved by making a truth table in a way similar to Table 5.2.

ABSORPTION LAW

This law is extremely important for the *elimination of redundant functions* in a system.

(a) $A(A + B) = A$ (b) $A + AB = A$

Figures 5.7(a) and (b) illustrate the law of absorption. In both cases the output is logic 0 when A is 0 and logic 1 when A is 1, *irrespective of the state of B*.

IDEMPOTENT LAW

This law states that if *a variable is ANDed or ORed with itself any number of times, the result will always be the original variable*.

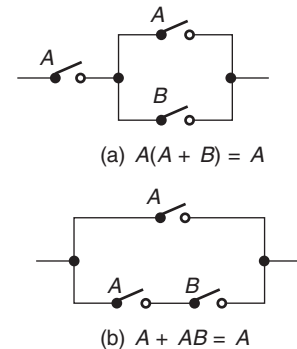


Fig. 5.7 Verifying Law of Absorption.

(a) $A = A \cdot A \cdot A \cdot A \dots$

(b) $A = A + A + A + A \dots$

Idempotent law is illustrated in Fig. 5.8. It must be noted that:

- (1) Boolean algebra gives a different result to normal algebra (in normal algebra $1 + 1 = 2$, whereas in Boolean algebra ($1 \text{ OR } 1 = 1$) $1 + 1 = 1$).
- (2) In Boolean algebra, a variable can have only one of two values, it is either 0 or 1, false or true (*nothing or something*).

A	A	A	...	$A \cdot A \cdot A \dots$	$A + A + A \dots$
0	0	0		0	0
1	1	1		1	1

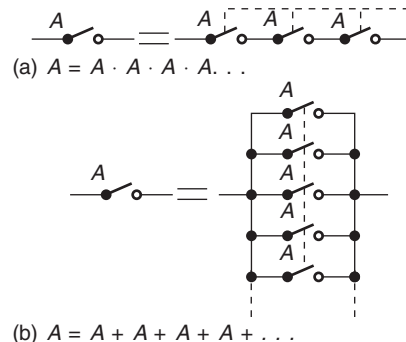


Fig. 5.8 Verifying Idempotent Law.

Obviously, if any number of *nothings* are added or multiplied together, the result will also be nothing, whereas if any number of *somethings* are added or multiplied together, the result will also be something.

5.5 Verify that the following operations are commutative but not associative.

(a) NAND

(b) NOR

Solution:

- (a) Since $AB = BA$, the NAND operation is commutative. In Table 5.3, the last two columns are not identical i.e.,

$$\overline{A \cdot (B \cdot C)} \neq \overline{(A \cdot B) \cdot C}$$

which means that the NAND operation is not associative.

- (b) Since $A + B = B + A$, the NOR operation is commutative.

By drawing a truth table similar to Table 5.3 it can be verified that $\overline{(A + B) + C} \neq \overline{A + (B + C)}$, which means that the NOR operation is not associative.

Table 5.3 The NAND Operation is not Associative.

A	B	C	$\overline{A \cdot (B \cdot C)}$	$\overline{(A \cdot B) \cdot C}$
0	0	0	1	1
0	0	1	1	0
0	1	0	1	1
0	1	1	1	0
1	0	0	0	1
1	0	1	0	0
1	1	0	0	1
1	1	1	1	1

LAW OF IDENTITY

The law of identity states that

$$A = A = A \dots \text{etc.}$$

It does have one *interesting consequence*, which is perhaps not quite so obvious:

$$\text{if } A = B \text{ and } B = C, \text{ then } A = C$$

LAW OF COMPLEMENTATION

This law states that *if a function consists of a variable and its inverse, then the function is a constant.*

$$(a) A\bar{A} = 0$$

$$(b) A + \bar{A} = 1$$

Since an AND gate requires *both* inputs to be logic 1 for a logic 1 output, $A\bar{A}$ is always logic 0, since A and \bar{A} can never be logic 1 simultaneously. This is shown in Fig. 5.9(a).

Since an OR gate requires *only one* input to be logic 1 for a logic 1 output, either A or \bar{A} must be 1 at any time, so the result of $A + \bar{A}$ is always logic 1. This is shown in Fig. 5.9(b).

A	\bar{A}	$A\bar{A}$
0	1	0
1	0	0

(a) $A\bar{A} = 0$

A	\bar{A}	$A + \bar{A}$
0	1	1
1	0	1

(b) $A + \bar{A} = 1$

Fig. 5.9 Verifying Law of Complementation.

The switching analogy to verify the law of complementation is illustrated in Fig. 5.10. *Something times nothing equals nothing* ($A\bar{A} = 0$), whereas *nothing plus something equals something* ($A + \bar{A} = 1$).

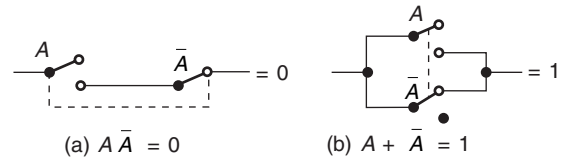


Fig. 5.10 Switching Analogy to Verify Law of Complementation.

CONNECTION WITH A CONSTANT

Four important relationships that can be used in the *simplification* of Boolean functions are:

1. *Conjunction* (AND and NAND functions) of a variable with *logic 0* always yields a constant ($A \cdot 0 = 0$, and $\bar{A} \cdot 0 = 1$).
2. *Conjunction* of a variable with *logic 1* results in the original variable ($A \cdot 1 = A$ and $\bar{A} \cdot 1 = \bar{A}$).
3. *Disjunction* (OR, XOR, and NAND functions) of a variable with *logic 0* results in the original variable ($A + 0 = A$ and $\bar{A} + 0 = \bar{A}$).
4. *Disjunction* of a variable with *logic 1* results in a constant output ($A + 1 = 1$ and $\bar{A} + 1 = 1$).

5.6 Prove that

$$A + \bar{A}B = A + B$$

Solution:

$$\begin{aligned} A + \bar{A}B &= (A + AB) + \bar{A}B && \text{(rule 30)} \\ &= (AA + AB) + \bar{A}B && \text{(rule 9)} \\ &= AA + AB + \bar{A}A + \bar{A}B && \text{(rule 10)} \\ &= (A + \bar{A})(A + B) && \text{(by factoring)} \\ &= 1 \cdot (A + B) && \text{(rule 20)} \\ &= A + B && \text{(rule 34)} \end{aligned}$$

5.7 Prove that

$$(A + B)(A + C) = A + BC$$

Solution:

$$\begin{aligned} (A + B)(A + C) &= AA + AC + AB + BC \\ &= A + AC + AB + BC && \text{(distributive law)} \\ &= A + AC + AB + BC && \text{(rule 9)} \end{aligned}$$

$$\begin{aligned}
 &= A(1 + C) + AB + BC \\
 &\quad \text{(distributive law)} \\
 &= A \cdot 1 + AB + BC \quad \text{(rule 18)} \\
 &= A(1 + B) + BC \quad \text{(distributive law)} \\
 &= A \cdot 1 + BC \quad \text{(rule 18)} \\
 &= A + BC \quad \text{(rule 7)} \\
 (A + B)(A + C) &= A + BC \quad \text{(rule 27)}
 \end{aligned}$$

5.8 Prove that

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

Solution:

$$\begin{aligned}
 (A + C)(A + D)(B + C)(B + D) &= (A + CD)(B + CD) \\
 &= AB + ACD + BCD + CD \cdot CD \quad \text{(rule 27)} \\
 &= AB + ACD + BCD + CD \quad \text{(rule 9)} \\
 &= AB + ACD + CD(B + 1) \quad \text{(factoring)} \\
 &= AB + ACD + CD \quad \text{(rule 18)} \\
 &= AB + CD(A + 1) \quad \text{(factoring)} \\
 &= AB + CD \quad \text{(rule 18)} \\
 AB + CD &= (A + C)(A + D)(B + C)(B + D)
 \end{aligned}$$

5.9 Prove that

$$A(\bar{A} + C)(\bar{A}B + \bar{C}) = 0$$

Solution:

$$\begin{aligned}
 A(\bar{A} + C)(\bar{A}B + \bar{C}) &= (A\bar{A} + AC)(\bar{A}B + \bar{C}) \\
 &= (0 + AC)(\bar{A}B + \bar{C}) \quad \text{(rule 10)} \\
 &= AC\bar{A}B + AC\bar{C} \\
 &= A\bar{A}BC + AC\bar{C} \\
 &= 0 \cdot BC + A \cdot 0 \quad \text{(rule 10)} \\
 &= 0
 \end{aligned}$$

Therefore, $A(\bar{A} + C)(\bar{A}B + \bar{C}) = 0$ **5.10 Simplify the Boolean expression**

$$AB + A(B + C) + B(B + C)$$

Solution:

$$\begin{aligned}
 AB + A(B + C) + B(B + C) &= AB + AB + AC + BB + BC \\
 &= AB + AB + AC + B + BC \quad (BB = B, \text{ rule 9}) \\
 &= AB + AC + B + BC \quad (AB + AB = AB, \text{ rule 19}) \\
 &= AB + AC + B(1 + C) \\
 &= AB + AC + B \quad (1 + C = 1, \text{ rule 18}) \\
 &= AB + B + AC \\
 &= B(A + 1) + AC \\
 &= B + AC \quad (A + 1 = 1, \text{ rule 17})
 \end{aligned}$$

Therefore, $AB + A(B + C) + B(B + C) = B + AC$ **5.11 Show that $(A + B)(\bar{A} + C) = AC + \bar{A}B$** *Solution:*

$$\begin{aligned}
 (A + B)(\bar{A} + C) &= \bar{A}A + AC + \bar{A}B + BC \\
 &= 0 + AC + \bar{A}B + BC(A + \bar{A}) \\
 &= AC + \bar{A}B + ABC + \bar{A}BC
 \end{aligned}$$

$$\begin{aligned}
 &= AC + ABC + \bar{A}B + \bar{A}BC \\
 &= AC(1 + B) + \bar{A}B(1 + C) \\
 &= AC \cdot 1 + \bar{A}B \cdot 1 \\
 &= AC + \bar{A}B = \text{R.H.S.}
 \end{aligned}$$

5.12 Simplify the expression

$$ABC + \bar{A}\bar{B}\bar{C} + ABC\bar{C} + \bar{A}BC$$

Solution:

$$\begin{aligned}
 &ABC + \bar{A}\bar{B}\bar{C} + ABC\bar{C} + \bar{A}BC \\
 \text{Rearrange} &ABC + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + ABC \\
 \text{Distributive law} &BC(A + \bar{A}) + \bar{A}\bar{B}\bar{C} + ABC\bar{C} \quad \text{(rule 26)} \\
 \text{OR law} &BC + \bar{A}\bar{B}\bar{C} + ABC\bar{C} \quad \text{(rule 20)} \\
 \text{Rearrange} &BC + ABC\bar{C} + \bar{A}\bar{B}\bar{C} \\
 \text{Distributive law} &B(C + AC\bar{C}) + \bar{A}\bar{B}\bar{C} \quad \text{(rule 26)} \\
 \text{Distributive law} &B(A + C)(C + \bar{C}) + \bar{A}\bar{B}\bar{C} \quad \text{(rule 27)} \\
 &B(A + C) + \bar{A}\bar{B}\bar{C}
 \end{aligned}$$

5.13 Simplify the Boolean equation $\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} +$

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

Solution:

$$\begin{aligned}
 &\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} + BC + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} \\
 \text{Distributive law} &\bar{B}\bar{C}(\bar{A} + A) + BC + \bar{B}\bar{C}(\bar{A} + A) \quad \text{(rule 26)} \\
 \text{OR law} &\bar{B}\bar{C} + BC + \bar{B}\bar{C} \quad \text{(rule 20)} \\
 \text{Distributive law} &\bar{B}(\bar{C} + C) + BC \quad \text{(rule 26)} \\
 \text{OR law} &\bar{B} + BC \quad \text{(rule 20)}
 \end{aligned}$$

5.14 Simplify $\bar{A}BC + \bar{B}CD + AC + \bar{A}\bar{B}\bar{C}\bar{D}$ *Solution:*

$$\begin{aligned}
 &\bar{A}BC + \bar{B}CD + AC + \bar{A}\bar{B}\bar{C}\bar{D} \\
 \text{OR law} &\bar{A}BC + \bar{B}CD(A + \bar{A}) + AC(B + \bar{B}) \\
 &\quad + \bar{A}\bar{B}\bar{C}\bar{D} \quad \text{(rule 26)} \\
 &\bar{A}BC + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + ABC + \bar{A}\bar{B}\bar{C} \\
 &\quad + \bar{A}\bar{B}\bar{C}\bar{D} \\
 \text{Rearrange} &\bar{A}BC + ABC + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C} \\
 &\quad + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} \\
 \text{Distributive law} &BC(\bar{A} + A) + \bar{A}\bar{B}\bar{C}(D + 1) \\
 &\quad + \bar{A}\bar{B}\bar{C}(D + \bar{D}) \quad \text{(rule 26)} \\
 \text{OR law} &BC + \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}\bar{C} \quad \text{(rule 20)} \\
 \text{Distributive law} &BC + \bar{B}\bar{C}(A + \bar{A}) \quad \text{(rule 26)} \\
 \text{OR law} &BC + \bar{B}\bar{C} \quad \text{(rule 20)} \\
 \text{Distributive law} &C(B + \bar{B}) \quad \text{(rule 26)} \\
 \text{OR law} &C \quad \text{(rule 20)}
 \end{aligned}$$

LAW OF DUALISATION (DEMORGAN'S THEOREM)

This is one of the most important laws of Boolean algebra, since it formulates the relationship between N

(AND) and N(OR) functions that allows one type of function to be implemented using a different type of gate.

$$(a) \overline{A+B} = \overline{A} \cdot \overline{B} \quad (b) \overline{A \cdot B} = \overline{A} + \overline{B}$$

The practical value of these equations is immediately apparent. Equation (a) means that a NOR function can be implemented by inverting the two inputs to an AND function. Equation (b) means that a NAND function can be implemented by inverting the two inputs to an OR function.

Inverting these equations gives two further important relationships:

$$(c) A+B = \overline{\overline{A} \cdot \overline{B}} \quad (d) A \cdot B = \overline{\overline{A} + \overline{B}}$$

Equation (c) states that an OR function can be implemented by inverting the inputs to a NAND function. Equation (d) means that an AND function can be implemented by inverting the two inputs to a NOR function.

The first law states that the complement of a sum is equal to the product of the complements. This is stating that the complement of two or more variables ORed is the same as the AND of the complements of each individual variable.

The second law states that the complement of a product is equal to the sum of the complements. This is stating that the complement of two or more variables ANDed is the same as the OR of the complements of each individual variable.

A useful general rule for remembering DeMorgan's theorem is as follows:

When the overline extends across a function (e.g. $\overline{A+B}$) then it *inverts the connective*. The inverse of OR is AND and vice versa. The function may then be rewritten using the *inverse connective* and *individual overlines* on the variables. For example:

$$\left. \begin{array}{l} \overline{A \cdot B} = \overline{A} \cdot \overline{B} \\ \text{'}\cdot\text{' = '+', } \overline{A \cdot B} = \overline{A} + \overline{B} \\ \overline{A+B} = \overline{A} \cdot \overline{B} \\ \text{'}\cdot\text{' = '+', so } \overline{A+B} = \overline{A} \cdot \overline{B} \end{array} \right\}$$

The converse is also quite true.

Individual overlines on the variables may be replaced by a continuous overline and the inverse connective.

All possible applications of DeMorgan's theorem to two variables are given in Table 5.4.

Table 5.4

OR	\leftrightarrow	NAND	AND	\leftrightarrow	NOR
$\overline{A} + \overline{B}$		$\overline{A \cdot B}$	$\overline{A} \cdot \overline{B}$		$\overline{A+B}$
$A + \overline{B}$		$\overline{A \cdot \overline{B}}$	$A \cdot \overline{B}$		$\overline{A+B}$
$\overline{A} + B$		$\overline{\overline{A} \cdot B}$	$\overline{A} \cdot B$		$\overline{A+B}$
$A + B$		$\overline{\overline{A} \cdot \overline{B}}$	$A \cdot B$		$\overline{A+B}$

The truth of DeMorgan's theorem can be demonstrated in several ways.

Using switch contacts, as in Fig. 5.11, the only handicap is that the *inversion of a total function must be illustrated by adding an inverter*.

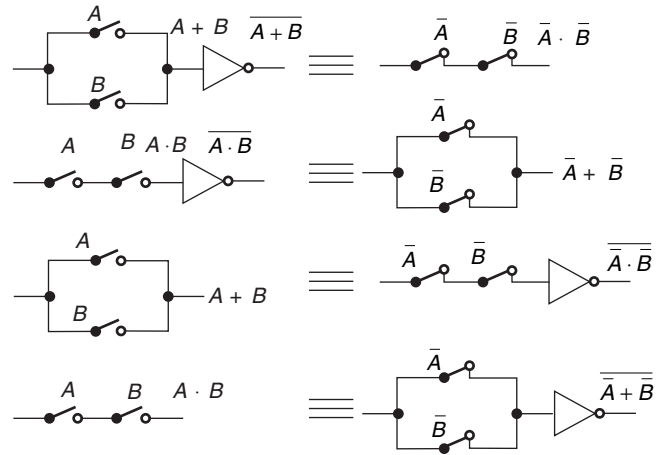


Fig. 5.11 Demonstrating the Truth of DeMorgan's Theorem.

Even so, the results are obvious. For instance, in the first circuit if either switch A or switch B is closed the input to the inverter is 1 and therefore its output is 0. This is identical to the second circuit, where closing either of the switches breaks the circuit, so that the output becomes 0. These two circuits therefore illustrate the first statement: $\overline{A+B} = \overline{A} \cdot \overline{B}$.

5.15 Reduce the expression $\overline{A+B(C+DE)}$.

Solution:

$$A+B(C+DE)$$

Break the line, change the sign.

First break the line between D and E

$$A+B(\overline{C+D+E})$$

Next break the line between C and D and E

$$A+B(\overline{C} \overline{D} \overline{E}) = A+B\overline{C} \overline{D} \overline{E}$$

5.16 Reduce the expression $\overline{AB+A+AB}$.

Solution:

$$\overline{AB+A+AB}$$

Demorganise \overline{AB} $\overline{A+B+A+AB}$ (rule 33)

Reduce $\overline{A+B+A+AB}$ (rule 19)

Reduce $\overline{A+B+A}$ (rule 34)

Rearrange	$\overline{A + \overline{A} + \overline{B}}$	(rule 25)
Reduce	$\overline{1 + \overline{B}}$	(rule 20)
Reduce	$\overline{1}$	(rule 22)
Convert	0	

5.17 Demorganise the function $\overline{AB + C}$.

Solution:

	$\overline{AB + C}$
Complement function	$\overline{AB} + \overline{C}$
Change operators	$(A + \overline{B})(\overline{C})$
Complement variables	$(\overline{A} + B)(\overline{C})$

5.18 Demorganise the expression $\overline{A + BC + D(E + F)}$.

Solution:

	$\overline{A + BC + D(E + F)}$
Break the bar, change the sign,	$\overline{(A + BC)}[\overline{D(E + F)}]$
Cancel the double bars over the left term.	$(A + \overline{BC})[\overline{D(E + F)}]$
	$[\overline{D(E + F)}]$
Break the bar, change the sign.	$(A + \overline{BC})[\overline{D(E + F)}]$
Cancel the double bars.	$(A + \overline{BC})(\overline{D + E + F})$

5.19 Reduce $\overline{AB + ABC + A(B + AB)}$.

Solution:

	$\overline{AB + ABC + A(B + AB)}$
Factorize	$\overline{A(\overline{B} + BC) + A(B + AB)}$
Reduce	$\overline{A(\overline{B} + C) + A(B + A)}$
Multiply	$\overline{AB + AC + AB + AA}$
Reduce	$\overline{AB + AC + AB + A}$
Factorize	$\overline{AB + AC + A(B + 1)}$
Reduce	$\overline{AB + AC + A}$
Demorganise	$\overline{AB + AC} \quad (\overline{A + B})(\overline{A + C}) + A$
Multiply	$\overline{AA + AC + AB + BC + A}$
Reduce	$\overline{A + AC + AB + BC + A}$
Factorize	$\overline{A(1 + C) + AB + BC + A}$
Reduce	$\overline{A + AB + BC + A}$
Factorize	$\overline{A(1 + B) + BC + A}$

Reduce	$\overline{A + BC} + A$
Reduce	$\overline{1 + BC}$
Demorganise	$\overline{1 + \overline{B} + C}$
Reduce	$\overline{1 + \overline{B}}$
Reduce	$\overline{1}$
	0

5.20 Prove the following identities using Boolean algebra and DeMorgan's theorems.

(a) $\overline{AB + BC + CA} = \overline{AB} + \overline{BC} + \overline{CA}$

(b) $AB + \overline{AC} + \overline{ABC}(AB + C) = 1$

Solution:

(a)	$\overline{AB + BC + CA} = \overline{AB} + \overline{BC} + \overline{CA}$
L.H.S.	$= \overline{AB + BC + CA}$
	$= \overline{AB + (BC + CA)}$
Demorganise	$= \overline{AB}(\overline{BC + CA})$
Demorganise	$= \overline{AB}\{\overline{C(B + A)}\}$
	$= (\overline{A} + \overline{B})\{\overline{C} + \overline{(B + A)}\}$
Demorganise	$= (\overline{A} + \overline{B})(\overline{C} + \overline{BA})$
	$= \overline{AC} + \overline{ABA} + \overline{BC} + \overline{BBA}$
	$= \overline{AC} + \overline{AAB} + \overline{BC} + \overline{BBA}$
Reduce	$= \overline{AC} + \overline{AB} + \overline{BC} + \overline{BA}$
	$= (\overline{AB} + \overline{AB}) + \overline{BC} + \overline{CA}$
	$= \overline{AB} + \overline{BC} + \overline{CA} = \text{R.H.S.}$

(b) $AB + \overline{AC} + \overline{ABC}(AB + C) = 1$

L.H.S.	$= AB + \overline{AC} + \overline{ABCAB} + \overline{ABCAC}$
	$= AB + \overline{AC} + AAB\overline{BC} + AAB\overline{CC}$
	$= AB + \overline{AC} + 0 + A\overline{BC}$
	$= AB + \overline{ABC} + \overline{AC}$
	$= A(B + \overline{BC}) + \overline{AC}$
	$= A\{B(1 + C) + \overline{BC}\} + \overline{AC}$
	$= A(B + BC + \overline{BC}) + \overline{AC}$
	$= A\{B + C(B + \overline{B})\} + \overline{AC}$
	$= AB + AC + \overline{AC}$
	$= AB + (AC + \overline{AC})$
	$= AB + 1$
	$= 1 = \text{R.H.S.}$

5.21 Simplify the following Boolean function to a minimum number of literals $XY + \bar{X}Z + YZ$.

Solution:

$$\begin{aligned} XY + \bar{X}Z + YZ &= XY + \bar{X}Z + YZ(X + \bar{X}) \\ &= XY + \bar{X}Z + XYZ + \bar{X}YZ \\ &= XY + XYZ + \bar{X}Z + \bar{X}YZ \\ &= XY + (1 + Z) + \bar{X}Z(1 + Y) \\ &= XY + \bar{X}Z \end{aligned}$$

5.22 Find the complement of the functions $F_1 = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$ and $F_2 = A(\bar{B}\bar{C} + BC)$.

Solution:

Applying DeMorgan's theorem as many times as necessary, the complements are obtained as follows:

$$\begin{aligned} \bar{F}_1 &= \overline{(\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C)} \\ &= (\bar{A}\bar{B}\bar{C})(\bar{A}\bar{B}C) \\ &= (A + \bar{B} + C)(A + B + \bar{C}) \\ \bar{F}_2 &= \overline{A(\bar{B}\bar{C} + BC)} \\ &= \bar{A} + \overline{(\bar{B}\bar{C} + BC)} \\ &= \bar{A} + (\overline{\bar{B}\bar{C}})(\overline{BC}) \\ &= \bar{A} + (B + C)(\bar{B} + \bar{C}) \end{aligned}$$

5.23 Find the complement of the functions F_1 and F_2 of Problem 5.22 by taking their duals and complementing each literal.

Solution:

$$1. F_1 = \bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$$

The dual of F_1 is $(\bar{A} + B + \bar{C})(\bar{A} + \bar{B} + C)$

Complementing each literal:

$$(A + \bar{B} + C)(A + B + \bar{C}) = \bar{F}_1$$

$$2. F_2 = A(\bar{B}\bar{C} + BC)$$

The dual of F_2 is: $A + (\bar{B} + \bar{C})(B + C)$

Complement each literal:

$$\bar{A} + (B + C)(\bar{B} + \bar{C}) = \bar{F}_2$$

5.24 Simplify the expression $[A\bar{B}(C + BD) + \bar{A}\bar{B}]C$ as much as possible.

Solution:

$$\begin{aligned} &[A\bar{B}(C + BD) + \bar{A}\bar{B}]C \\ &= (A\bar{B}C + A\bar{B}BD + \bar{A}\bar{B})C \\ &= (A\bar{B}C + 0 + \bar{A}\bar{B})C \\ &= (A\bar{B}C + \bar{A}\bar{B})C \\ &= A\bar{B}CC + \bar{A}\bar{B}C \\ &= A\bar{B}C + \bar{A}\bar{B}C \\ &= \bar{B}C(A + \bar{A}) \\ &= \bar{B}C \cdot 1 \\ &= \bar{B}C \end{aligned}$$

5.25 Show that using Boolean algebra and DeMorgan's theorems:

$$\bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + \bar{W}XY\bar{Z} + WY\bar{Z} = Z$$

Solution:

$$\begin{aligned} \text{L.H.S.} &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + \bar{W}XY\bar{Z} + WY\bar{Z} \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z}(1 + Y) + \bar{W}XY\bar{Z} + WY\bar{Z} \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + \bar{W}\bar{X}Y\bar{Z} + \bar{W}XY\bar{Z} + WY\bar{Z} \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + \bar{W}Y\bar{Z}(\bar{X} + X) + WY\bar{Z} \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + \bar{W}Y\bar{Z} + WY\bar{Z} \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + Y\bar{Z}(\bar{W} + W) \\ &= \bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Z} + Y\bar{Z} \\ &= \bar{Y}\bar{Z} + Y\bar{Z} + \bar{W}\bar{X}\bar{Z} \\ &= \bar{Z}(\bar{Y} + Y) + \bar{W}\bar{X}\bar{Z} \\ &= \bar{Z} + \bar{W}\bar{X}\bar{Z} \\ &= \bar{Z} + (1 + \bar{W}\bar{X}) \\ &= \bar{Z} \cdot 1 \\ &= \bar{Z} = \text{R.H.S.} \end{aligned}$$

5.26 Simplify the following expression.

$$T(A, B, C) = (A + B) \left[\overline{A(\bar{B} + \bar{C})} \right] + \bar{A}\bar{B} + \bar{A}\bar{C}$$

Solution:

It is necessary to first apply DeMorgan's theorem and multiply out the expression in parentheses.

$$\begin{aligned} T(A, B, C) &= (A + B)(A + BC) + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= (A \cdot A + ABC + AB + BBC) + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A + ABC + AB + BC + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A + AB(C + 1) + BC + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A + AB + BC + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A(1 + B) + BC + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A + BC + \bar{A}\bar{B} + \bar{A}\bar{C} \\ &= A + BC + \bar{B} + \bar{C} \\ &= A + C + \bar{B} + \bar{C} \\ &= A + \bar{B} + 1 \\ &= 1 \end{aligned}$$

5.27 Prove the following Boolean identity.

$$ABC + \bar{A}\bar{B}C + \bar{A}BC + ABC + \bar{A}\bar{B}\bar{C} = \bar{A}\bar{B} + B(A + C)$$

Solution:

$$\begin{aligned} \text{L.H.S.} &= ABC + \bar{A}\bar{B}C + \bar{A}BC + ABC + \bar{A}\bar{B}\bar{C} \\ &= ABC + \bar{A}\bar{B}C + \bar{A}\bar{B}\bar{C} + \bar{A}BC + ABC \\ &= ABC + \bar{A}\bar{B}(C + \bar{C}) + \bar{A}BC + ABC \\ &= ABC + \bar{A}\bar{B} + \bar{A}BC + ABC \\ &= \bar{A}\bar{B} + ABC + \bar{A}BC + ABC \\ &= \bar{A}\bar{B} + BC(A + \bar{A}) + ABC \\ &= \bar{A}\bar{B} + BC + ABC \\ &= \bar{A}\bar{B} + B(C + AC) \\ &= \bar{A}\bar{B} + B(A + C) = \text{R.H.S.} \end{aligned}$$

5.28 Evaluate

$$f(W, X, Y, Z) = (WX + \overline{XY})Z + \overline{WX} + \overline{YZ} \text{ for } W = 1, X = 1, Y = 0, Z = 1.$$

Solution:

1. Place parentheses around the *complemented* expressions.

$$f(W, X, Y, Z) = (WX + (\overline{XY}))Z + (\overline{WX} + (\overline{YZ}))$$

2. Evaluate the *inner* parenthesized expressions, (\overline{XY}) and (\overline{YZ}) .

$$\overline{X} = \overline{1} = 0; \overline{XY} = 0 \cdot 0 = 0; \overline{XY} = \overline{0} = 1;$$

$$YZ = 0 \cdot 1 = 0; \overline{YZ} = \overline{0} = 1$$

3. Evaluate the *outer* parenthesized expressions,

$$WX = 1 \cdot 1 = 1; WX + \overline{XY} = 1 + 1 = 1;$$

$$\overline{WX} = 1 \cdot 0 = 0;$$

$$\overline{WX} + \overline{YZ} = 0 + 1 = 1; (\overline{WX} + \overline{YZ}) = \overline{1} = 0$$

4. Substitute the values of parenthesized expressions into the *remaining* expressions and evaluate the entire function.

$$f(W, X, Y, Z) = 1 \cdot Z + 0 = 1 \cdot 1 + 0 = 1 + 0 = 1$$

5.29 Evaluate

$$f(A, B, C, D) = AC + \overline{B(CD)} + (\overline{AB} + BC)D \text{ for } A = B = 1, C = D = 0.$$

Solution:

1. Place parentheses around the *complemented* expressions.

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

2. Evaluate the parenthesized expressions, proceeding from *inner to outer* expressions.

$$CD = 0 \cdot 0 = 0 \cdot 1 = 0, \quad B(CD) = 1 \cdot 0 = 0$$

$$(\overline{B(CD)}) = \overline{0} = 1, \quad \overline{AB} + BC = 1 \cdot \overline{1} + 1 \cdot 0 = 1 \cdot 0 + 1 \cdot 0 = 0 \cdot 0 = 0$$

$$(\overline{AB} + BC)0 = 0 \cdot 0 = 0, \quad ((\overline{AB} + BC)D) = \overline{0} = 1$$

3. Substitute the values of the parenthesized expressions into the *remaining* expressions and evaluate the entire function.

$$\begin{aligned} f(A, B, C, D) &= AC + 1 + 1 \\ &= 1 \cdot 0 + 1 + 1 \\ &= 0 + 1 + 1 \\ &= 1 \end{aligned}$$

Note: Whenever one factor of a *logical product* is 0, the product is 0. Whenever one term of a *logical sum* is 1, the sum is 1.

5.30 Simplify the expression $(\overline{A+B}) + \overline{A} \cdot \overline{B}$ by constructing a truth table.

Solution:

The *truth table* for the expression may be derived in the usual way.

Table 5.5

A	B	\overline{A}	$\overline{A} \cdot \overline{B}$	$\overline{A+B}$	$(\overline{A+B}) + \overline{A} \cdot \overline{B}$
0	0	1	0	1	1
0	1	1	1	0	1
1	0	0	0	0	0
1	1	0	0	0	0

An examination of the *last column* shows that

$$(\overline{A+B}) + \overline{A} \cdot \overline{B} = \overline{A}$$

Note: A considerable simplification of the original expression has been achieved. *This method of simplification forms the basis of another method called Karnaugh mapping.*

5.31 Simplify the expression

$$X = A \cdot (B + \overline{B} \cdot \overline{C}) + \overline{A} \cdot \overline{B} \cdot C$$

by constructing a truth table.

Solution:

Table 5.6

A	B	C	\overline{A}	\overline{B}	\overline{C}	$\overline{B} \cdot \overline{C}$	$\overline{A} \cdot \overline{B} \cdot C$	$B + \overline{B} \cdot \overline{C}$	$\overline{A} \cdot (B + \overline{B} \cdot \overline{C})$	X
0	0	0	1	1	1	1	0	1	0	0
0	0	1	1	1	0	0	1	0	0	1
0	1	0	1	0	1	0	0	1	0	0
0	1	1	1	0	0	0	0	1	0	0
1	0	0	0	1	1	1	0	1	0	0
1	0	1	0	1	0	0	0	0	1	1
1	1	0	0	0	1	0	0	1	0	0
1	1	1	0	0	0	0	0	1	0	0

Note, that $X = 1$ for $A = 0, B = 0$ and $C = 1$, i.e. $\overline{A} \cdot \overline{B} \cdot C$ and also when $A = 1, B = 0, C = 1$ i.e. $\overline{A} \cdot \overline{B} \cdot C$.

$$\therefore X = \overline{A} \cdot \overline{B} \cdot C + \overline{A} \cdot \overline{B} \cdot C = \overline{B}C (\overline{A} + A)$$

$$X = \overline{B} \cdot C$$

5.32 Use DeMorgan's laws to enable the following expressions to be implemented with a *single* logic gate. Use the inputs A, B , and C rather than $\overline{A}, \overline{B}$ and \overline{C} . Apply DeMorgan's laws to the expression and then select one of the following gates to implement it: OR; AND; NOR; NAND.

(a) $\overline{A} + \overline{B}$

(b) $\overline{A} \cdot \overline{B} \cdot \overline{C}$

Solution:

(a) By DeMorgan $\overline{\overline{A + B}} = A \cdot B$

$$\therefore \overline{A + B} = \overline{A \cdot B} \quad (\text{NAND function})$$

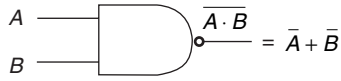


Fig. 5.12

(b) By DeMorgan $\overline{\overline{A \cdot B \cdot C}} = A + B + C$

$$\therefore \overline{A \cdot B \cdot C} = \overline{A + B + C} \quad (\text{NOR function})$$

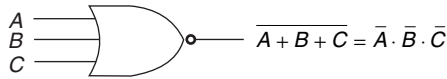


Fig. 5.13

5.33 Verify DeMorgan's law for the Boolean function $A + B = \overline{\overline{A \cdot B}}$ by examining truth tables.

Solution:

Table 5.7

A	B	\overline{A}	\overline{B}	$\overline{A \cdot B}$	$\overline{\overline{A \cdot B}}$	$A + B$
0	0	1	1	1	0	0
0	1	1	0	0	1	1
1	0	0	1	0	1	1
1	1	0	0	0	1	1

Note that $A + B = \overline{\overline{A \cdot B}}$, thus verifying one of the forms of DeMorgan's laws.

5.34 Verify, by examining truth tables, the form of DeMorgan's law that states $A \cdot B = \overline{\overline{A + B}}$

Solution:

Table 5.8

A	B	\overline{A}	\overline{B}	$\overline{A + B}$	$\overline{\overline{A + B}}$	$A \cdot B$
0	0	1	1	1	0	0
0	1	1	0	1	0	0
1	0	0	1	1	0	0
1	1	0	0	0	1	1

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

5.35 Use DeMorgan laws to convert the following expression to both the AND and the OR gate.

$$\overline{A \cdot B + C \cdot D}$$

Solution:

- (i) To convert to the AND form treat $\overline{A \cdot B}$ and $\overline{C \cdot D}$ as separate whole variables so that:

$$\overline{A \cdot B + C \cdot D} = \overline{\overline{A \cdot B} \cdot \overline{C \cdot D}}$$

- (ii) To convert to the OR form, transform the term $\overline{A \cdot B}$, then transform the term $\overline{C \cdot D}$.

$$\text{i.e.} \quad \overline{A \cdot B} = \overline{A} + \overline{B}$$

$$\overline{C \cdot D} = \overline{C} + \overline{D}$$

$$\therefore \overline{A \cdot B + C \cdot D} = \overline{\overline{A} + \overline{B} + \overline{C} + \overline{D}}$$

5.36 Simplify the expression

$$\overline{\overline{A \cdot B + C}} + \overline{\overline{A + B \cdot C}}$$

resulting from an initial study of a particular design problem, thereby saving unnecessary logic gates.

Solution:

Using DeMorgan's law we see that

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

and that $\overline{\overline{A + B \cdot C}} = A + \overline{B} \cdot \overline{C}$

Therefore the expression simplifies to

$$A \cdot \overline{B} \cdot \overline{C} + A + \overline{B} \cdot \overline{C}$$

A further simplification is possible since $A \cdot B$ is common to both terms of the expression.

$$\begin{aligned} A \cdot \overline{B} \cdot \overline{C} + A + \overline{B} \cdot \overline{C} &= A \cdot \overline{B} (C + \overline{C}) \\ &= A \cdot \overline{B} \text{ since } C + \overline{C} = 1 \end{aligned}$$

This is clearly much simpler than the original.

5.37 Using DeMorgan's laws, simplify

$$\overline{A + B} + (\overline{\overline{A \cdot C + B}}) + C$$

Solution:

$$\overline{A + B} + (\overline{\overline{A \cdot C + B}}) + C$$

DeMorgan's law applied to the terms in brackets gives:

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

The remainder of the expression is:

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

The total expression is now:

$$\overline{A \cdot B \cdot C} + A \cdot \overline{B} \cdot \overline{C}$$

which is of the form $\overline{X} + X$, where $X = A \cdot B \cdot \overline{C}$. So clearly the expression always has the value 1.

BOOLEAN EXPRESSIONS AND LOGIC DIAGRAMS

Boolean algebra is useless unless it can be translated into hardware. Similarly, Boolean algebra can be a useful technique for analysing existing circuits only if hardware can be translated into a Boolean expression. This capability develops with experience.

Boolean Algebra to Logic The easiest way to convert an expression into a logic diagram is to start with the output and work back toward the input.

Logic to Boolean Algebra The reverse process can be used for converting logic to Boolean algebra. Instead of proceeding from output to input, we start with the input signals and develop terms until the output is reached.

In logic, the following precedence rules apply:

1. Consider the NOT operator applied to an expression as putting parentheses around the expression. (Do not bother to put parentheses around a single complemented variable.)
2. Evaluate expressions within parentheses working from inner to outer parentheses in the following order:
 - (a) First evaluate NOTs of values from left to right.
 - (b) Then evaluate ANDs of pairs of values from left to right.
 - (c) Then evaluate ORs of pairs of values from left to right.
3. Substitute the values resulting from evaluations of parenthesized expressions into the remaining expressions and evaluate in the same order, that is:
 - (a) First evaluate NOTs from left to right.
 - (b) Then evaluate ANDs from left to right.
 - (c) Then evaluate ORs from left to right.

5.38 Write the Boolean equation for each of the logic circuits shown in Fig. 5.14.

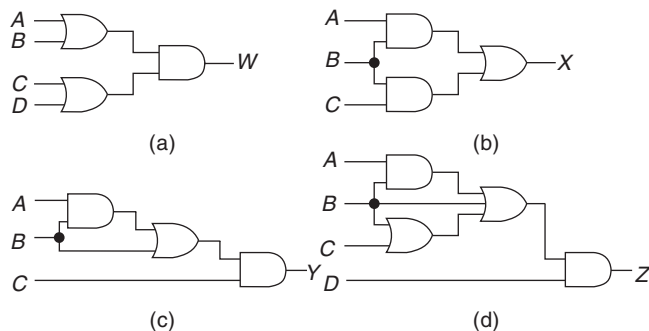


Fig. 5.14

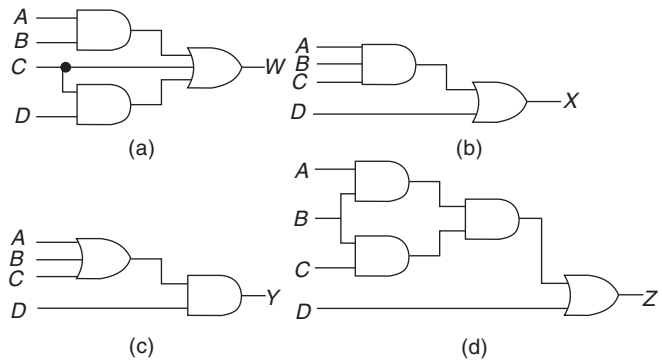
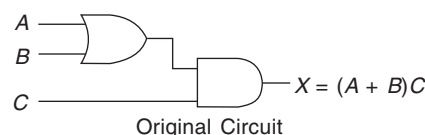
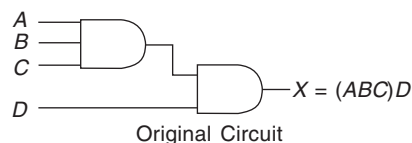
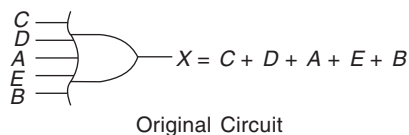


Fig. 5.15

Solution:

- (a) $W = (A + B)(C + D)$
- (b) $X = AB + BC$
- (c) $Y = (AB + B)C$
- (d) $Z = (AB + B) + (B + C))D$

5.39 Write the Boolean equation for each of the logic circuits shown in Fig. 5.15.

Solution:

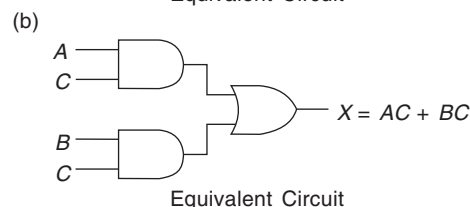
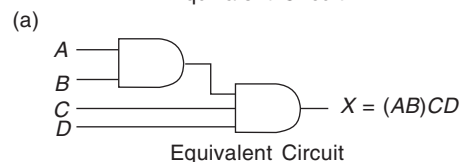
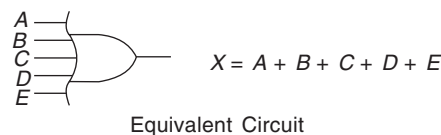
- (a) $W = (AB + C) + CD$
- (b) $X = ABC + D$
- (c) $Y = (A + B + C)D$
- (d) $Z = ((AB)(BC)) + D$

5.40 State the Boolean law that makes each of the equivalent circuits shown in Fig. 5.16 valid.

Solution:

- (a) Commutative law
- (b) Associative law
- (c) Distributive law

5.41 Write the Boolean equation for the circuits given in Fig. 5.17. Simplify the equations and draw simplified logic circuits.



(c)

Fig. 5.16

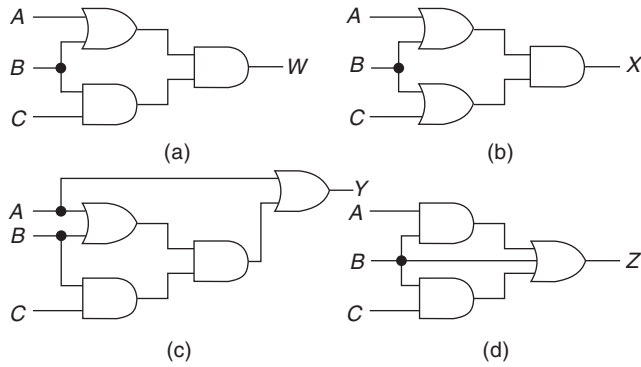


Fig. 5.17

Solution:

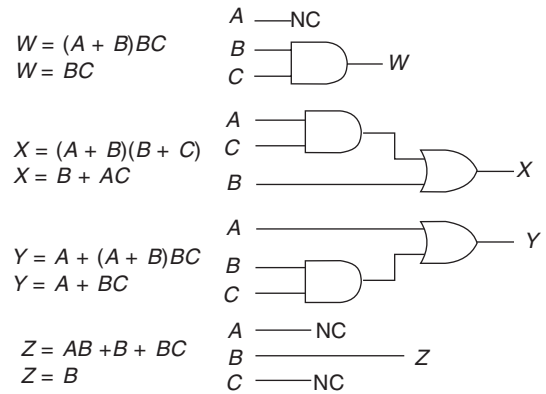


Fig. 5.18 Solution for Problem 5.41.

5.42 Draw the logic circuit for the following equations. Simplify the equations and draw the simplified logic circuits.

(a) $V = AC + ACD + CD$

(b) $W = (BCD + C)CD$

(c) $X = (B + D)(A + C) + ABD$

(d) $Y = AB + BC + ABC$

(e) $Z = ABC + CD + CDE$

Solution:

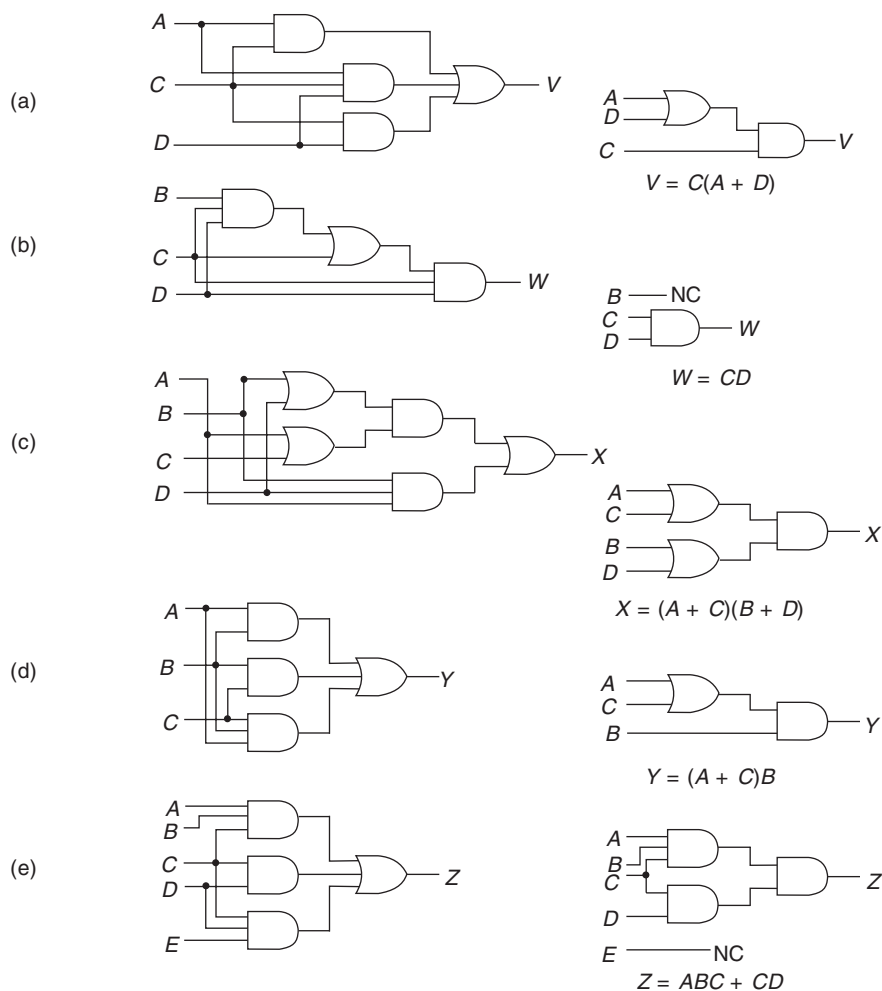


Fig. 5.19 Solution for Problem 5.42.

5.43 Draw the logic circuit for the following equations. Apply DeMorgan's theorem and Boolean algebra rules to reduce them to equations having inversion bars over single variables only. Draw the simplified circuit.

(a) $W = \overline{AB} + \overline{A} + C$

(b) $X = \overline{AB} + \overline{C} + \overline{BC}$

(c) $Y = \overline{(AB)} + \overline{C} + \overline{BC}$

(d) $Z = \overline{AB} + (\overline{A} + C)$

Solution:

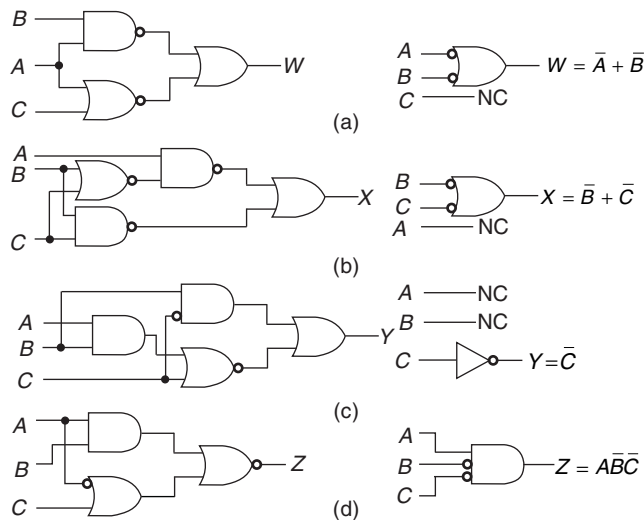


Fig. 5.20 Solution for Problem 5.43.

5.44 Repeat Problem 5.43 for the following equations.

(a) $W = \overline{\overline{AB} + CD + ACD}$

(b) $X = \overline{\overline{A} + B \cdot BC + \overline{BC}}$

(c) $Y = \overline{ABC + D + \overline{AB} + \overline{BC}}$

(d) $Z = \overline{(C + D) \overline{ACD} (\overline{AC} + \overline{D})}$

Solution:

(See Fig. 5.21).

5.45 Explain the bubble-pushing technique.

Solution:

Bubble-pushing, based on DeMorgan's theorem, is illustrated in Fig. 5.22. To form the equivalent logic circuit, you must

- Change the logic gate (AND to OR or OR to AND).
- Add bubbles to the inputs and outputs, where there were none and remove the original bubbles.

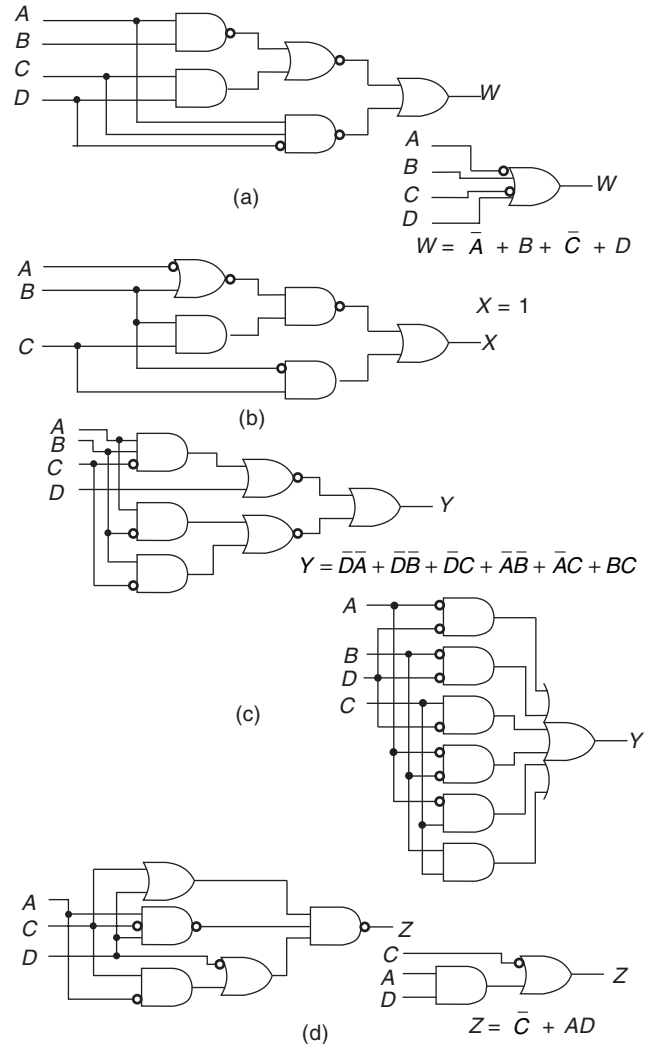


Fig. 5.21 Solution for Problem 5.44.

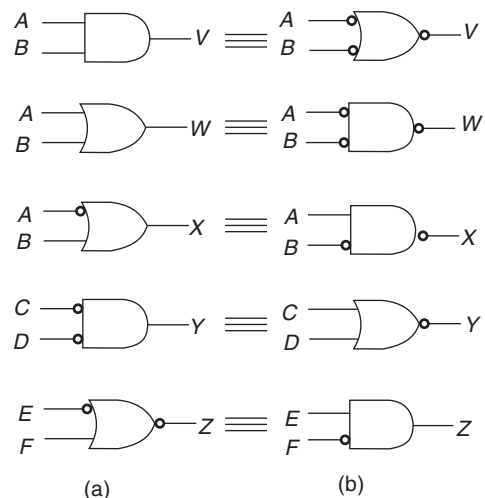


Fig. 5.22 (a) Original Logic Circuits; (b) Equivalent Logic Circuits.

5.46 Use the bubble-pushing technique to convert the gates in the Fig. 5.23.

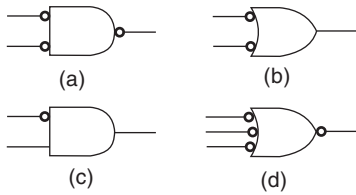


Fig. 5.23

Solution:

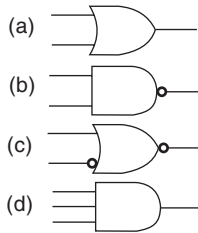


Fig. 5.24 Solution for Problem 5.46.

5.47 Realize the expression $Y = A\bar{B} + \bar{A}B$ using AND, OR, NOR gates.

Solution:

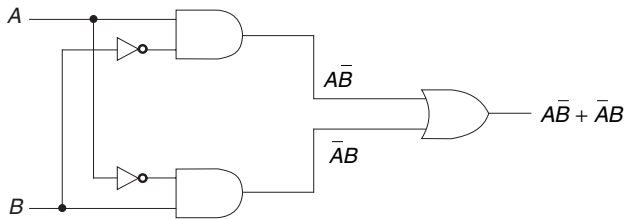


Fig. 5.25 Solution for Problem 5.47.

5.48 Realise the expression $Y = A\bar{B} + \bar{A}B$ using only NAND gates.

Solution:

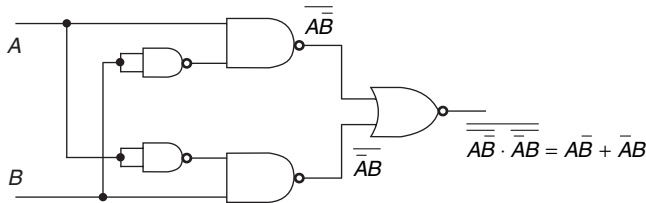


Fig. 5.26 Solution for Problem 5.48.

5.49 Draw the logic circuit for $Y = A\bar{B} + AB$. Simplify this Boolean equation and the corresponding logic circuit.

Solution:

We have a sum-of-products equation. This implies two AND gates driving an OR gate as shown below.

To simplify the logic circuit, factor the equation as follows:

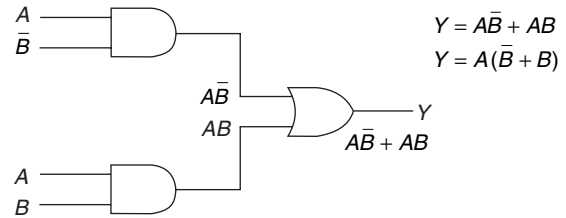


Fig. 5.27 Solution for Problem 5.49.

The corresponding logic circuit is shown below. This is simpler because it uses only one AND gate.

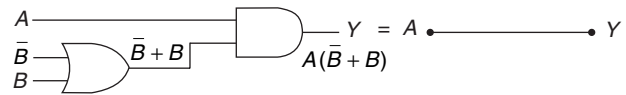


Fig. 5.27a Simplification of the Original Diagram.

5.50 Show the logic circuit for the Boolean equation

$Y = (\bar{A} + B)(A + B)$. Simplify the circuit as much as possible using algebra.

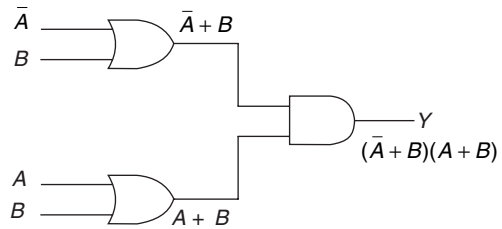


Fig. 5.28 Logic Circuit for $Y = (\bar{A} + B)(A + B)$

Solution:

Figure 5.28 shows the logic circuit for the given Boolean equation. Next, multiply the factors of the given equation to get

$$Y = \bar{A}A + \bar{A}B + BA + BB$$

A variable ANDed with its complement equals zero, so the first term drops out. A variable ORed with itself equals itself, so the last term reduces to B . Because of the *commutative law* $AB = BA$. The foregoing simplifications give us the following equation.

$$Y = \bar{A}B + AB + B$$

Figure 5.29 shows the corresponding logic circuit. As you can see, this circuit is more complicated than the original.

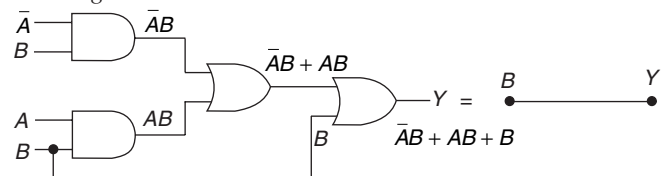


Fig. 5.29 Logic Circuit for $\bar{A}B + AB + B$

We can also factor the foregoing equation as follows:

$$Y = (\bar{A} + A)B + B = B + B = B$$

Since $Y = B$, we don't need a logic circuit. All we need is a wire connecting the input B to the output Y .

5.51 Implement the original and minimised expressions for the function

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

Solution:

$$\begin{aligned} f &= ABD + AB\bar{D} + \bar{A}C + \bar{A}BC + ABC \\ &= AB(D + \bar{D}) + \bar{A}C + BC(\bar{A} + A) \\ &= AB + \bar{A}C + BC \\ f &= AB + \bar{A}C \end{aligned}$$

The *original expression*, Figure 5.30(a), requires five AND gates and one OR gate for implementation. The *minimised expression* requires two AND gates and one OR gate, as can be seen in Figure 5.30(b). The number of literals have been reduced from 14 to 4, and the variable D has been eliminated.

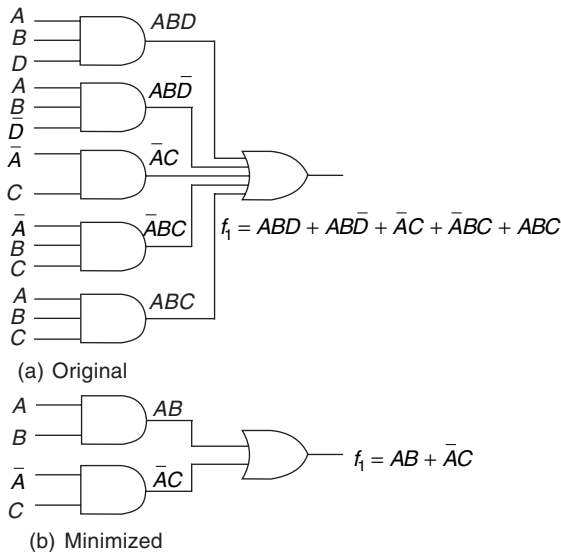


Fig. 5.30 Implementation of Original and Minimised Expression.

5.52 Implement the original and minimised expression for the function

$$f = A \cdot C + A \cdot D + B \cdot C + B \cdot D$$

Solution:

$$f = A \cdot C + A \cdot D + B \cdot C + B \cdot D$$

After simplification,

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

The corresponding circuit required to perform the logic function has been *reduced* from four, two-input AND gates and a four-input OR gate, to two, two-input OR gates and a two-input AND gate as shown in Fig. 5.31.

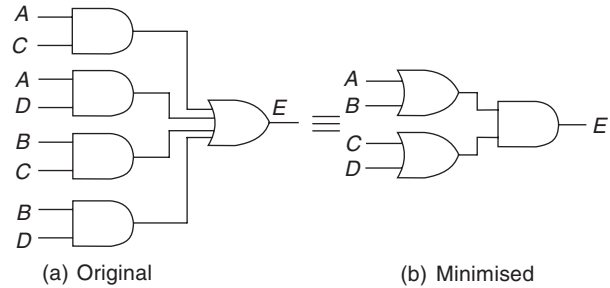


Fig. 5.31 Implementation of Original and Minimised Expression.

5.53 Implement the original and minimised expression for the function

$$Y = \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B$$

Solution:

$$Y = \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B$$

Table 5.9

A	B	\bar{A}	\bar{B}	$\bar{A} \cdot B$	$A \cdot \bar{B}$	$A \cdot B$	$Y = \bar{A} \cdot B + A \cdot \bar{B} + A \cdot B$
0	0	1	1	0	0	0	0
0	1	1	0	1	0	0	1
1	0	0	1	0	1	0	1
1	1	0	0	0	0	1	1

The truth table for the Boolean function is the truth function for a two-input OR gate. The simple Boolean expression for a two-input OR gate is $A + B = Y$.

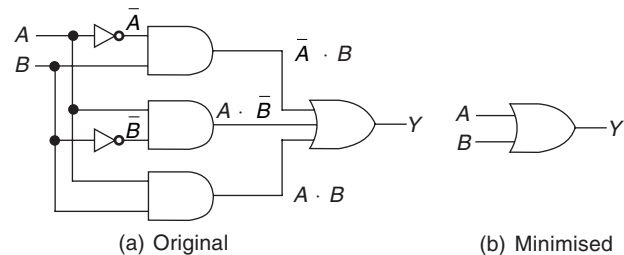


Fig. 5.32 Implementation of Original and Minimised Expression.

5.54 Implement the original and minimised expression for the function

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

Solution:

$$\begin{aligned} f &= \bar{A}\bar{B}C + B\bar{C} + \bar{A}BC + ABC \\ &= \bar{A}\bar{B}C + B\bar{C} + BC(\bar{A} + A) \\ &= \bar{A}\bar{B}C + B\bar{C} + BC \\ &= \bar{A}\bar{B}C + B(\bar{C} + C) \\ &= \bar{A}\bar{B}C + B \\ &= \bar{A}C + B \end{aligned}$$

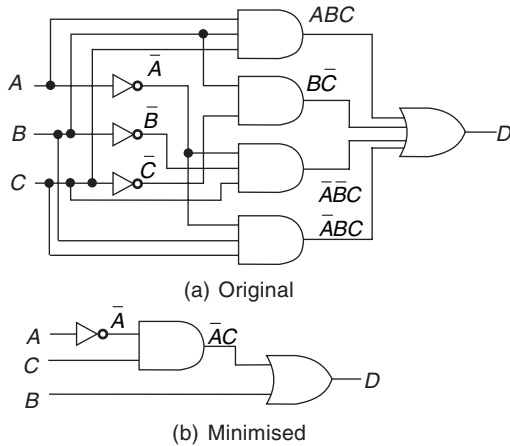


Fig. 5.33 Implementation of Original and Minimised Expression.

5.55 Implement the minimised Boolean expression for the function

$$f = B\bar{C}\bar{D} + \bar{A}BD + ABD + BC\bar{D} + \bar{B}CD + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D$$

Solution:

This can be implemented directly, as in the last example, but would use a large number of gates. Alternatively it may be simplified as follows.

$$\begin{aligned} f &= B\bar{C}\bar{D} + \bar{A}BD + ABD + BC\bar{D} + \bar{B}CD \\ &\quad + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D \\ &= BD(A + \bar{A}) + \bar{B}\bar{D}(C + \bar{C}) + \bar{B}CD + \bar{B}\bar{C}D(A + \bar{A}) \\ &= BD + \bar{B}\bar{D} + \bar{B}CD + \bar{B}\bar{C}D \\ &= B(D + \bar{D}) + \bar{B}\bar{D}(C + \bar{C}) \\ &= B + \bar{B}\bar{D} \\ &= B + D \end{aligned}$$

This expression can be implemented using a single gate.



Fig. 5.34 Implementation of Minimised Expression.

CANONICAL FORMS

The form of an expression determines how many and which type of logic gates are needed, as well as how they are connected together. *The more complicated the expression, the more complex will be the gate network. It is, therefore, best to simplify an expression as much as possible to get the simplest gate network.* There are two standard or *canonical forms* used to express any combinational logic network: the sum-of-products (SOP) form and the product-of-sums (POS) form.

5.56 Identify each of the following Boolean equations as a produce-of-sums (POS) expression, a sum-of-products (SOP) expression, or both.

- (a) $U = A\bar{B}C + BC + \bar{A}C$
- (b) $V = (A + C)(\bar{B} + \bar{C})$
- (c) $W = A\bar{C}(\bar{B} + C)$
- (d) $X = AB + \bar{C} + BD$
- (e) $Y = (A\bar{B} + D)(A + \bar{C}D)$
- (f) $Z = (A + \bar{B})(BC + A) + \bar{A}B + CD$

Solution:

- (a) SOP (b) POS (c) POS
- (d) SOP (e) POS (f) POS, SOP

5.57 Simplify the circuit shown in Fig. 5.35 down to its SOP form; then draw the logic circuit of the simplified form using a 74LS54 AOI gate.

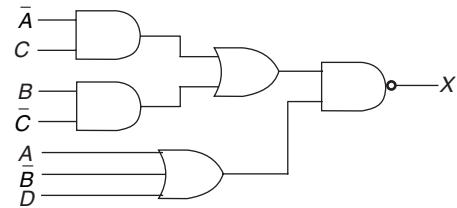


Fig. 5.35 Original Circuit for Problem 5.57.

Solution:

$$\begin{aligned} X &= (\bar{A}C + B\bar{C}) \cdot (A + B + D) \\ &= \bar{A}C + B\bar{C} + A + B + D \\ &= \bar{A}C \cdot \bar{B}\bar{C} + \bar{A}B\bar{D} \\ &= (A + \bar{C})(\bar{B} + C) + \bar{A}B\bar{D} \\ &= A\bar{B} + AC + \bar{B}\bar{C} + \bar{C}C + \bar{A}B\bar{D} \\ &= A\bar{B} + AC + \bar{B}\bar{C} + \bar{A}B\bar{D} \end{aligned} \quad \text{SOP}$$

The simplified circuit is shown in Fig. 5.36.

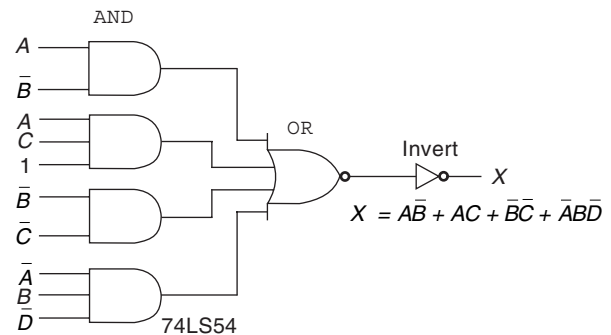


Fig. 5.36 Using an AOI IC to Implement the Simplified SOP Equation for Problem 5.57.

5.58 Implement the original and minimised expression for the function

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

Solution:

Since $X = X \cdot X$, we can expand f as follows:

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

Substitute $A + B$ for $(A + B + C)(A + B + \bar{C})$

$A + C$ for $(A + B + C)(A + \bar{B} + C)$

and $B + C$ for $(A + \bar{B} + C)(\bar{A} + \bar{B} + C)$

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

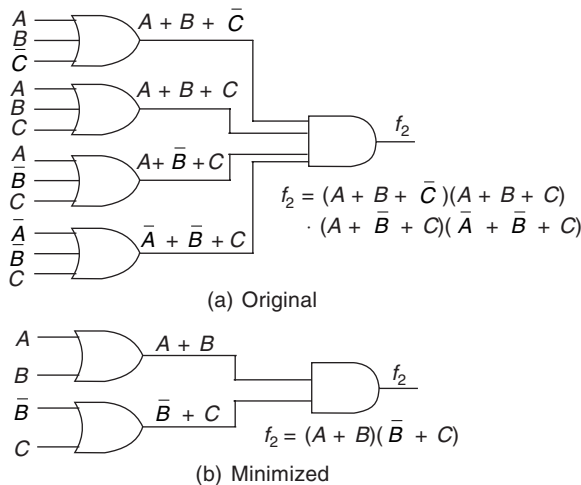


Fig. 5.37 Circuit for Original and Minimised Expression.

Note: Implementation of minimised expression requires two OR gates and one AND gate instead of four OR gates and one AND gate. Literals have been reduced from 12 to 4.

MINTERMS

An examination of the truth table verifies these results: With any row in a truth table we can associate two terms.

1. A minterm is the *logical product* for all literals in the row.
2. We get a minterm by *complementing* any variables that are 0 for the row and *leaving unbarred* all variables that are 1 for the row.
3. We form the logical product (AND) of all literals defined by this process.
4. The sum-of-products is known as the minterm form.

Table 5.10

A	B	X
0	0	0
0	1	1
1	0	1
1	1	0

Truth table $f(XOR)$ function

$(X = A \oplus B)$

$A = 0$ when $B = 1 \rightarrow \bar{A}B$

$A = 1$ when $B = 0 \rightarrow A\bar{B}$

$X = A\bar{B} + \bar{A}B = (A + B)(\bar{A} + \bar{B})$

SOP

POS

MAXTERMS

1. A maxterm is the *logical sum* for all literals in the row.
2. We get a maxterm by *barring* all variables that are 1 for the row and *leaving unbarred* all variables that are 0 for the row.
3. We form the logical sum (OR) of all literals defined by this process.
4. The product-of-sums is known as the maxterm form.

5.59 Convert $A + B$ to minterms.

Solution:

$$A + B = A(1) + B(1) \\ = A(B + \bar{B}) + B(A + \bar{A}) \\ = AB + A\bar{B} + AB + \bar{A}B \\ = AB + A\bar{B} + \bar{A}B$$

Each term in the example contains *all* the letters used: A and B . The terms AB , $A\bar{B}$, and $\bar{A}B$ are therefore *minterms*.

5.60 Find the minterms for $A + BC$ and prove that the result is $A + BC$.

Solution:

Write down the terms

$A + BC$

Insert X 's where letters are missing

AXX, XBC

Vary all the X 's in AXX

$A\bar{B}\bar{C}, A\bar{B}C,$

$AB\bar{C}, ABC$

Vary all the X 's in XBC

$\bar{A}BC, ABC$

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

Proof that the result is $A + BC$

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

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

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

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

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

$$= A\bar{B} + AB + BC$$

$$= A(\bar{B} + B) + BC$$

$$= A + BC$$

TWO-LEVEL REALISATION

An output function that is 1 for several rows of the truth table is written as a *logical sum* of minterms for all

rows for which the function is 1. This form is the canonical sum-of-products, AND-OR form. This realisation is known as two-level realisation. The *first level* consists of AND gates and the *second level* consists of OR gates. The Greek letter Σ is used to denote the sum. The decimal expression $\Sigma m(1, 2, 4)$ means that the minterms of rows 1, 2, and 4 are ORed together.

An output function that is 0 for several rows of the truth table is written as a *logical product* of maxterms for all rows for which the function is 0. This form is the canonical product-of-sums OR-AND form. This is also a two-level realisation. The *first level* consists of OR gates and the *second level* consists of the AND gate. The Greek letter Π is used for the product. The decimal expression $\Pi M(0, 6, 7)$ means that the maxterms of rows 0, 6 and 7 are ANDed together.

5.61 Draw a truth table for two variables. Map the minterms and denote the output.

Solution:

Table 5.11 Two-Variables Truth Table

INPUTS		OUTPUT
A	B	C
0	0	0
0	1	1
1	0	1
1	1	1

A	B	0	1
	0	0 ₀	1 ₁
1	1	1 ₂	1 ₃

Two-Variable Map

Each column of the truth table represents a minterm.

1's are placed to indicate that the output *contains* a particular minterm in its sum and 0's when that term is *excluded* from the sum.

$$C = \bar{A}B + A\bar{B} + AB$$

$$C = m_1 + m_2 + m_3$$

C is the sum of minterms 1, 2, and 3.

Another way of representing this relation is to use the Greek letter sigma and an *m* to represent "sum of minterms ..."

$$C = \Sigma m(1, 2, 3)$$

5.62 Draw the truth table for three variables showing minterms and maxterms.

Solution:

Table 5.12

Row Number	A	B	C	Minterms	Maxterms
0	0	0	0	$\bar{A}\bar{B}\bar{C}$	$A + B + C$
1	0	0	1	$\bar{A}\bar{B}C$	$A + B + \bar{C}$
2	0	1	0	$\bar{A}B\bar{C}$	$A + \bar{B} + C$
3	0	1	1	$\bar{A}BC$	$A + \bar{B} + \bar{C}$
4	1	0	0	$A\bar{B}\bar{C}$	$\bar{A} + B + C$
5	1	0	1	$A\bar{B}C$	$\bar{A} + B + \bar{C}$
6	1	1	0	$AB\bar{C}$	$\bar{A} + \bar{B} + C$
7					

5.63 For any truth table: (a) write the output function that has a 1 output for only one row; (b) write the output functions that has a 0 output for only one row.

Solution:

Table 5.13

Row Number	A	B	C	Function f_1
0	0	0	0	0
1	0	0	1	0
2	0	1	0	1
3	0	1	1	0
4	1	0	0	0
5	1	0	1	0
6	1	1	0	0
7	1	1	1	0

$$f_1 = \bar{A}\bar{B}\bar{C}$$

Table 5.14

Row Number	A	B	C	Function f_2
0	0	0	0	1
1	0	0	1	1
2	0	1	0	1
3	0	1	1	1
4	1	0	0	1
5	1	0	1	1
6	1	1	0	0
7	1	1	1	1

$$f_2 = \bar{A} + \bar{B} + C$$

5.64 Write the sum of products form.

Solution:

Table 5.15

Row Number	A	B	C	Function f_3	Minterms with output 1
0	0	0	0	0	
1	0	0	1	1	$\bar{A}\bar{B}\bar{C}$
2	0	1	0	1	$\bar{A}B\bar{C}$
3	0	1	1	0	
4	1	0	0	0	
5	1	0	1	1	$A\bar{B}\bar{C}$
6	1	1	0	1	$AB\bar{C}$
7	1	1	1	1	ABC

The sum-of-products form uses the minterms of rows for which the function is 1.

$$f_3 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC$$

$$= \Sigma m(1, 2, 5, 6, 7)$$

5.65 Write the product-of-sums form.

Solution:

Table 5.16

Row Number	A	B	C	Function f_3	Minterms with output 0
0	0	0	0	0	$A + B + C$
1	0	0	1	1	
2	0	1	0	1	
3	0	1	1	0	$A + \bar{B} + \bar{C}$
4	1	0	0	0	$\bar{A} + B + C$
5	1	0	1	1	
6	1	1	0	1	
7	1	1	1	1	

The product-of-sums form uses the maxterms of rows for which the function is 0.

$$f_3 = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + B + C) \\ = \text{PIM } (0, 3, 4)$$

CONVERTING CIRCUITS TO UNIVERSAL LOGIC

We can simply substitute NANDs for both ANDs and ORs in the AND-OR implementation. Similarly, we can substitute NORs for both ORs and ANDs in the OR-AND implementation. More than two levels may be required. Hence *either NAND or NOR is a complete set that can express any switching function*. The conversion can easily be accomplished by the following procedure:

1. Draw the circuit in AND-OR-Invert (AOI) logic
2. If NAND hardware has been chosen, add a circle to the output of each AND gate on the logic diagram, and provide circles on the inputs to all OR gates.
3. If NOR hardware has been chosen, add a circle to the output of each OR gate on the logic diagram, and provide circles on the inputs to all AND gates.
4. Add or subtract an inverter on each line that received a circle in step 2 or 3.

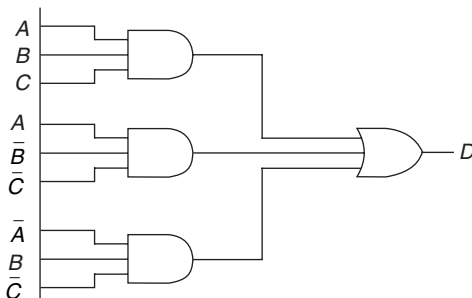
5.66 Implement the Boolean expression

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

directly; implement the same expression using only NAND gates.

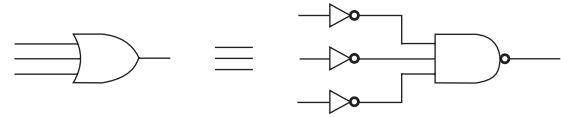
Solution:

The function $D = ABC + \bar{A}\bar{B}\bar{C} + \bar{A}BC$ can be implemented directly as shown in Fig. 5.38.

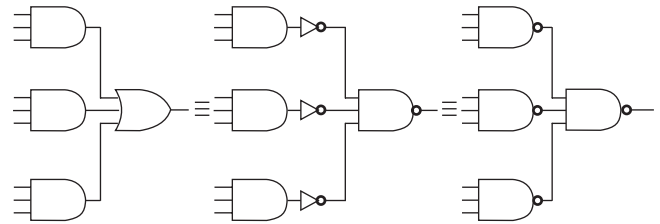
**Fig. 5.38** Direct Implementation of the Given Function.

However, from DeMorgan's theorem we know that

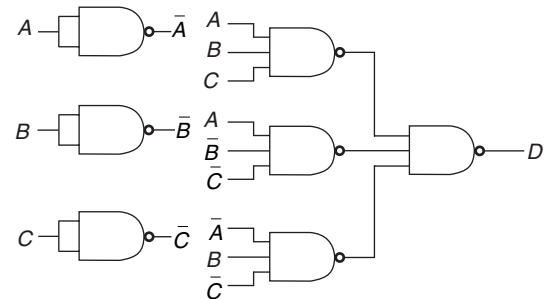
$$A + B + C = \overline{\bar{A}\bar{B}\bar{C}}$$

**Fig. 5.39** Direct Implementation of the Given Function.

It follows that functions implemented using AND and OR gates may be modified to use only NAND gates, since. (See Fig. 5.40)

**Fig. 5.40** Inversion of AND-OR to 'NAND Only'.

This simplification assumes that inverses of the inputs are available. If this is not the case they can be obtained by using NAND gates as inverters. In this way any logic function can be implemented using *only* NAND gates and our example is implemented as shown in Fig. 5.41.

**Fig. 5.41** Solution of Problem 5.66.

This manipulation can also be achieved using Boolean algebraic manipulation by noting that

$$D = ABC + \bar{A}\bar{B}\bar{C} + \bar{A}BC = \overline{\overline{ABC} \cdot \overline{\bar{A}\bar{B}\bar{C}} \cdot \overline{\bar{A}BC}}$$

This expression is in a form suitable for direct implementation using NAND gates only.

5.67 A manipulation similar to that given in the previous example can be used to implement functions using only NOR gates. Explain. (See Fig. 5.42)

Solution:

The inversions at the front end of this implementation are achieved simply by using the inverted or non-inverted signal as required. Therefore, using the previous example, the solution can be given as shown in Fig. 5.43.

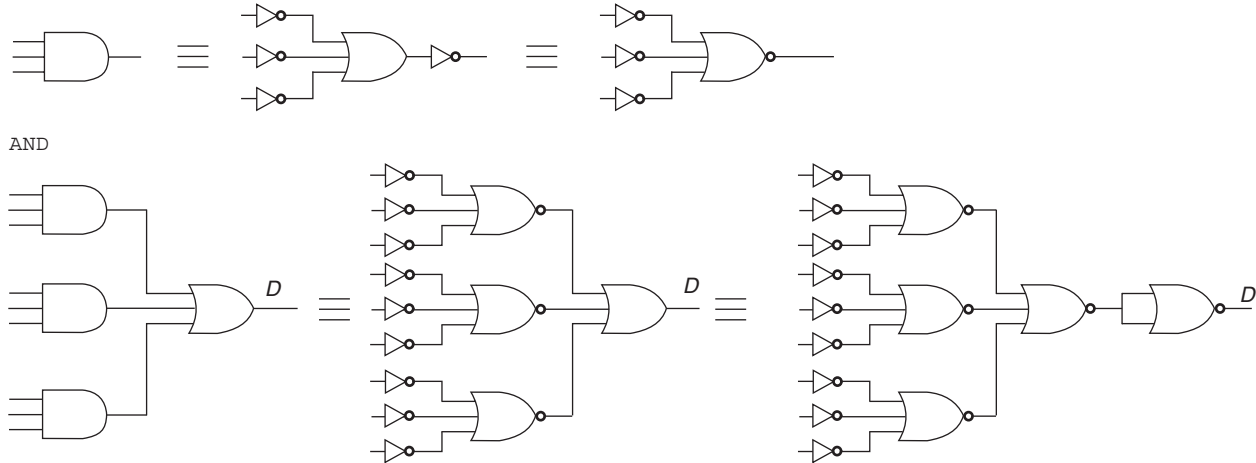


Fig. 5.42 Inversion of AND-OR to 'NOR Only'.

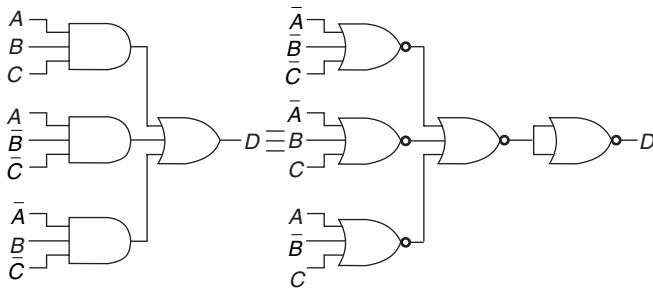


Fig. 5.43 Solution for Problem 5.67.

5.68 Simplify and implement the Boolean expression

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

Solution:

The expression may be implemented directly as shown in Fig. 5.44.

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

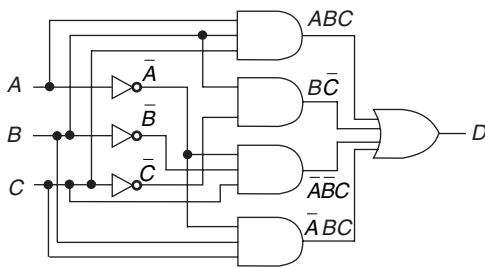


Fig. 5.44 Figure for Problem 5.68.

Alternatively, it can be reduced using Boolean algebra, as follows:

$$\begin{aligned} X &= \bar{A}\bar{B}C + B\bar{C} + \bar{A}BC + ABC \\ &= \bar{A}\bar{B}C + B\bar{C} + BC(\bar{A} + A) \\ &= \bar{A}\bar{B}C + B\bar{C} + BC \\ &= \bar{A}\bar{B}C + B(\bar{C} + C) \\ &= \bar{A}\bar{B}C + B \\ &= \bar{A}C + B \end{aligned}$$

This can be implemented using only three gates as shown in Fig. 5.45.

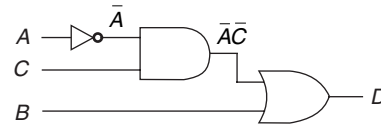


Fig. 5.45 Implementation of the Simplified Expression.

5.69 Simplify and implement the expression

$$B\bar{C}\bar{D} + \bar{A}BD + ABD + BC\bar{D} + \bar{B}CD + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D$$

Solution:

$$\begin{aligned} X &= B\bar{C}\bar{D} + \bar{A}BD + ABD + BC\bar{D} + \bar{B}CD \\ &\quad + \bar{A}\bar{B}\bar{C}D + A\bar{B}\bar{C}D \\ &= BD(A + \bar{A}) + B\bar{D}(C + \bar{C}) + \bar{B}CD \\ &\quad + \bar{B}\bar{C}D(A + \bar{A}) \\ &= BD + B\bar{D} + \bar{B}CD + \bar{B}\bar{C}D \\ &= B(D + \bar{D}) + \bar{B}D(C + \bar{C}) \\ &= B + \bar{B}D \\ &= B + D \end{aligned}$$

The expression can be implemented directly using a large number of gates. However, after simplification, the same expression can be implemented using a single gate as shown below.

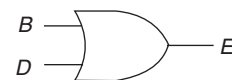


Fig. 5.46 Implementation of the Simplified Expression.

This represents a considerable simplification as compared with the original expression.

SUMMARY

- Boolean operators are the codes for the basic logic gates. You can use them as shorthand notation for digital logic circuits.
- Boolean expressions are not unique.
- The rules of Boolean algebra consist of a set of rules and a set of laws.
- Commutative law states that the elements of a function can be arranged in any sequence provided the connective is the same.
- Distributive laws allow the factoring or multiplying of expressions.
- Associative law states that in any Boolean function containing elements separated by the same connective, it does not matter if some of the elements are taken as a group.
- Absorption law is used for the elimination of redundant functions in a system.
- Idempotent law states that if a variable is ANDed or ORed with itself any number of times, the result will always be the original variable.
- If any number of nothings are added or multiplied together, the result will also be nothing.
- If any number of somethings are added or multiplied together, the result will also be something.
- The NAND operation is commutative and not associative.
- The NOR operation is commutative and not associative.
- Law of identity states that if $A = B$ and $B = C$, then $A = C$.
- Conjunction of a variable with logic 0 always yields a constant.
- Conjunction of a variable with logic 1 results in the original variable.
- Disjunction of a variable with logic 0 results in the original variable.
- Disjunction of a variable with logic 1 yields a constant.
- A NOR function can be implemented by inverting the two inputs to an AND function.
- A NAND function can be implemented by inverting the two inputs to an OR function.
- The complement of a sum is equal to the product of the complements.
- The complement of a product is equal to the sum of the complements.
- To convert a logic expression into a logic diagram, start with the output and work back towards the input.
- To convert a logic diagram to a logic expression start with the input signals and develop terms until the output is reached.
- To use the bubble-pushing technique: change the logic gate (AND to OR or OR to AND) and add bubbles to the input and outputs where there were none, and remove the original bubbles.
- To convert a Boolean expression into a logic diagram, start with the output and go back towards the input.
- To convert a logic diagram into a Boolean expression start with the input signals and develop terms until the output is reached.
- There are two canonical forms to express any combinational logic network: the sum-of-products (SOP) and the product-of-sums (POS).
- A minterm is the logical product for all literals in the row.
- A maxterm is the logical sum for all literals in the row.
- We can substitute NANDs for both ANDs and ORs in the AND-OR implementation.
- We can substitute NORs for both ORs and ANDs in the OR-AND implementation.

REVIEW QUESTIONS

Test your
understanding

1. Complete each expression:

(a) $A + 1 =$	(b) $A \cdot A =$	(c) $B \cdot \bar{B} =$	(d) $C + C =$
(e) $x \cdot 0 =$	(f) $D \cdot 1 =$	(g) $D \cdot 0 =$	(h) $C + \bar{C} =$
2. With the OR operation $1 + 1 = 1$
 $1 + 1 + 1 = ?$
3. With the AND operation $A \cdot A = A$
 $A \cdot A \cdot A = ?$

4. Apply the associative law to the expression $A + (B + C + D)$.
5. Apply the distributive law to the expression $A(B + C + D)$.
6. Draw the truth table for a three-input XOR gate.
7. Write the Boolean expression for a three-input NOR gates.
8. What is the significance of DeMorgan's theorem?
9. Apply DeMorgan's theorems to the following expression:
 (a) $\overline{ABC} + (\overline{D} + E)$ (b) $\overline{(A + B)C}$ (c) $\overline{A + B + C} + \overline{DE}$
10. How will you implement a NAND function?
11. How will you implement a NOR function?
12. Explain the difference between SOP and POS forms.
13. Give three examples of SOP expressions.
14. Give three examples of POS expressions.
15. Why is the SOP expression used more often?
16. How will you convert a Boolean expression into a logic diagram?
17. How will you convert a logic diagram into a Boolean expression?
18. Draw the logic circuit for the SOP expression $X = \overline{A}C + \overline{A}\overline{D} + BC + B\overline{D}$. What does it signify?
19. Draw the logic circuit for the POS expression $X = (\overline{A} + B)(C + \overline{D})$. What does it signify?
20. Give the pin configuration of AOI IC-74LS54.
21. Use an AOI IC to implement $\overline{X} = \overline{A}C + \overline{A}\overline{D} + BC + B\overline{D}$.

Test your understanding

SUPPLEMENTARY PROBLEMS

22. Write the output of the given logic circuit.
23. Construct a logic circuit to implement

$$f(A, B, C, D) = (\overline{A}BC) + D + (\overline{A}\overline{C})$$
24. How will you extract the Boolean expression of a function from its truth table?
25. Simplify the Boolean equation

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

Draw the logic circuit of the simplified expression. Compare it with the original circuit. (See Fig. 5.55(a))

26. Show the logic circuit for $Y = \overline{A}\overline{B} + AB$. Simplify the Boolean equation and the corresponding logic circuit.
27. Simplify the Boolean equation $Y = \overline{A}\overline{B}\overline{C} + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$, and describe the logic circuit.
28. Implement the function $Y = \overline{AB} + CDE + FG + HJK$ with AOI IC-74LS54.
29. Make the external connections to a 4001 CMOS NOR IC to implement the function $X = \overline{A} + B$.
30. Extract the Boolean expression for the circuit given in Fig. 5.48.
31. Develop the logic diagram of the Boolean expression

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

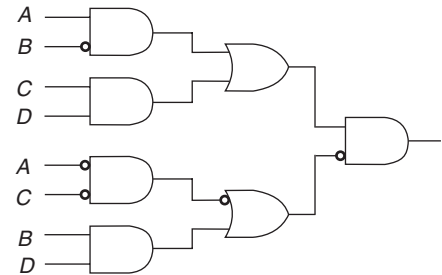


Fig. 5.47

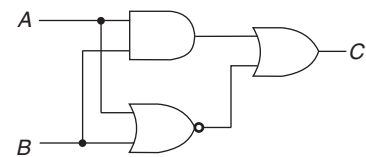


Fig. 5.48

Test your understanding

OBJECTIVE TYPE QUESTIONS

Fill in the Blanks

32. A variable may be ORed or ANDed with itself any number of times and the result will be the _____ variable.

33. A NOR function can be implemented by _____ the inputs to an AND function.
34. A NAND function can be implemented by inverting the inputs to an _____ function.
35. To _____ from Boolean algebra to logic, start with the output and work back toward the input.
36. To convert from logic to Boolean algebra _____ the circuit from input to output until the final expression is obtained.
37. Assertion level refers to the level _____ for an event to occur.
38. DeMorgan's laws are the most commonly used laws of _____.
39. Canonical forms are used to express any _____ logic function.
40. Minterm is a logical _____ for all literals in the row.
41. The sum-of-products is known as the _____ form.
42. Maxterms are _____ to minterms.
43. We can write the output function by _____ maxterms and minterms.
44. We can substitute _____ for both ANDs and ORs in the AND-OR implementation.
45. We can substitute _____ for both ORs and ANDs in the OR-AND implementation.
46. NAND or NOR is a complete set that can express any _____ function.
47. The number of terms in the expression corresponds to the number of _____.
48. The number of _____ in the expression corresponds to the number of inputs.
49. NAND and NOR gates are sometimes referred to as _____ gates.
50. The SOP form is _____ to deal with.
51. The Boolean operators can be considered as _____ for the basic gates.
52. Compound gates can be represented by _____ of the elementary functions.
53. The order in which terms are ANDed or ORed together is _____.
54. The distributive laws allows the _____ or multiplying of functions.
55. In any Boolean function containing elements separated by the same connective, it does not matter if some of these elements are considered as a _____.
56. 'Absorption law' is extremely important for the elimination of _____ functions.
57. If a function consists of a variable and its inverse, then the function is a _____.
58. Conjunction of a variable with logic 1 results in the _____ variable.
59. A variable plus _____ is always something.
60. A NOR function can be implemented by inverting the two inputs to an _____ function.
61. A _____ function can be implemented by inverting the two inputs to an OR function.
62. The complement of a product is equal to the _____ of the complements.
63. Inversion of a total function must be illustrated by adding an _____.
64. Each term in the two _____ forms contains each of the binary variables once and only once.
65. The Greek letter _____ is used for the logical product of maxterms.
66. Either NAND or NOR is a complete set that can express any _____ function.
67. A variable in complemented or uncomplemented form is known as a _____.

True/False Questions

State whether the following statements are True or False.

68. The elements of a function can be arranged in any sequence provided the logical connective is the same.
69. The associative law does not hold if elements are connected by the same connective.
70. When many conditions are to be ANDed or ORed together, the order in which conditions are combined is very important.
71. Idempotent law states that a variable may be ANDed or ORed with itself any even number of times and the result will still be the same variable.
72. If any number of nothings are added or multiplied together the result will also be nothing.
73. Conjunction of a variable with logic 0 always yields a constant.
74. Disjunction of a variable with logic 0 results in a constant output.
75. The complement of a sum is equal to the product of the complements.
76. The Greek letter sigma is used for the logical product of maxterms.
77. Either AND or OR is a complete set that can express any switching function.
78. The simplest Boolean equation results in the simplest array of logic gates for the function.
79. DeMorgan's laws are used to transform a whole expression.
80. Number of literals corresponds to number of inputs.

81. The SOP expression is seldom used because it does not lend itself to the development of truth tables and timing diagrams.
82. Boolean function cannot be simplified by graphical methods.
83. DeMorgan's laws are used to change only part of an expression in order to get the whole expression into a desirable form.
84. A minterm is a logical product of all literals in the row.
85. AND and OR gates are sometimes referred to as universal gates.

Multiple Choice Questions

86. Which of the following gates can be used as an inverter?
 - (a) NAND
 - (b) AND
 - (c) NOR
 - (d) None of the above
87. Which of the following operations is commutative but not associative?
 - (a) AND
 - (b) OR
 - (c) XOR
 - (d) NAND
88. Which of the following operations is not associative?
 - (a) NOR
 - (b) OR
 - (c) XOR
 - (d) AND
89. Identify the operation which is commutative but not associative.
 - (a) OR
 - (b) NOR
 - (c) XOR
 - (d) AND
90. The minterm designator of the term $\overline{A}\overline{B}\overline{C}\overline{D}$ is
 - (a) 8
 - (b) 9
 - (c) 10
 - (d) 11
91. The maxterm designator of the term $\overline{A} + \overline{B} + C + \overline{D}$ is
 - (a) 11
 - (b) 2
 - (c) 13
 - (d) 10
92. A switching function $f(A, B, C, D) = \overline{A}\overline{B}CD + \overline{A}B\overline{C}D + \overline{A}BCD + A\overline{B}\overline{C}D + A\overline{B}CD$ can also be written as
 - (a) $\Sigma m(1, 3, 5, 7, 9)$
 - (b) $\Sigma m(3, 5, 7, 9, 11)$
 - (c) $\Sigma m(3, 5, 9, 11, 13)$
 - (d) $\Sigma m(5, 7, 9, 11, 13)$
93. A switching function $f(A, B, C) = (A + \overline{B} + C)(A + \overline{B} + \overline{C})(\overline{A} + \overline{B} + C)$ can also be written as
 - (a) $\Pi m(1, 4, 5)$
 - (b) $\Pi m(2, 3, 6)$
 - (c) $\Pi m(0, 2, 3)$
 - (d) $\Pi m(3, 4, 5)$
94. A switching function $f(A, B, C, D) = \Sigma m(5, 9, 11, 14)$ can be written as
 - (a) $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}B\overline{C}\overline{D} + A\overline{B}\overline{C}\overline{D} + \overline{A}BC\overline{D}$
 - (b) $\overline{A}BC\overline{D} + \overline{A}\overline{B}CD + \overline{A}BCD + \overline{A}\overline{B}\overline{C}D$
 - (c) $\overline{A}\overline{B}\overline{C}\overline{D} + ABCD + \overline{A}\overline{B}CD + \overline{A}BCD$
 - (d) $\overline{A}BC\overline{D} + \overline{A}\overline{B}CD + \overline{A}BCD + ABCD$
95. The switching function $f(A, B, C, D) = \Pi m(5, 8, 11, 13)$ can also be written as
 - (a) $\overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}BCD + ABCD + \overline{A}\overline{B}CD$
 - (b) $(A + \overline{B} + C + \overline{D})(A + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + C + D)(A + B + \overline{C} + D)$
 - (c) $(\overline{A} + B + \overline{C} + D)(A + \overline{B} + \overline{C} + \overline{D})(A + \overline{B} + C + D)(\overline{A} + \overline{B} + C + \overline{D})$
 - (d) $ABCD + \overline{A}BCD + ABCD + \overline{A}\overline{B}CD$
96. If the SOP form of a switching function is $f(A, B, C) = \overline{A}\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\overline{C} + ABC$ then the POS form of the function will be
 - (a) $(\overline{A} + B + C)(A + \overline{B} + C)(A + B + \overline{C})(\overline{A} + B + \overline{C})$
 - (b) $(\overline{A} + \overline{B} + C)(\overline{A} + B + \overline{C})(A + \overline{B} + \overline{C})(A + B + C)$
 - (c) $(A + \overline{B} + C)(A + B + \overline{C})(\overline{A} + \overline{B} + C)(A + \overline{B} + C)$
 - (d) $(A + \overline{B} + C)(\overline{A} + \overline{B} + \overline{C})(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})$
97. The dual of the Boolean theorem $A \cdot (B + C) = A \cdot B + A \cdot C$ is
 - (a) $A + (B \cdot C) = A \cdot B + A \cdot C$
 - (b) $A \cdot (B + C) = (A + B)(A + C)$
 - (c) $A + (B \cdot C) = (A + B) \cdot (A + C)$
 - (d) None of the above

98. $\overline{A \cdot B \cdot C}$ is equivalent to
 (a) $\overline{A} + \overline{B} + \overline{C}$ (b) $\overline{A \cdot B \cdot C}$
 (c) $A + B + C$ (d) $A \cdot B \cdot C$
99. $\overline{A + B + C}$ is equivalent to
 (a) $A \cdot B \cdot C$ (b) $A + B + C$
 (c) $\overline{A \cdot B \cdot C}$ (d) $\overline{A} + \overline{B} + \overline{C}$
100. The dual of a Boolean function is obtained by
 (a) interchanging all 0s and 1s only
 (b) changing all 0s to 1s only
 (c) changing all 1s to 0s only
 (d) interchanging (i) all 0s and 1s and (ii) '+' and '·' signs
101. In a combinational circuit the outputs on any instant of time depend
 (a) only on inputs present at that instant of time
 (b) only on the past inputs
 (c) only on the past outputs
 (d) on past inputs as well as present inputs
102. A combinational circuit
 (a) never contains memory elements
 (b) always contains memory elements
 (c) may sometimes contain memory elements
 (d) contains only memory elements
103. The SOP form of logical expression is most suitable for designing logic circuits using only
 (a) XOR gates (b) AND gates
 (c) NAND gates (d) NOR gates
104. The POS form of logical expression is most suitable for designing logic circuits using only
 (a) XOR gates (b) NAND gates
 (c) AND gates (d) NOR gates
105. The logical expression $Y = AB + AC + BC$ is known as
 (a) standard SOP form (b) SOP form
 (c) standard POS form (d) POS form
106. The logical expression $Y = (A + B + C)(\overline{A} + C)(\overline{B} + C)(A + \overline{C})$ is known as
 (a) standard SOP form (b) SOP form
 (c) standard POS form (d) POS form
107. The logical expression $Y = \Sigma m(0, 3, 6, 7, 10, 12, 15)$ is equivalent to
 (a) $Y = \Pi M(0, 3, 6, 7, 10, 12, 15)$ (b) $\Pi M(1, 2, 4, 5, 8, 9, 11, 13, 14)$
 (c) $\Sigma m(1, 2, 4, 5, 8, 9, 11, 13, 14)$ (d) $\Sigma m(0, 2, 4, 6, 8, 10, 12, 14)$
108. The logic expression $Y = \Pi M(1, 4, 6, 9, 10, 11, 14, 15)$ is equivalent to
 (a) $\Pi M(0, 2, 3, 5, 7, 8, 12, 13)$ (b) $\Pi M(0, 2, 3, 4, 5, 6, 12, 13)$
 (c) $\Sigma m(1, 4, 6, 9, 10, 11, 14, 15)$ (d) $\Sigma m(0, 2, 3, 5, 7, 8, 12, 13)$
109. The minterm corresponding to decimal number 13 is
 (a) $A + B + \overline{C} + D$ (b) $\overline{A} + \overline{B} + C + \overline{D}$
 (c) $AB\overline{C}D$ (d) $\overline{A}\overline{B}\overline{C}\overline{D}$
110. The maxterm corresponding to decimal number 15 is
 (a) $ABCD$ (b) $\overline{A}\overline{B}\overline{C}\overline{D}$
 (c) $A + B + C + D$ (d) $\overline{A} + \overline{B} + \overline{C} + \overline{D}$

ANSWERS

1. (a) 1 (b) A (c) 0 (d) C (e) 0 (f) D (g) 0 (h) 1
 2. 1 3. A 4. $A(B + C + D) = (A + B + C) + D$
 5. $A(B + C + D) = AB + AC + AD$

6.

Table 5.17 Solution for Problem 6. Truth Table for a Three-input XOR Gate $(A \oplus B) \oplus C$

INPUTS			OUTPUT
C	B	A	X
0	0	0	0
0	0	1	1 ✓
0	1	0	1 ✓
0	1	1	0
1	0	0	1 ✓
1	0	1	0
1	1	0	0
1	1	1	1 ✓

7. $X = \overline{A + B + C}$.

8. DeMorgan's theorem formulates the relationship between $N(\text{AND})$ and $N(\text{OR})$ functions that allows one type of function to be implemented using a different type of function.

9. (a) $\overline{A} + \overline{B} + \overline{C} + D\overline{E}$ (b) $\overline{A}\overline{B} + \overline{C}$ (c) $\overline{A}\overline{B}\overline{C} + D + \overline{E}$

10. A NAND function can be implemented by inverting the two inputs to an OR function.

11. A NOR function can be implemented by inverting the two inputs to an AND function.

12. The SOP expression usually takes the form of two or more variables ANDed together, ORed with two more variables ANDed together.

The POS expression usually takes the form of two or more variable ORed together, ANDed with two or more variables ORed together.

13. $X = \overline{A}\overline{B} + AC + \overline{A}BC$

$$X = AC\overline{D} + \overline{C}D + B$$

$$X = B\overline{C}\overline{D} + \overline{A}BDE + CD$$

14. $X = (A + \overline{B}) \cdot (B + \overline{C})$

$$X = (B + \overline{C} + \overline{D}) \cdot (BC + \overline{E})$$

$$X = (A + \overline{C}) \cdot (\overline{B} + E) \cdot (C + B)$$

15. Because it lends itself nicely to the development of truth tables and timing waveforms, SOP circuits can also be constructed using a special combinational gate called AND-OR-INVERT (AOI) g.ate.

16. Start with the output and work back towards the input.

17. Start with the input signals and develop terms until the output is reached.

18. AND gates feeding into an OR gate. (See Fig. 5.49)

19. OR gates feeding into AND gate. (See Fig. 5.50)

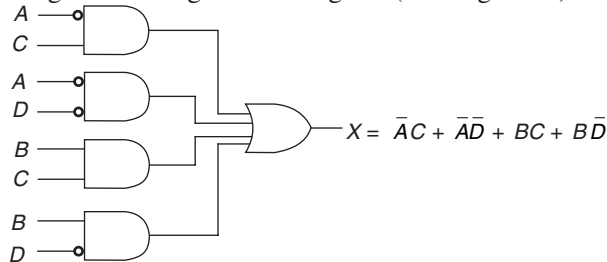


Fig. 5.49 Solution for Problem 18

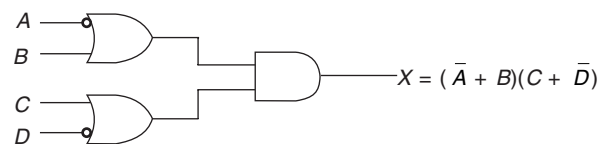


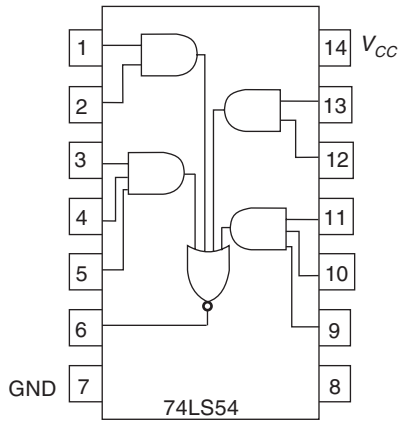
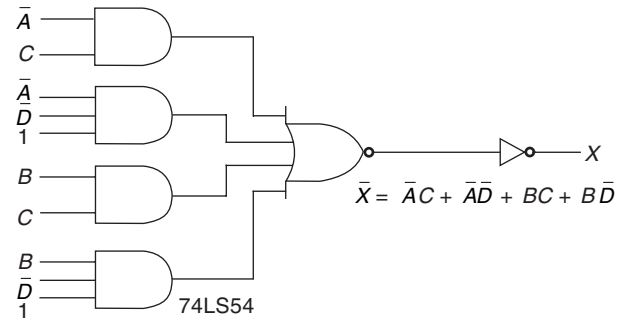
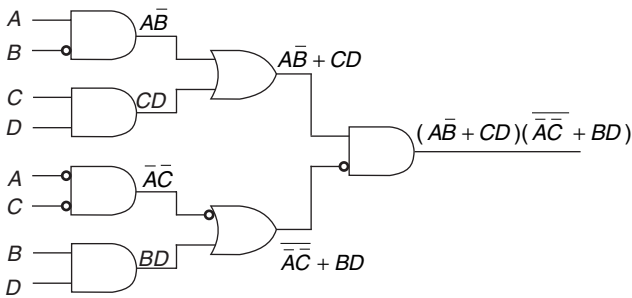
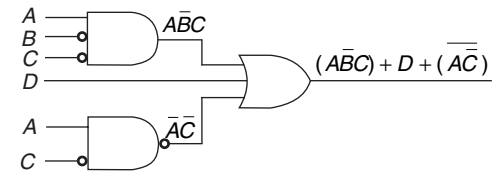
Fig. 5.50 Solution for Problem 19

20. (See Fig. 5.51)

21. (See Fig. 5.52)

22. (See Fig. 5.53)

23. (See Fig. 5.54)


Fig. 5.51 Solution for Problem 20

Fig. 5.52 Solution for Problem 21

Fig. 5.53 Solution for Problem 22

Fig. 5.54 Solution for Problem 23

24.

Table 5.18

A	B	C
0	0	0
0	1	1
1	0	1
1	1	0

C is true if B is true and A is not true, or if A is true and B is not true.

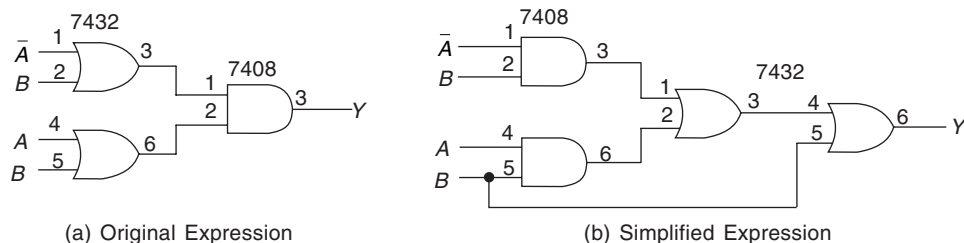
$$C = (\bar{A} \cdot B) + (A \cdot \bar{B}) = \bar{A}B + A\bar{B} \quad (\text{XOR function})$$

25. Simplified expression $Y = \bar{A}B + AB + B$

The circuit for the simplified expression is more complicated than that for the original expression. We can factor the foregoing equation as follows

$$Y = (\bar{A} + A)B + B = B + B = B$$

Since $Y = B$, we don't need a logic circuit. All we need is a wire connecting the input B to the Y output.


Fig. 5.55 Solution for Problem 25.

26. $Y = A (\bar{B} + B)$
 $= A$

We don't need a logic circuit. All we have to do is connect a wire between input A and output Y .

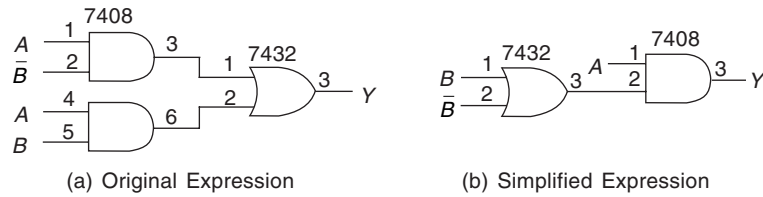


Fig. 5.56 Solution for Problem 26.

27. $Y = \bar{C}$ You don't need a logic circuit. All that is required is a wire connecting input \bar{C} to output Y .

28. (See Fig. 5.57)

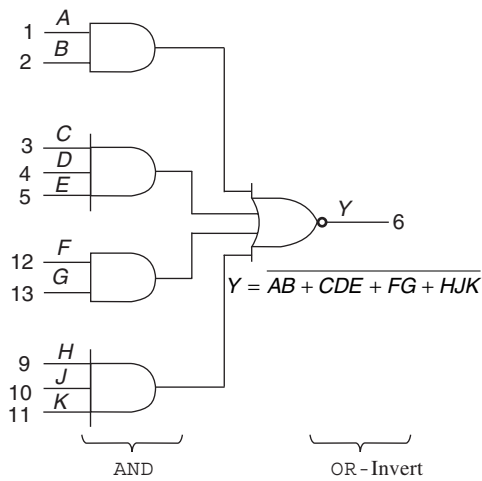


Fig. 5.57 Solution for Problem 28.

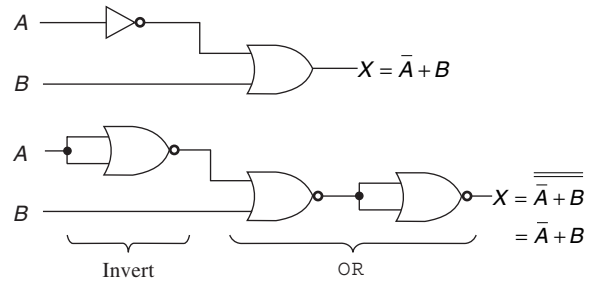


Fig. 5.58 NOR-NOR circuit for Problem 29.

29. We will need an inverter and an OR gate to provide the function for X . An inverter can be made from a NOR by connecting the inputs, and an OR can be made by inverting the output of a NOR, as shown in Fig. 5.58.

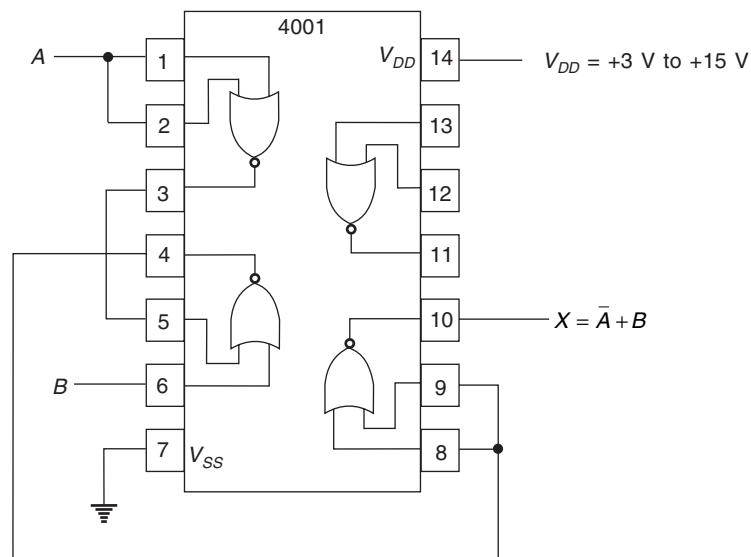
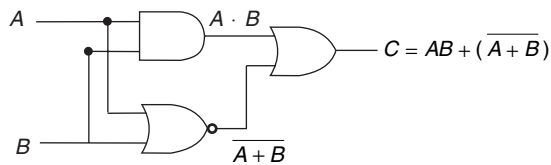
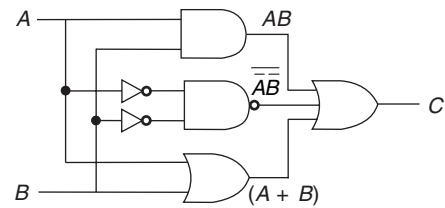


Fig. 5.59 Solution for Problem 29 External Connections to a 4001 CMOS IC to Implement $X = \bar{A} + B$

30. (See Fig. 5.60)

31. (See Fig. 5.61)

**Fig. 5.60** Solution for Problem 30**Fig. 5.61** Solution for Problem 31

- | | | | | | |
|---------------------|---------------|-------------------|-----------------|--------------|------------------|
| 32. original | 33. inverting | 34. OR | 35. convert | 36. analyse | 37. necessary |
| 38. Boolean algebra | | 39. combinational | | 40. product | 41. minterm |
| 42. complements | | 43. combining | 44. NANDs | 45. NORs | 46. switching |
| 47. logic gates | 48. literals | 49. universal | 50. easier | 51. codes | 52. combinations |
| 53. unimportant | 54. factoring | 55. group | 56. redundant | 57. constant | 58. original |
| 59. something | 60. AND | 61. NAND | 62. sum | 63. inverter | 64. Canonical |
| 65. Pi(Π) | 66. switching | 67. literal | 68. True | 69. False | 70. False |
| 71. False | 72. True | 73. True | 74. False | 75. True | 76. False |
| 77. False | 78. True | 79. False | 80. True | 81. False | 82. False |
| 83. True | 84. False | 85. False | 86. (a) and (c) | 87. (d) | 88. (a) |
| 89. (b) | 90. (c) | 91. (c) | 92. (b) | 93. (a) | 94. (d) |
| 95. (c) | 96. (d) | 97. (c) | 98. (c) | 99. (a) | 100. (d) |
| 101. (a) | 102. (a) | 103. (c) | 104. (d) | 105. (b) | 106. (d) |
| 107. (b) | 108. (d) | 109. (c) | 110. (d) | | |

Reduction Techniques

INTRODUCTION

The canonical expressions, two-level circuits, are usually not the most *economical implementation* of the logic functions. There are ways of simplifying or minimising logic functions to achieve economical implementations. *Simplifying* means finding an expression with fewer terms or fewer literals. *Minimising* means finding an expression that is best by some minimisation criteria. Usually, we try to minimise first the *number of terms* in the expression and then the *number of literals*. This classic procedure corresponds to minimising first the *number of logic gates* and then the *number of inputs* in the implementation. There are three methods of minimising:

1. Algebraic simplification (discussed in detail in Chapter 5)
2. Karnaugh map
3. Quine McClusky tables

6.1 Find the minterms for $AB + AC$.

Solution:

$ABXX$ generates $AB\bar{C}\bar{D}$, $AB\bar{C}D$, $ABC\bar{D}$, $ABCD$

$AXCD$ generates $A\bar{B}\bar{C}D$, $ABCD$

$$AB + CD = AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD + A\bar{B}\bar{C}D + ABCD$$

6.2 Find the minterm designation of $A\bar{B}\bar{C}\bar{D}$.

Solution:

Copy original term $A\bar{B}\bar{C}\bar{D}$

Substitute 1's for *non-barred* letters and 0's for *barred* letters 1000

Express as decimal subscript of m m_8

$$A\bar{B}\bar{C}\bar{D} = m_8$$

6.3 Find the minterm designation of $\bar{W}\bar{X}Y\bar{Z}$.

Solution:

Copy original term $\bar{W}\bar{X}Y\bar{Z}$

Convert to binary 1000

Express as decimal subscript of m m_2

$$\bar{W}\bar{X}Y\bar{Z} = m_2$$

6.4 How will you implement a specified network using logic gates? The block diagram and truth table of the network are given in Fig. 6.1.

Solution:

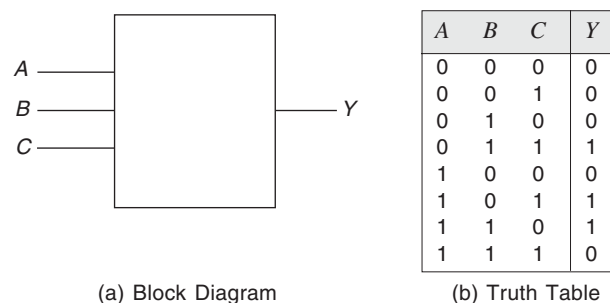


Fig. 6.1

The required function can be described in words by saying that the *output should be 1 if exactly two of the three input signals are 1 and should be 0 otherwise*. This condition is met by only three combinations of the input signals as shown in the truth table.

The function can be realised by constructing three gate networks to detect each of the combination of the inputs and then ORing the outputs from these three networks.

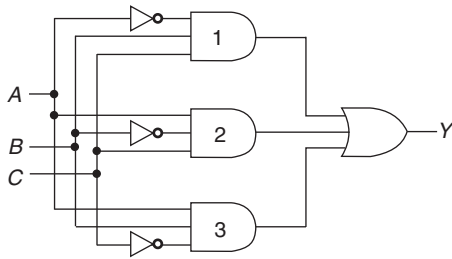


Fig. 6.2 Solution for Problem 6.4.

KARNAUGH MAPPING

If the aim is simply to eliminate any redundant terms AND obtain the simplest equation to express a given function, then *the graphical methods give quick and positive results*. The graphical method most commonly used is the *Karnaugh map*, also called the *Veitch diagram*.

6.5 Explain the significance of Karnaugh mapping.

Solution:

A Karnaugh map is similar to a truth table in that it graphically shows the output level of a Boolean equation for each of the possible input variable combinations. A Karnaugh map is simply *a graphical method of applying the laws of complementation and absorption*. Each output level is placed in a separate *cell* (square) of the Karnaugh-map. Karnaugh-maps can be used to simplify equations having two to six different input variables. Solving five-and six-variable Karnaugh-maps is extremely cumbersome; they can be more practically solved using advanced computer techniques.

6.6 How will you construct an n -variable K-map?

Solution:

Determining the number of cells in a K-map is the same as finding the number of combinations are entries in a truth table. An n -variable map requires 2^n cells. This is shown in Fig. 6.3.

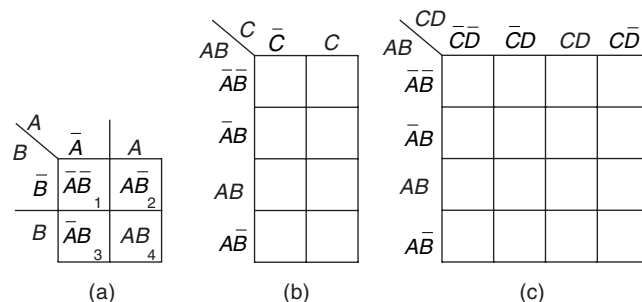


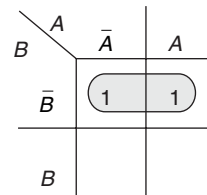
Fig. 6.3 (a) Two-variables K-maps (b) Three-variable K-maps (c) Four-variables K-maps.

Each cell in the K-map corresponds to a particular *minterm* (a particular combination of the input variables). In the *two- variable* K-map, for example, the upper left cell corresponds to $\bar{A}\bar{B}$ (00), the lower left

cell is $\bar{A}B$ (01), the upper right cell is $A\bar{B}$ (10) and the lower right cell is AB (11). These four terms represent all possible conjunctions of the two variables and their complements.

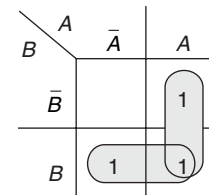
6.7 What are the basic rules that must be followed when using Karnaugh maps?

Solution:



Law of Complementation

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



Law of Absorption

$$\begin{aligned} \bar{A}\bar{B} + AB + BA + AB \\ = A(\bar{B} + B) + B(\bar{A} + A) \\ = A + B \end{aligned}$$

Fig. 6.4

The *basic rules* that must be followed when using Karnaugh maps are as follows:

1. Transform the Boolean equation to be reduced into an SOP expression.
2. The map is drawn up in such a way that *the terms in adjacent cells differ by only one variable*.
3. The terms in the equation to be simplified are entered by writing 1's in the appropriate cells in the map.
4. Where cells that are adjacent horizontally or vertically both contain a 1, *the variable that changes between the cells may be dropped* (by the law of complementation), leaving only the remainder of the term, common to both cells, as part of the final answer. *Whole terms may also disappear by being absorbed*.
5. When all the terms have been simplified, *the final equation is obtained by writing down all the simplified terms and connecting them by disjunction (OR)*.

THREE-VARIABLE MAPS

Although the two-variable map illustrates the principle of Karnaugh mapping, it is too trivial to show the full potential and properties of Karnaugh map, nor the procedure for drawing *multivariable maps*. A three-variable Karnaugh map is shown in Fig. 6.5 and consists of *four columns* and *two rows*. Combinations of the first two variables are written down at the head of each column, whilst the two possibilities for the third variable define the two rows. The map could also be drawn the other way, with *two columns* and *four rows* (see Fig. 6.3(b)).

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

Fig. 6.5 A Three-variable Karnaugh Map.

The rule that *only a one-variable change is allowed* must be followed. The extreme left-hand column is taken as being adjacent to the extreme right-hand column and the extreme top row is taken as adjacent to the extreme-bottom row. The rule must also be observed between these columns and rows.

6.8 Reduce the Boolean equation to an SOP expression. Draw a truth table of the SOP expression and then transfer it to a Karnaugh map. Verify your result.

$$X = \bar{A}(\bar{B}C + \bar{B}\bar{C}) + \bar{A}BC$$

Solution:

First transform the equation to an SOP expression:

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

Draw a truth table for the SOP expression and transfer to a K-map (Fig. 6.6).

A	B	C	X
0	0	0	1
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

← $(\bar{A}\bar{B}\bar{C})$
 ← $(\bar{A}\bar{B}C)$
 ← $(\bar{A}B\bar{C})$

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

Fig. 6.6 Truth Table and Karnaugh Map of the Expression.

Now encircle adjacent 1's as shown in Fig. 6.7. The simplified equation is obtained by determining which variables remain the same within each loop. $\bar{A}\bar{B}$ becomes one of the terms in the SOP expression. The second term in the SOP expression is $\bar{A}\bar{C}$. The final result in SOP format is $X = \bar{A}\bar{B} + \bar{A}\bar{C}$. This can be verified by algebraic simplification.

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

Fig. 6.9 Loops in a Three-variable K-map.

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

$$X = \bar{A}\bar{B} + \bar{A}\bar{C}$$

Fig. 6.7 Encircling Adjacent Cells in Karnaugh Map.

6.9 Simplify the following SOP equation using the Karnaugh map technique.

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

Solution:

Construct an eight-cell K-map (Fig. 6.8) AND fill a 1 in each cell that corresponds to a term in the original equation. $\bar{A}\bar{B}$ has no C variable. Therefore, $\bar{A}\bar{B}$ will fit in two cells: $\bar{A}\bar{B}\bar{C} + \bar{A}\bar{B}C$. Encircle adjacent cells in the largest group of two or four or eight. Identify the variables that remain the same within each circle and write the final simplified SOP equation by ORing them together.

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

$$X = \bar{A}\bar{B} + \bar{C}$$

Fig. 6.8 Karnaugh Map AND Final Equation for Problem 6.9.

LOOPS

The principle of eliminating variables by the law of complementation can be extended beyond two cells. In Fig. 6.9, a 1 appears in every cell of row C . Hence, both the variables A and B can be dropped, leaving C . Another combination of four cells that eliminates two variables is a square array of 2×2 cells, Fig. 6.10. Here the variable B changes horizontally and is eliminated, while the variable C changes vertically, and is eliminated, leaving only A out of these four terms.

	AB	$\bar{A}\bar{B}$	$A\bar{B}$	$\bar{A}B$
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$
\bar{C}	$\bar{A}B\bar{C}$	$\bar{A}BC$	$AB\bar{C}$	ABC
C	$\bar{A}\bar{B}\bar{C}$	$\bar{A}\bar{B}C$	$A\bar{B}\bar{C}$	$A\bar{B}C$

Fig. 6.10 A Square Array of 2×2 Cells.

To make it easier to see which terms may be grouped together for simplification loops are drawn around them. The *permitted* sizes of loops are any powers of two with the provision that the loop must be either *square* or *rectangular*. L-shaped or any other shape are not considered as single loops. An array of 3, 5, 6, 7, 9 squares etc. must be made up of two or more smaller loops. It is best to draw the *largest possible loop* in order to obtain the simplest equation. *Intersection* of two loops is allowed. (Fig. 6.11).

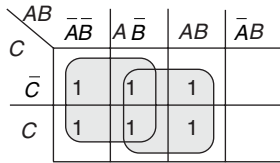


Fig. 6.11 Intersection of Two 4-square Loops.

A *pair* eliminates one variable that changes form, Fig. 6.4. A *quad* eliminates two variables and their complements, Figs. 6.9 and 6.10. An *octet* eliminates three variables and their complements, Fig. 6.12.

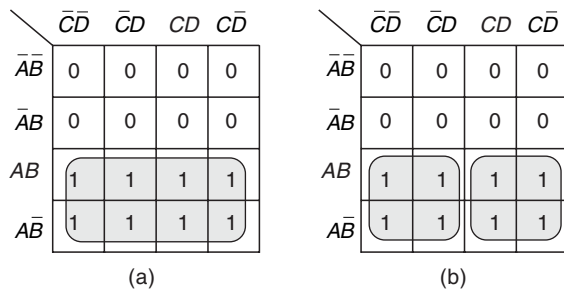


Fig. 6.12 Examples of Octet.

6.10 Draw the K-map for the given expression and make the groupings.

$$\bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + A\bar{B}C + \bar{A}BC + ABC$$

Solution:

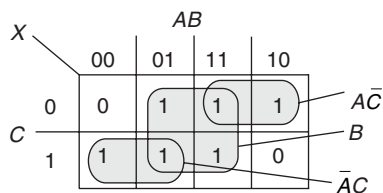


Fig. 6.13 Solution for Problem 6.10.

6.11 Draw a three-variable K-map for the expression

$$F = \bar{A}BC + AB(\bar{C} + C)$$

Solution:

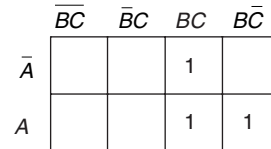


Fig. 6.14 Solution for Problem 6.11.

Encircle adjacent cells in the largest group of two or four or eight. Identify the variables that remain the same within each circle and write the final SOP equation by ORing them together.

6.12 Show the factoring or grouping for 1's for the expression

$$F = \bar{A}BC + AB(\bar{C} + C)$$

Solution:

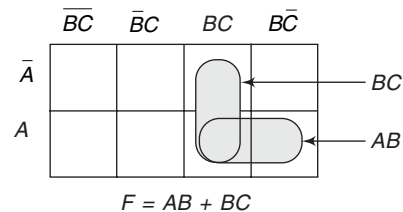


Fig. 6.15 Solution for Problem 6.12.

FOUR-VARIABLE KARNAUGH MAPS

Four-variable Karnaugh maps are drawn in much the same way as three-variable maps. A matrix of 4×4 cells is drawn up, combinations of the first two variables are written at the tops of the columns and combinations of the other variables at the side of the rows.

It should be noted that *as the end columns and top and bottom rows are taken as being adjacent, end-around and four-corner loops are also permitted*. However these possibilities are more pronounced on a larger map. Several possible loops up to eight cells in size are shown on the four-variable maps of Fig. 6.16.

6.13 Simplify the following SOP equation using K-map technique.

$$X = \bar{A}\bar{B}\bar{D} + \bar{A}B\bar{D} + \bar{A}BD + BCD + ABC\bar{D}$$

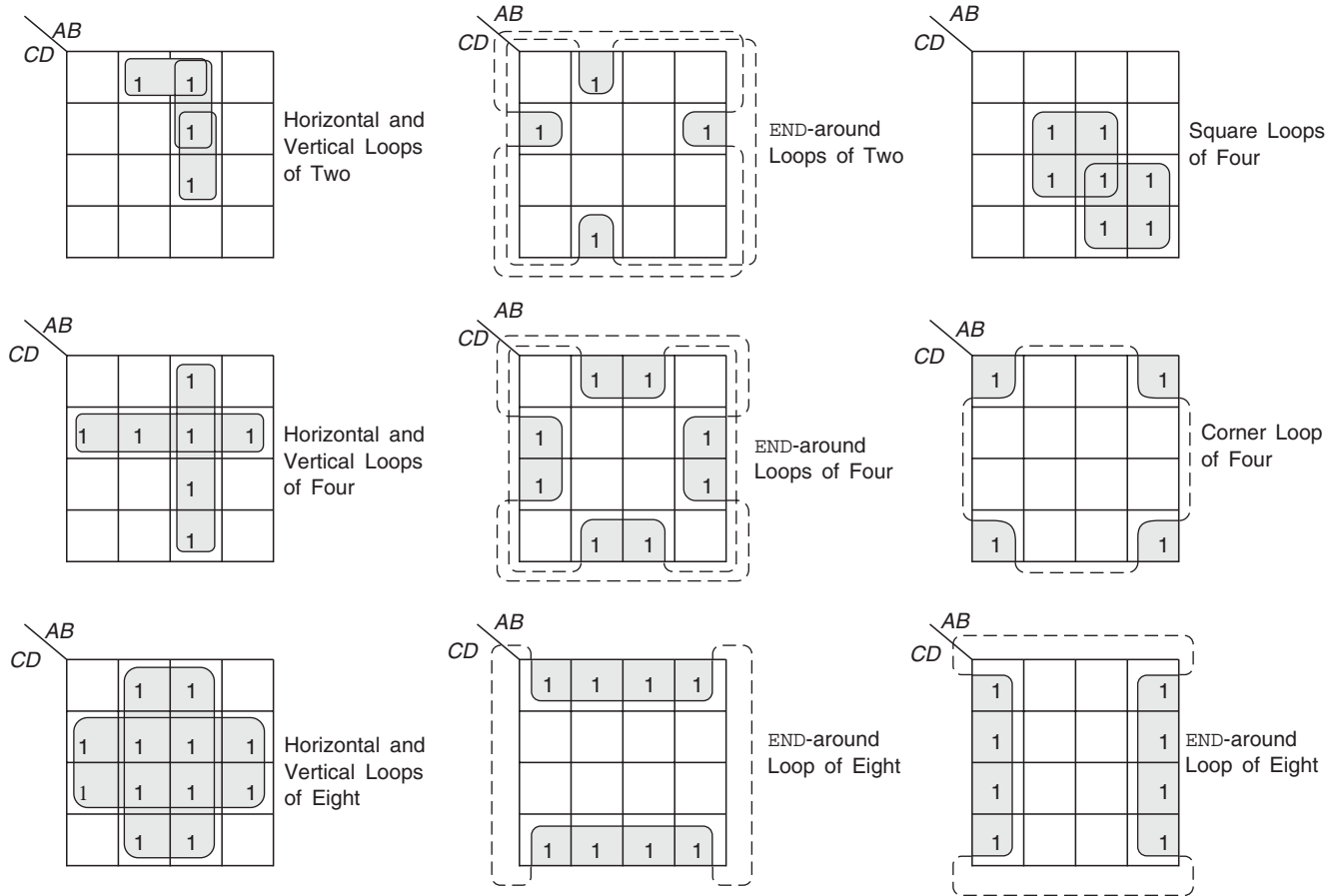


Fig. 6.16 Several Possible Loops up to Eight Cells in Size on Four-Variable Karnaugh Maps.

Solution:

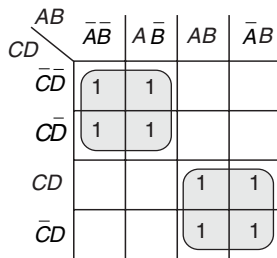


Fig. 6.17 Solution for Problem 6.13.

Identify the variables that remain the *same* within each circle and write the final simplified SOP equation by ORing them together.

$$X = \bar{B}\bar{D} + BD$$

6.14 Simplify the following equation using Karnaugh mapping procedure:

$$X = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}B\bar{C}\bar{D}$$

Solution:

Since there are four different variables in the equation, we need a 16-cell map ($2^4 = 16$) as shown in Fig. 6.18.

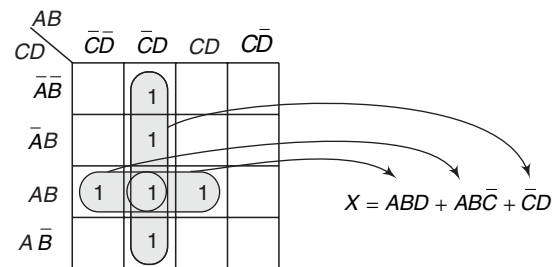


Fig. 6.18 Solution for Problem 6.14.

6.15 Solve the following equation using the Karnaugh mapping procedure:

$$X = \bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D$$

Solution:

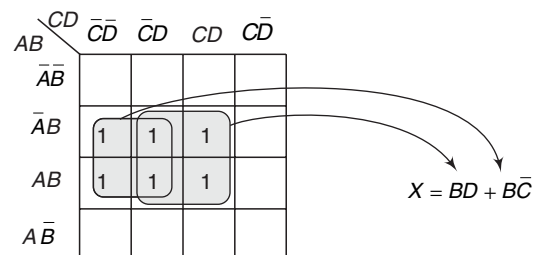


Fig. 6.19 Solution for Problem 6.15.

\overline{BCD} term in the original equation fills in two cells: $\overline{ABCD} + \overline{A\overline{B}CD}$. We could have encircled four cells and two cells, but that would not have given us the simplest final equation. By encircling four cells and then four cells, we are sure to get the simplest final equation. *Always encircle the largest number of cells possible, even if the cells have already been encircled in another group.*

6.16 Simplify the following equation using the Karnaugh mapping procedure.

$$X = \overline{A}\overline{B}\overline{C} + \overline{A}\overline{C}\overline{D} + \overline{A}\overline{B} + \overline{A}BC\overline{D} + \overline{A}\overline{B}C$$

Solution:

Notice in Figure 6.20 that a new technique called *wrap-around* is introduced. You have to think of the K-map as a *continuous cylinder in the horizontal direction*, like the label on a pickle can. This makes the *left row* of cells *adjacent* to the *right row* of cells. Also, in the *vertical direction*, a continuous cylinder like a pickle can lying on its side, makes the *top row* of cells *adjacent* to the *bottom row* of cells. In Figure, for example, the four top cells are adjacent to the four bottom cells, to combine as eight cells having the variable \overline{B} in common.

Yet another circle of four is formed by the *wrap-around adjacencies* of the lower left and lower right pairs combining to have \overline{AD} in common. The final equation becomes $X = \overline{B} + \overline{AD}$. Compare the simplified equation with the original equation that had five terms in it.

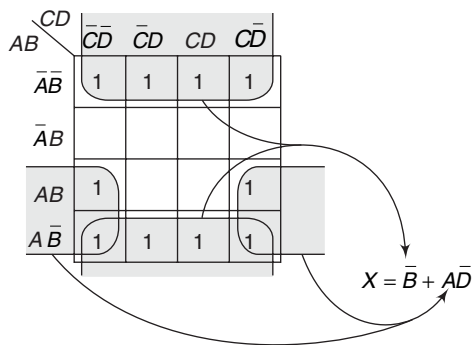


Fig. 6.20 Solution for Problem 6.16.

6.17 Simplify the following equation using K-mapping.

$$X = \overline{B}(CD + \overline{C}) + \overline{CD}(\overline{A} + \overline{B} + AB)$$

Solution:

Before filling in the K-map, an *SOP expression* must be formed.

$$\begin{aligned} X &= \overline{B}CD + \overline{B}\overline{C} + \overline{CD}(\overline{A} + \overline{B} + AB) \\ &= \overline{B}CD + \overline{B}\overline{C} + \overline{A}\overline{B}\overline{C} + \overline{B}\overline{C}D + \overline{A}BC\overline{D} + \overline{B}AB\overline{C} \end{aligned}$$

The group of four 1's can be encircled to form $\overline{A}\overline{B}$, as shown in Fig. 6.21. Another group of four can be

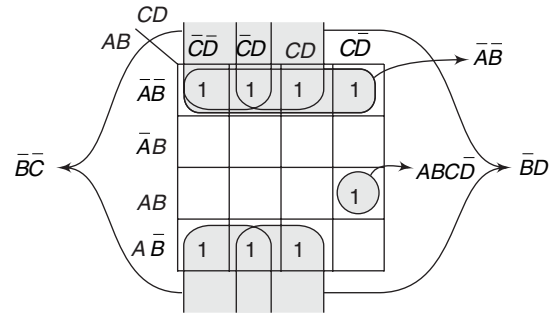


Fig. 6.21 Solution for Problem 6.17.

encircled using wraparound to form $\overline{B}\overline{C}$. That leaves two 1's that are not combined with any others. The unattached 1 in the bottom row can be combined within a group of four as shown, to form $\overline{B}D$.

The last 1 is not adjacent to any other, so it must be encircled by itself to form $\overline{A}BC\overline{D}$. The final simplified equation is

$$X = \overline{A}\overline{B} + \overline{B}\overline{C} + \overline{B}D + \overline{A}BC\overline{D}$$

6.18 Simplify the following equation by K-mapping.

$$X = \overline{A}\overline{D} + \overline{A}B\overline{D} + \overline{A}\overline{C}\overline{D} + \overline{A}CD$$

Solution:

First the group of eight cells can be encircled as shown in Fig. 6.22. \overline{A} is the only variable present in each cell within the circle, so the circle of eight simply reduces to \overline{A} . Notice that larger circles will reduce to four variables in the final equation. Also all four corners are adjacent to each other, because *the K-map can be wrapped around* in both vertical and horizontal directions. Encircling the four corners results in $\overline{B}D$. The final equation is

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

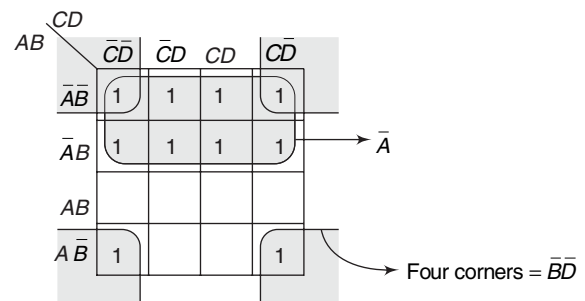


Fig. 6.22 Solution for Problem 6.18.

6.19 Simplify the following equation using the K-mapping procedure.

$$X = \overline{A}\overline{B}\overline{D} + \overline{A}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C} + \overline{A}BC\overline{D} + \overline{A}\overline{B}C\overline{D}$$

Solution:

End-around wrapping of the top corners produces $\overline{B}\overline{D}$. The group of fours forms $\overline{B}\overline{C}$. You may be tempted to

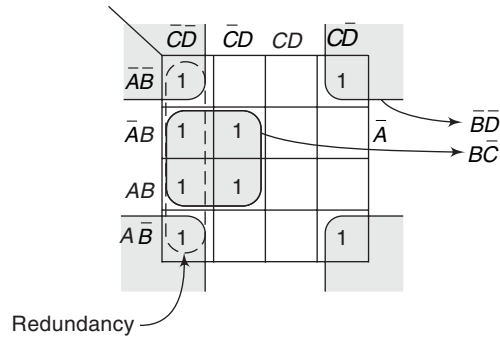


Fig. 6.23 Solution for Problem 6.19.

encircle CD group of fours as shown by the dotted line, but that would be *redundant* because each of these 1's is already contained within an existing circle. Therefore, the final equation is

$$X = \overline{B}\overline{D} + \overline{B}C$$

6.20 Map a four-variable function from the given truth table and simplify.

Solution:

D	C	B	A	$E = f(A, B \cdot C \cdot D)$
0	0	0	0	0
0	0	0	1	1
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
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	0
1	0	1	1	0
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Fig. 6.24 Solution for Problem 6.20.

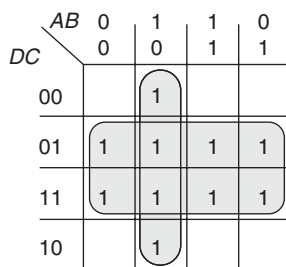


Fig. 6.25 Solution for Problem 6.20

Mapping the function and drawing loops in the usual way immediately obtains the function in its simplest form:

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

DON'T CARE CONDITIONS

There are applications where certain combinations of input variables never occur. As a result, we don't care what the function output is to be for these combinations of the variables because *they are guaranteed never to occur*. These don't care conditions can be used on a map to provide *further simplification* of the function. To distinguish the don't care conditions from 1's and 0's an X is used. *The X's may be assumed to be either 0 or 1, whichever gives the simplest expression*. In addition, an X need not be used at all if it does not contribute to covering a larger area. In each case, the choice depends only on the simplification that can be achieved.

6.21 From the table given draw the Karnaugh map with don't cares. Use don't cares to simplify the function. Implement the simplified function (Fig. 6.26).

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	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	1
1	0	1	0	x
1	0	1	1	x
1	1	0	0	x
1	1	0	1	x
1	1	1	0	x
1	1	1	1	x

Fig. 6.26

Solution:

The output is LOW for all input entries from 0000 to 1000, HIGH for input entry 1001, and an X for 1010 through 1111. *These don't cares are like trump cards because you can use them wherever you feel convenient* (Fig. 6.27a).

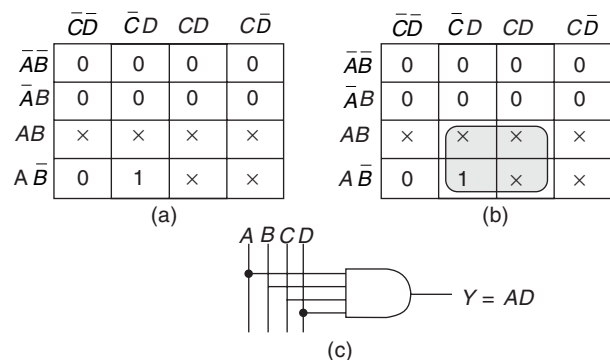


Fig. 6.27 Don't Care Condition.

Figure 6.27(b) shows the most efficient way to encircle the 1. Notice that the 1 is included in a quad, the largest group you can find if you visualise the X's as 1's. After the 1 has been encircled, all the X's outside the quad are visualised as 0's. In this way, the X's are all used to the best possible advantage. You are free to do this because don't cares correspond to input conditions that never appear.

The quad of Fig. 6.27(b) results in Boolean equation $Y = AD$. The logic circuit for this is an AND gate, Fig. 6.27(c).

6.22 Explain briefly the systematic procedure for using don't care conditions.

Solution:

Remember these ideas about don't care conditions:

1. Given the truth table, draw a Karnaugh map with 0's, 1's and don't cares.
2. Encircle the actual 1's on the Karnaugh map in the largest groups you can find by treating the don't-cares as 1's.
3. After the actual 1's have been included in groups, disregard the remaining don't cares by visualising them as 0's.

6.23 Suppose the table in Fig. 6.26 has a HIGH output for an input of 0000, LOW outputs for 0001 to 1001, and don't cares for 1010 to 1111. What is the simplest logic circuit with this truth table?

Solution:

The truth table has a 1 output only for the input condition 0000. The corresponding fundamental product is $\bar{A}\bar{B}\bar{C}\bar{D}$. Figure 6.28(a) shows the Karnaugh map with a 1 for the fundamental product, 0's for inputs 0001 to 1001, and X's for inputs 1010 to 1111. In this case, the don't cares are of no help. The best we can do is to encircle the isolated 1, while treating the don't-cares as 0's. So, the Boolean equation is

$$Y = \bar{A}\bar{B}\bar{C}\bar{D}$$

Figure 6.28(b) shows the logic circuit. The four-input AND gate produces a HIGH output only for the input condition $A = 0, B = 0, C = 0, \text{ and } D = 0$.

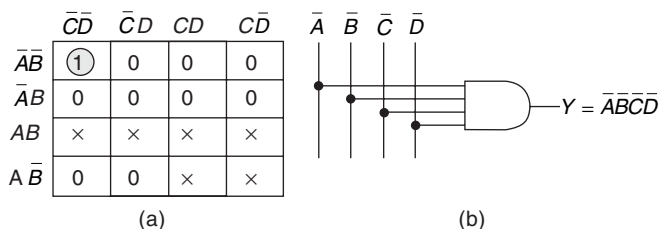


Fig. 6.28 Solution for Problem 6.23. Decoding 0000.

6.24 A truth table has LOW outputs for inputs of 0000 to 0110, a HIGH output for 0111, LOW outputs for 1000 to 1001, and don't-cares for

1010 to 1111. Show the simplest logic circuit for this truth table.

Solution:

Figure 6.29(a) is the Karnaugh map. The most efficient encircling is to group the 1's into a pair using the don't care as shown. Since this is the largest group possible, all remaining don't cares are treated as zeros. The equation for the pair is $Y = BCD$ and Fig. 6.29(b) is the logic circuit.

This three-input AND gate produces a HIGH output only for an input of $A = 0, B = 1, C = 1, \text{ and } D = 1$ because the input possibilities range only from 0000 to 1001.

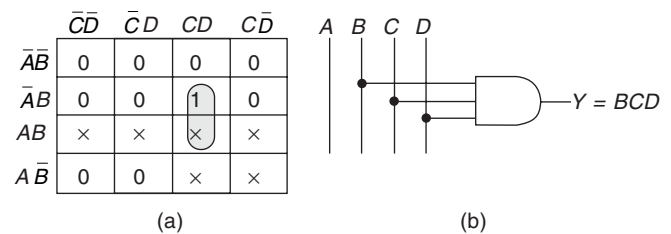


Fig. 6.29 Solution for Problem 6.24. Decoding 0111.

6.25 Simplify the Boolean function

$$F(W, X, Y, Z) = \sum m(1, 3, 7, 11, 15)$$

and the don't-care conditions

$$d(W, X, Y, Z) = \sum m(0, 2, 5)$$

Solution:

The minterms of F are the variable combinations that make the function equal to 1. The minterms of d are the don't-care combinations known never to occur. The minimisation is shown in Fig. 6.30. The minterms of F are marked by 1's, those of d are marked by X's and the remaining squares are filled with 0's. In (a), the 1's and X's are combined in any convenient manner so as to enclose the maximum number of adjacent cells. It is not necessary to enclose any or all of the X's, but only those useful for simplifying a term. One combination that gives a minimum function encloses one X and leaves two out. This results in a simplified sum-of-products function:

$$F = \bar{W}Z + YZ$$

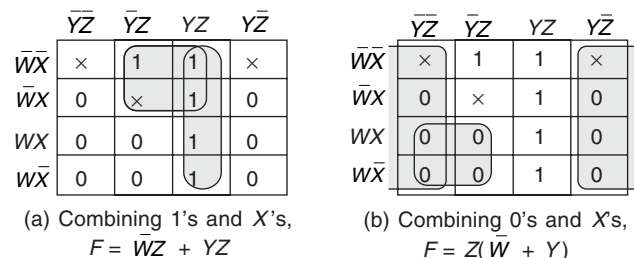


Fig. 6.30 Solution for Problem 6.25 with Don't-care Conditions.

In (b), the 0's Problem are combined with any X 's convenient to simplify the complement of the function. The best results are obtained if we enclose the two X 's as shown. The complement function is simplified to

$$F' = \bar{Z} + W\bar{Y}$$

Complementing again, we get a simplified *product-of-sums* function:

$$F = Z(\bar{W} + Y)$$

The two expressions obtained in Problem 6.25 give two functions which can be shown to be algebraically equal. This is not always the case when don't-care conditions are involved. As a matter of fact, *if an X is used as 1 when combining the 1's and again as a 0 when combining the 0's, the two resulting functions will not yield algebraically equal answers.*

FROM TRUTH TABLE TO KARNAUGH MAP

Sometimes even the unsimplified equation of a given function is not available, but a truth table is given. It might appear that it would be necessary to derive an equation from the truth table before inserting it in the Karnaugh map for simplification, but fortunately this is not the case.

It is fairly obvious that *since the truth table and Karnaugh map show all possible combinations of variables, a Karnaugh map can be considered as simply another way of writing a truth table.* Data can thus be transferred directly from a truth table to a Karnaugh map, and the equation is obtained straight from the map in its simplest *minterm* form.

For example, consider a four-variable function containing the term $A \cdot B \cdot \bar{C} \cdot D$. This would be entered on the Karnaugh map as a 1 in the appropriate cell. In the truth table this term is represented by the *input code* 1011 (i.e., a 1 in input column D , a 0 in input column C , a 1 in input column B , and a 1 in input column A), and if this term exists in the function a 1 appears in the output column. This means that *where a 1 appears in the output column of a truth table a 1 is entered in the Karnaugh map in the cell that corresponds to that input code.* This procedure can be greatly simplified if, instead of writing the actual terms at the ends of the rows and columns, the combination of logic states that they can assume are written instead (see Fig. 6.31)

		A B			
		0 0	0 1	1 1	1 0
D C	0 0	0000	0001	0011	0010
	0 1	0100	0101	0111	0110
	1 1	1100	1101	1111	1110
	1 0	1000	1001	1011	1010
		DCBA			

Fig. 6.31 Transferring Data from Truth Table to the Map.

Care must be taken when transferring data from the truth table to the map for several reasons. Firstly, the terms in a Karnaugh map change by only one bit from cell to cell, whereas the input codes of truth tables are commonly written down in a binary sequence, which can change by more than one bit from one code to the next. This means that the terms in the truth table do not appear in the same order as they do in the map.

Secondly, the maps so far considered have had the variables written down in alphabetical order (A, B, C, D, \dots) since that is how terms are generally written in equations. However, in truth tables codes are generally written in the form ($\dots D, C, B, A$) since that is the way binary numbers are normally written.

PRODUCT-OF-SUMS METHOD

With the *sum-of-products* method we start with a *truth table* that summarises the desired input-output conditions. The next step is to *convert* the truth table into an equivalent sum-of-product equation. The final step is to draw the AND-OR network or its equivalent NAND-NAND network.

The *product-of-sums* method is also similar. Given a truth table, you identify the fundamental sums needed for a logic design. Then by ANDing these sums, you get the product-of-sums equation corresponding to the truth table. But there are some differences between the two approaches. *With the sum-of-products method, the fundamental product produces an output 1 for the corresponding input condition. But with the product-of-sums method, the fundamental sum produces an output 0 for the corresponding input condition.*

6.26 Obtain the product-of-sums equation from the given truth table.

A	B	C	X
0	0	0	$0 \rightarrow A + B + C$
0	0	1	1
0	1	0	1
0	1	1	$0 \rightarrow A + \bar{B} + \bar{C}$
1	0	0	1
1	0	1	1
1	1	0	$0 \rightarrow \bar{A} + \bar{B} + C$
1	1	1	1

Fig. 6.32

Solution:

Locate each output 0 in the truth table and write down its fundamental sum. The first output 0 appears for $A = 0$, $B = 0$, and $C = 0$. The fundamental sum for these inputs is $A + B + C$, because *this produces an output 0 for the corresponding input conditions.*

$$X = A + B + C = 0 + 0 + 0 = 0$$

The second output 0 appears for the input condition of $A = 0$, $B = 1$, and $C = 1$. The fundamental sum for this is $A + \bar{B} + \bar{C}$. Notice that B and C are comple-

mented because this is the *only* way to get a logical sum of 0 for the given input conditions:

$$X = A + \bar{B} + \bar{C} = 0 + 1 + 1 = 0$$

Similarly, the third output 0 occurs for $A = 1$, $B = 1$, and $C = 0$, therefore, its fundamental sum is $\bar{A} + \bar{B} + C = 0$:

$$X = \bar{A} + \bar{B} + C = 1 + 1 + 0 = 0 + 0 + 0 = 0$$

Notice that each variable is *complemented* when the corresponding input variable is a 1; the variable is *uncomplemented* when the corresponding input variable is a 0. To get the product-of-sums equation, all you have to do is AND the fundamental sums:

$$X = (A + B + C)(A + \bar{B} + \bar{C})(\bar{A} + \bar{B} + C)$$

This is the *product-of-sums* equation for the given truth table.

SIMPLIFICATION

You can simplify with *Boolean algebra*. Alternatively, you may prefer simplification based on the *Karnaugh map*. There are several ways of using the Karnaugh map.

6.27 Draw the K-map for the truth table in Fig. 6.33. Obtain the sum-of-products equation. Show the corresponding NAND-NAND circuit. Draw a complemented map. Obtain the corresponding product-of-sums NOR-NOR circuit using DeMorgan's theorem.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	0
1	0	0	1	0
1	0	1	0	1
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. 6.33

Solution:

Draw the K-map in the usual way, Fig. 6.34(a). The encircled groups allow us to write a *sum-of-products* equation:

$$Y = \bar{A}\bar{B} + AB + AC$$

Figure 6.34(b) shows the corresponding NAND-NAND circuits.

To get a *product-of-sums* circuit, begin by complementing each 0 and 1 on the K-map, Fig. 6.35(a). The

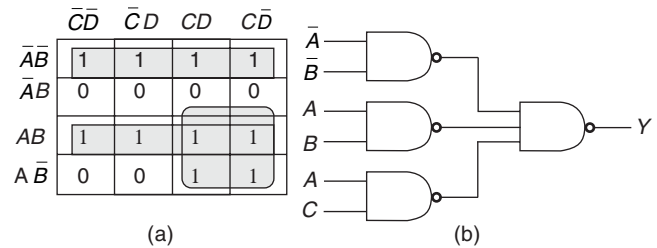


Fig. 6.34 (a) Karnaugh map. (b) Sum-of-products circuit for Y.

encircled 1's allow us to write the following sum-of-products equation from the *complemented* map:

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

Complementing the K-map is the same as complementing the output of the truth table, which means the sum-of-products equation for Fig. 6.35(a) is for \bar{Y} instead of Y, the complement of the desired output (Fig. 6.35(b)).

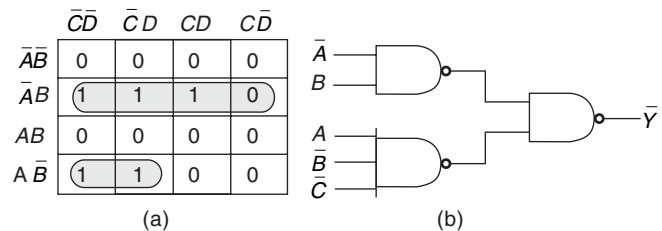


Fig. 6.35 (a) Complemented Map for Fig. 6.34(a). (b) Sum-of-products Circuit for \bar{Y} .

NAND gates can be replaced by bubbled OR gates; therefore, we can replace Fig. 6.35(b) by Fig. 6.36(a). A bus with each variable and its complement is usually available in a digital system. So instead of connecting A and B to a bubbled OR gate, we connect A and B to an OR gate, as shown in Fig. 6.36(b). In a similar manner A, B, and C are connected to an OR gate. In short Fig. 6.35(b) is equivalent to Fig. 6.36(a).

The next step is to *slide the bubbles* to the left from the output gate to the input gates. This changes the input OR gate to NOR gates. The final step is to use a NOR gate on the output to produce Y instead of \bar{Y} , as shown in Fig. 6.36(d).

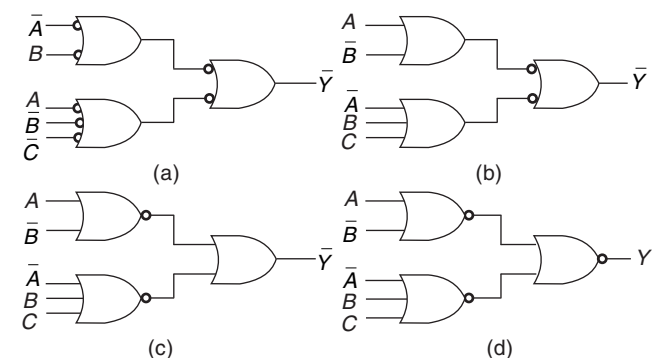


Fig. 6.36 Deriving the Product-of-sums Circuit.

DUALITY THEOREM

You don't have to go through every step in changing a complementary NAND-NAND circuit to an equivalent NOR-NOR circuit. Instead you can apply the duality theorem. Given a logic circuit, we can find its *dual* as follows:

1. Change each AND gate to an OR gate.
2. Change each OR gate to an AND gate.
3. Complement all input-output signals.

Compare the NOR-NOR circuit of Fig. 6.36(d) with the NAND-NAND circuit of Fig. 6.35(b).

6.28 Use Karnaugh maps to simplify the following equations:

(a) $D = AC + \bar{A}C + \bar{C}$

(b) $D = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + BC + \bar{A}\bar{B}C + \bar{A}BC$

Solution:

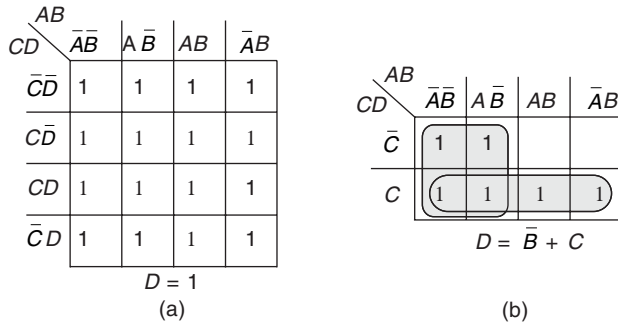


Fig. 6.37

6.29 Use Karnaugh map to simplify the following equations:

(a) $E = \bar{A}BC + \bar{B}CD + AC + \bar{A}\bar{B}C\bar{D}$

(b) $E = ABC + BCD + AC + BC$

Solution:

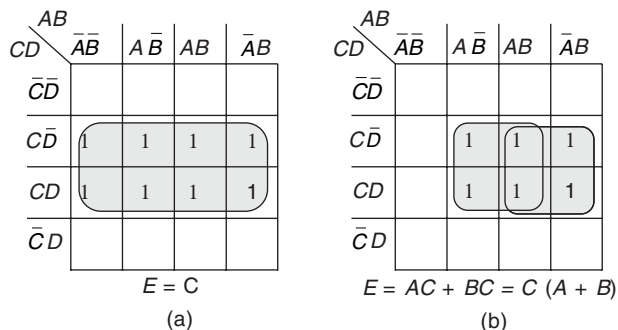


Fig. 6.38 (a) Solution for Problem 6.29(a).
(b) Solution for Problem 6.29(b).

6.30 Simplify the given expression by K-mapping:

$$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}\bar{D} + \bar{A}BC\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D}$$

Solution: A group of eight (*octet*) 1's can be factored as shown in Fig. 6.38, because the 1's in the outer columns are *adjacent*. A group of four (*quad*) 1's is formed by the *wraparound adjacency* of cells to pick up the remaining two 1's. The *minimum form* of the original equation is: $X = \bar{D} + \bar{B}C$

The reduced equation requires one two-input AND gate and one two-input OR gate for implementation.

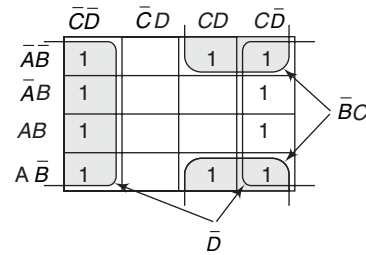


Fig. 6.39 Solution for Problem 6.30.

6.31 Design a circuit that can be built using an AOI and inverter that will output a HIGH (1) whenever the four-bit hexadecimal input is an odd number from 0 to 9.

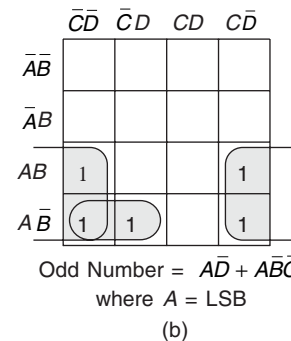
Solution:

Table 6.1 Hex Truth Table Used to Determine the Equation for Odd Numbers^a from 0 to 9

D	C	B	A	DEC
0	0	0	0	0
0	0	0	1	1 $\leftarrow \bar{A}\bar{B}\bar{C}\bar{D}$
0	0	1	0	2
0	0	1	1	3 $\leftarrow \bar{A}\bar{B}C\bar{D}$
0	1	0	0	4
0	1	0	1	5 $\leftarrow \bar{A}BC\bar{D}$
0	1	1	0	6
0	1	1	1	7 $\leftarrow \bar{A}BCD$
1	0	0	0	8
1	0	0	1	9 $\leftarrow \bar{A}\bar{B}\bar{C}D$

$$^a \text{Odd number} = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D} + \bar{A}BCD + \bar{A}\bar{B}\bar{C}D$$

(a)



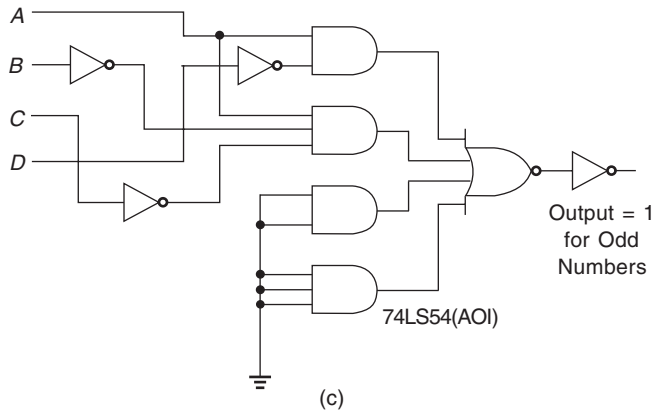


Fig. 6.40 (a) Hex Truth Table Used to Determine the Equation for Odd Numbers^a from 0 to 9. (b) Simplified Equation Using K-Map. (c) Implementation of the Odd-Number Decoding Using an AOI.

First build a *truth table* to identify which hex codes from 0 to 9 produce odd numbers. Use the variable *A* to represent 2^0 hex input, *B* for 2^1 , *C* for 2^2 , and *D* for 2^3 . Next, reduce this equation to its simplest form by using a *Karnaugh map*. Finally, using an AOI with inverters, the *circuit* can be constructed as shown.

6.32 A chemical plant needs a microprocessor-driven alarm system to warn of critical conditions in one of its chemical tanks. The tank has four HIGH/LOW(1/0) switches that monitor temperature (*T*), pressure (*P*), fluid level (*L*), and weight (*W*). Design a system that will notify the microprocessor to activate an alarm when any one of the following conditions arise:

1. High fluid level with high temperature and high pressure.
2. Low fluid level with high temperature and low weight.
3. Low fluid level with low temperature and high pressure.
4. Low fluid level with high temperature and low weight.

Solution:

First, write in Boolean equation form the conditions that will activate the alarm:

$$\text{Alarm} = LTP + \bar{L}TW + \bar{L}\bar{T}P + \bar{L}\bar{W}T$$

Next, factor the equation into its simplest form by using a Karnaugh map, as shown in Fig. 6.41. Finally, using AOI with inverters, the circuit can be constructed as shown. (See Fig. 6.41)

THE QUINE-McCLUSKY METHOD

The obvious disadvantage of the K-map is that it is essentially a *trial-and-error procedure*, which relies on the ability of human user to recognise certain patterns. For functions of six or more variables, it is difficult to

	$\bar{L}\bar{W}$	$\bar{L}W$	$L\bar{W}$	LW
$\bar{T}P$				
$\bar{T}\bar{P}$	1	1		
TP	1	1	1	1
$T\bar{P}$	1	1		1

Alarm = $TP + \bar{P}\bar{L} + T\bar{L}$

(a)

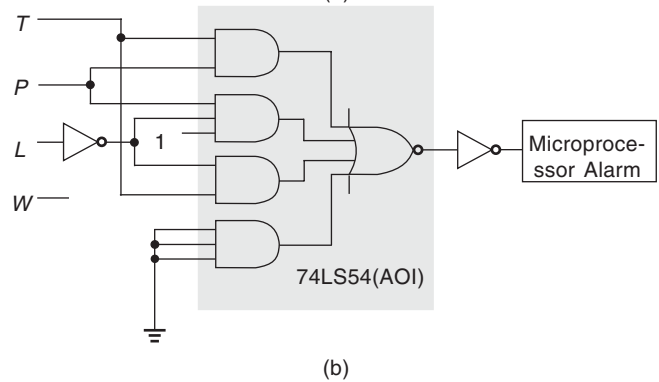


Fig. 6.41 (a) Simplified Equation Derived from a Karnaugh Map. (b) Implementation of the Chemical Tank Using an AOI.

be sure that the best selection has been made. The *tabulation method*, also known as the Quine-McClusky method overcomes this difficulty. It is a specific step-by-step procedure that is guaranteed to produce a standard-form expression for a function. It can be applied to problems with many variables and has the advantage of being *suitable for machine computation*. However, it is quite tedious for human use and is prone to mistakes.

The tabular method of simplification consists of two parts. The first is to find by an exhaustive search all the terms that are candidates for inclusion in the simplified function. These are called *prime implicants*. The second is to choose among the prime implicants those that give an expression with the least number of literals.

The flow chart for the tabulation method of minimisation is given in Fig. 6.42.

6.33 Find out the set of prime-implicants from the switching function given below, obtain the minimal expression using the Karnaugh map.

$$f(A, B, C, D) = \Sigma(0, 1, 2, 5, 7, 8, 9, 10, 13, 15)$$

Solution:

The location of minterms and a four-variable Karnaugh map are given in Figs. 6.43 (a) and (b), respectively.

In the *first step*, adjacent minterms are combined to form the following *sub-cubes of two cells*.

$$(0, 1), (0, 2), (0, 8), (1, 5), (1, 9), (2, 10), (5, 13), (5, 7), (7, 15), (8, 9), (8, 10), (9, 13), (13, 15)$$

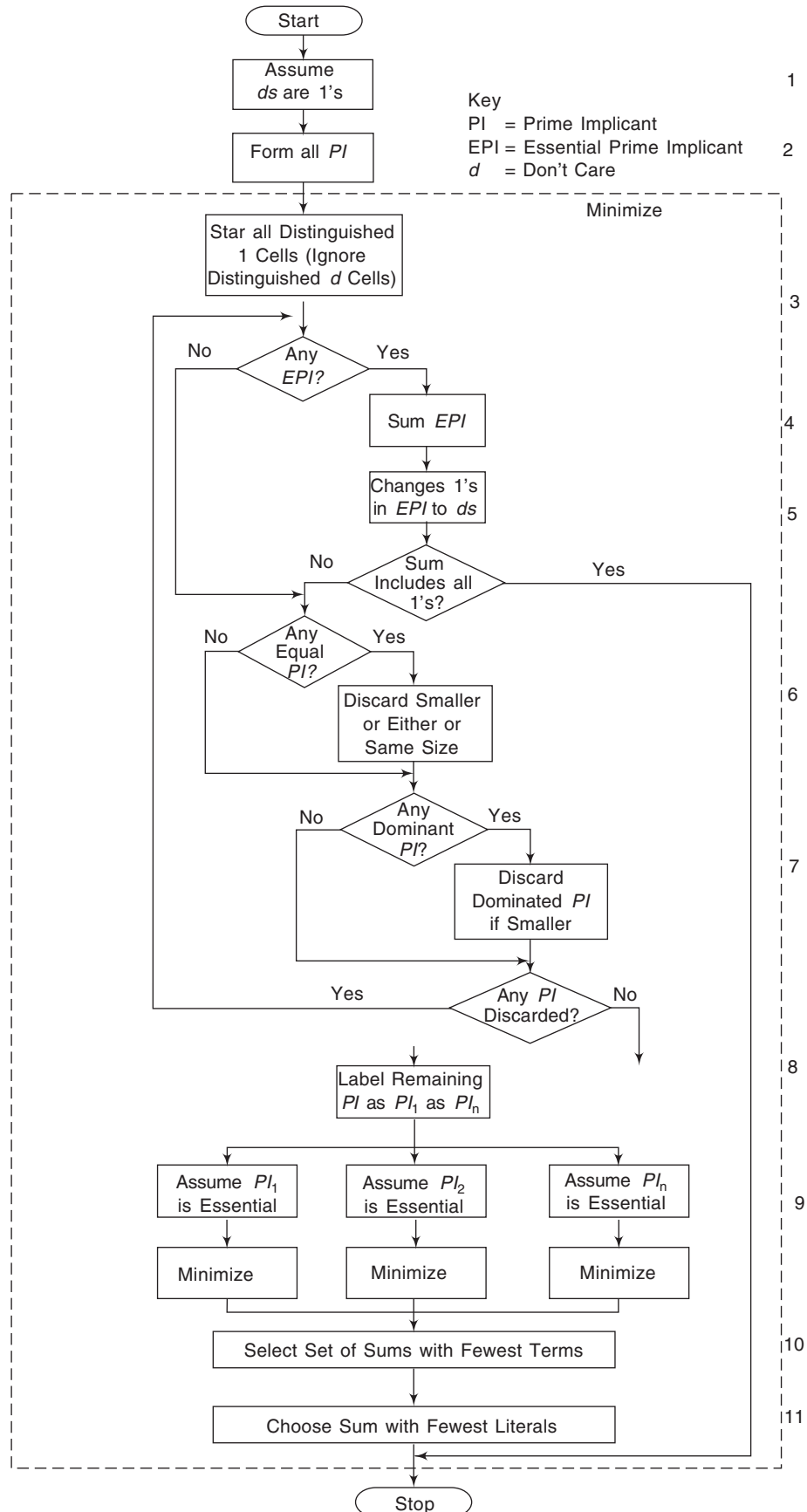


Fig. 6.42 Flow Chart for the Tabulation Method of Minimisation Procedure.

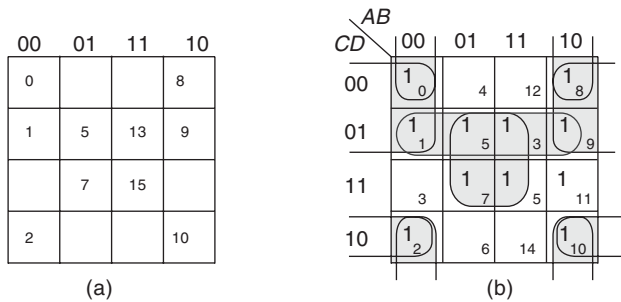


Fig. 6.43 (a) Location of Minterms (b) Karnaugh Map.

In the *second step*, the adjacent sub-cubes of two cells, obtained in the first step, are combined to form the following *sub-cubes of four cells*.

(0, 1, 8, 9), (0, 2, 8, 10), (1, 5, 9, 13), (5, 7, 13, 15)

The sub-cubes of two cells (pairs), which become included while forming the sub-cubes of four cells, *thus become redundant and therefore stand deleted from the set of pairs obtained in the first step*. On examining we find that all the sub-cubes of two cells (pairs) obtained in the first step are included in the sub-cubes of four cells. Thus, all the pairs obtained become redundant and stand deleted.

In the *third step* sub-cubes of four cells are combined, if possible, to form a bigger *sub-cube of eight cells*. This is not possible in this case. Thus, no bigger sub-cube can be formed. Therefore, the sub-cubes of four cells and the sub-cubes of two cells, *which are not deleted* while forming the bigger sub-cubes, all stand for the *prime implicant* of the given switching function. The sub-cubes which are not deleted are

(0, 1, 8, 9) which indicates $\bar{B}\bar{C}$

(0, 2, 8, 10) which indicates $\bar{B}\bar{D}$

(1, 5, 9, 13) which indicates $\bar{C}D$

and (5, 7, 13, 15) which indicates BD

Now $\bar{B}\bar{D}$ (0, 2, 8, 10) and BD (5, 7, 13, 15) are both *essential prime-implicants* and one out of $\bar{B}\bar{C}$ (0, 1, 8, 9) and $\bar{C}D$ (1, 5, 9, 13) at a time can be taken as the *third essential prime-implicant*. If we include $\bar{B}\bar{C}$ as the essential prime-implicant, then we must neglect $\bar{C}D$ (1, 5, 9, 13) because all 1, 5, 9, 13 are included in one prime implicant or the other. Therefore, the *minimal expression* is

$$f(A, B, C, D) = \bar{B}\bar{C} + \bar{B}\bar{D} + BD$$

AND now if take into account CD (1, 5, 9, 13) then we must neglect BC (0, 1, 8, 9) because then all 1, 5, 9, 13 are included in one prime-implicant or the other. Therefore, the second minimal expression is

$$f(A, B, C, D) = \bar{C}D + \bar{B}\bar{D} + BD$$

It is seen that the minimal expression obtained is *not unique*.

6.34 Find out the set of prime-implicants from the switching function given below. Find out the minimal expression using the Karnaugh map.

$$f(A, B, C, D) = \Sigma(0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$$

Solution:

Karnaugh map of the four-variable switching function is shown below

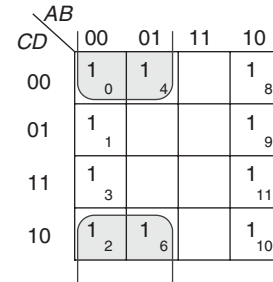


Fig. 6.44 Karnaugh Map for $\Sigma(0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$.

In the *first step*, the adjacent minterms are combined to form the following sub-cubes of *two '1' cells* to form the pairs.

(0, 1), (0, 4), (0, 2), (0, 8), (1, 3), (1, 9), (2, 3), (2, 6), (2, 10), (3, 11), (4, 6), (8, 9), (8, 10), (9, 11), (10, 11)

In the *second step*, the adjacent sub-cubes of two cells obtained above in the first step, are combined to form the following sub-cubes of four cells to yield *quads of '1' cells*.

(0, 1, 2, 3), (0, 1, 8, 9), (1, 3, 9, 11), (0, 2, 4, 6),

(0, 2, 8, 10), (2, 3, 10, 11), (8, 9, 10, 11)

The sub-cubes of two cells (pairs), which are included while forming the sub-cubes of four cells, *thus become redundant and therefore stand deleted from the set of pairs obtained in the first step*. On examination, we find that all the sub-cubes of two cells (pairs) obtained in the first step are included in the sub-cubes of four cells. Thus, all the pairs become redundant and stand deleted.

In the *third step*, the adjacent sub-cubes of four cells are combined, if possible, to form the sub-cube of eight cells (*octet*) given below.

(0, 1, 2, 3, 8, 9, 10, 11)

The quads, which become included while forming the octets thus become redundant and, therefore, stand deleted from the set of quads obtained in the second step. On examination, we find that all the quads except (0, 2, 4, 6) obtained in the second step are included in the octet. Thus, all quads except (0, 2, 4, 6) become *redundant* and stand deleted.

Since no bigger sub-cubes of 16 cells can be obtained, therefore, the sub-cubes of eight cells (octets), sub-cubes of four cells (quads), and the sub-cubes of two cells, which are not deleted while forming the bigger sub-cubes, all stand for the *prime-implicant* of the given switching function.

The sub-cubes which are not deleted are:

(0, 1, 2, 3, 8, 9, 10, 11) which indicates \bar{B} and (0, 2, 4, 6) which indicates $\bar{A}\bar{D}$.

The above two prime implicants are both *essential prime-implicants*, because prime-implicant \bar{B} contains some minterms or '1' cells viz. 1, 3, 8, 9, 10, 11 which are not included in any other prime-implicant. Similarly,

the prime-implicant AD contains some minterms or '1' cells viz. 4 and 6 which are not included in any other prime-implicant.

Thus, the unique minimal expression is given by the sum of these two essential prime implicants as

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

6.35 Simplify the minimise the following four-variable switching function using the Quine-McClusky tabulation method.

$$f(A, B, C, D) = \Sigma(0, 1, 2, 3, 4, 6, 8, 9, 10, 11)$$

Solution:

The problem is solved by constructing a table (Fig. 6.45) in the three columns, as shown below:

Index No.	Decimal No.	Column 1	First Column 2 Reduction	Second Column 3 Reduction
		A B C D	A B C D	A B C D
0	0	0 0 0 0 ✓	(0, 1) 0 0 0 - ✓	(0, 1, 2, 3) 0 0 - - } ✓
1	1	0 0 0 1 ✓	(0, 2) 0 0 - 0 ✓	(0, 2, 1, 3) 0 0 - - } ✓
	2	0 0 1 0 ✓	(0, 4) 0 - 0 0 ✓	(0, 2, 4, 6) 0 - - 0 } ✓
	4	0 1 0 0 ✓	(0, 8) - 0 0 0 ✓	(0, 4, 2, 6) 0 - - 0 } ✓
	8	1 0 0 0 ✓	(1, 3) 0 0 - 1 ✓	(0, 8, 1, 9) - 0 0 - ✓
2	3	0 0 1 1 ✓	(1, 9) - 0 0 1 ✓	(2, 3, 10, 11) - 0 0 - } ✓
	6	0 1 1 0 ✓	(2, 3) 0 0 1 - ✓	(2, 10, 3, 11) - 0 1 - } ✓
	9	1 0 0 1 ✓	(2, 6) 0 - 1 0 ✓	(8, 9, 10, 11) 1 0 - - } ✓
	10	1 0 1 0 ✓	(2, 10) - 0 1 0 ✓	(8, 10, 9, 11) 1 0 - - } ✓
3	11	1 0 1 1	(4, 6) 0 1 - 0 ✓	
			(8, 9) 1 0 0 - ✓	
			(8, 10) 1 0 - 0 ✓	
			(3, 11) - 0 1 1 ✓	
			(9, 11) 1 0 - 0 ✓	
			(10, 11) 1 0 1 - ✓	

Fig. 6.45

In *column 1*, the minterms are represented in binary form and written in ascending order according to the number of 1's contained in them. The minterms have been in four groups separated by horizontal lines. The '0' group and contains 0 number of 1's, first group one 1's, second group two 1's, third group three 1's, and the fourth group contains four 1's, respectively. Their equivalent decimal numbers are also written alongside. Any two minterms differing from each other in one variable only are combined together like (0, 1), (0, 2), (0, 4), etc. When any two minterms are same in all bit positions except one bit position, then a check (✓) is written towards the right side of both these minterms to indicate that these minterms have been used. The pair that results due to this combination is written along with its decimal equivalent in *column 2* of the table. A variable which is eliminated as a result of this comparison and combination is replaced by a dash (-) in its bit position. For example 0 (= 0000) combines with 1 (= 0001) to give the pair (0, 1) = (000-) and minterm

0 (= 0000) combines with 2 (= 0010) to form the pair (0, 2) = (00-0) and so on.

Now all the terms in column 2 have three variables. A '0' under the variable indicates that this variable is *complemented*. A '1' under the variable indicates that this variable is in *uncomplemented* form. A () under the variable indicates that the variable is not contained in the term. The comparison and combination procedure is repeated for the terms in column 2 to form the terms in column 3. The process is again repeated for the terms in column 3 to form the following terms:

- (0, 1, 2, 3, 8, 9, 10, 11) resulting by combination of (0, 1, 2, 3) and (8, 9, 10, 11) or by combination of (0, 8, 1, 9) and (2, 3, 10, 11).
- (0, 2, 4, 6)

The term (0, 1, 2, 3, 8, 9, 10, 11)

$$= A B C D \text{ stands for } \bar{B}.$$

$$- 0 - -$$

The term (0, 2, 4, 6) = $A B C D$ stands for $\bar{A}\bar{D}$.

$$0 - - 0$$

The sum of these two essential prime-implicants gives the minimal expression in SOP form as

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

6.36 Simplify and minimise the following four-variable switching function using the Quine-McClusky tabulation method.

$$\begin{aligned} &W\bar{X}\bar{Y}\bar{Z} + W\bar{X}\bar{Y}\bar{Z} + \bar{W}X\bar{Y}\bar{Z} + W\bar{X}YZ \\ &+ \bar{W}\bar{X}YZ + W\bar{X}YZ + \bar{W}\bar{X}YZ + \bar{W}\bar{X}\bar{Y}\bar{Z} \end{aligned}$$

Solution:

Step 1 Convert the expression to its *binary form*. The given expression becomes

$$1000 + 1100 + 0110 + 1101 + 0011 + 1001 + 0001 + 0000$$

Step 2 Arrange the binary representations in a table with a different number of 1's in each section:

0000
0001
1000
1100
0110
0011
1001
1101

Step 3 Perform matches between table entries Two terms match if they differ exactly in one position. A *new term* is then formed with a *dash* (–) substituted in the position where the two binary values differ. The new terms are then arranged according to Step 2. Of these new terms, a *match occurs* if two terms differ exactly in one position and have a dash (–) in the same position. Again a dash (–) is substituted in the position where the terms differ, and this procedure is repeated until no more matches are found. Terms involving matches are *checked*. All possible matches must be considered. Repeated term need not be copied. Only terms in different sections need be considered. The unchecked terms are called *prime implicants*, or PIs.

✓ 0000		
✓ 0001	✓ 000–	
✓ 1000	✓ –000	–00–
✓ 1100	00–1	1–0–
0110	✓ –001	
✓ 0011	✓ 1–00	
✓ 1001	✓ 100–	
✓ 1101	✓ 110–	
	✓ 1–01	

Fig. 6.46

Step 4 Construct a *prime implicant table* with all minterms listed at the bottom and all prime implicants from the match table along the side. Place X's at the intersections where a prime implicant matches the ca-

nonical term in each binary value; *treat dashes as don't cares*. If a prime implicant agrees with a minterm in each binary position, the PI *covers* the minterm.

0110								x	
00–1		x							x
–00–	x	x	x						x
1–0–			x	x				x	x
	000	001	100	110	011	001	100	110	

Fig. 6.47

Step 5 Choose a minimal set of prime implicants so that each minterm is covered. The logical sum of these is the minimal expression.

$$00-1 + -00- + 1-0- + 0110$$

or $\bar{W}\bar{X}\bar{Z} + \bar{X}\bar{Y} + W\bar{Y} + \bar{W}X\bar{Y}\bar{Z}$

6.37 Simplify and minimise the following four-variable switching function using the Quine-McClusky tabulation method.

$$\begin{aligned} f(W, X, Y, Z) = &\bar{W}\bar{X}\bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Y}\bar{Z} \\ &+ W\bar{X}YZ + W\bar{X}\bar{Y}\bar{Z} + \bar{W}\bar{X}\bar{Y}\bar{Z} \\ &+ \bar{W}\bar{X}YZ + W\bar{X}\bar{Y}\bar{Z} + W\bar{X}\bar{Y}\bar{Z} \end{aligned}$$

Solution:

✓ 0000	✓ 000–
✓ 0001	✓ 00–0
✓ 0010	✓ 00–1
✓ 0011	0–01
✓ 0101	✓ 001–
✓ 1100	–011
✓ 1011	–101
✓ 1101	110–
✓ 1110	11–0

(a)

0–01									
–011									
–101									
110–									
11–0									
00––	x	x	x	x					
	000	001	010	011	100	101	110	111	110

(b)

Fig. 6.48

1. Convert the expression to its *binary form*.

$$0000 + 0001 + 0010 + 1011 + 1100 + 0101 + 0011 + 1101 + 1110$$

2. Arrange the binary expression with a different number of 1's in each section.
3. Perform matches between table entries.

Two terms match if they differ exactly in one position.

Form new terms with a dash(–) substituted in the position where the two binary values differ. Arrange the new terms.

Perform matches between entries. A match occurs if two terms differ exactly in one position and have a dash (–) in the sample position.

A dash is substituted in the position where the terms differ.

This procedure is repeated until no more matches are found.

Terms involving matches are checked.

- Construct a prime implicant table. Treat dashes as don't cares.
- Choose a minimal set of prime implicants so that each minterm is covered.

Choose $00-- + -011 + 11-0 + -101$

$$\bar{W}\bar{X} + \bar{X}YZ + WX\bar{Z} + XY\bar{Z}$$

6.38 Simplify and minimise the four-variable product-of-sums switching function using the Quine-McClusky tabulation method.

$$f(a, b, c) = (a + \bar{b} + \bar{c}) + (\bar{a} + \bar{b} + c)(\bar{a} + b + c) \\ (a + b + \bar{c})(\bar{a} + \bar{b} + \bar{c})$$

Solution:

To minimise a product-of-sums, first complement the expression, then minimise, then complement the result.

$$\bar{f} = \bar{a}bc + ab\bar{c} + a\bar{b}\bar{c} + \bar{a}\bar{b}c + abc$$

or $011 + 110 + 100 + 001 + 111$

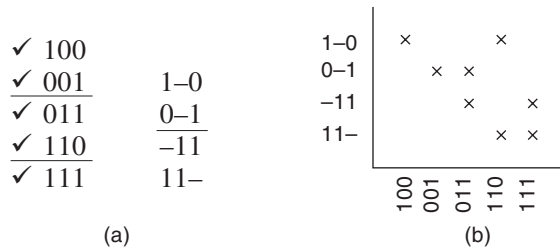


Fig. 6.49

There is a choice of two minimal expressions:

$$(1-0) + (0-1) + (-11) \quad \text{or} \quad (1-0) + (0-1) + (11-)$$

$$\bar{f} = a\bar{c} + \bar{a}c + bc \quad \text{or} \quad \bar{f} = a\bar{c} + \bar{a}c + ab$$

$$f = (\bar{a} + c)(a + \bar{c})(\bar{b} + \bar{c}) \quad \text{or} \quad f = (\bar{a} + c)(a + \bar{c})(\bar{a} + \bar{b})$$

6.39 Simplify and minimise the following four-variable switching function using the Karnaugh map and Quine-McClusky tabulation method of minimisation.

$$f(A, B, C, D) = \Sigma(0, 5, 7, 8, 9, 10, 11, 14, 15)$$

Solution:

The location of '1' cells on a four-variable Karnaugh map for the given switching function is given in Fig. 6.50.

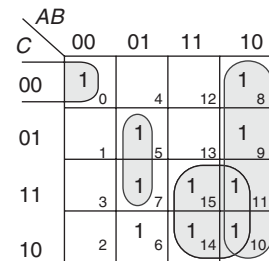


Fig. 6.50 Karnaugh Map for $f(A, B, C, D)$.

By combining the adjacent '1' cells of minterms, the following sub-cubes of two-cells (pairs) and four-cells (quads) are obtained which are not included entirely in any other bigger sub-cube.

$$(0, 8) \quad \text{which indicates} \quad \bar{B}\bar{C}\bar{D}$$

$$(5, 7) \quad \text{which indicates} \quad \bar{A}BD$$

$$(8, 9, 10, 11) \quad \text{which indicates} \quad A\bar{B}$$

$$(10, 11, 14, 15) \quad \text{which indicates} \quad AC$$

The sum of the above four product terms gives the required minimal SOP expression for the given switching function as

$$f(A, B, C, D) = \bar{B}\bar{C}\bar{D} + \bar{A}BD + A\bar{B} + AC$$

The same problem is solved again by minimising the given switching function represented by the sum of minterms as

$$f(A, B, C, D) = \Sigma(0, 5, 7, 8, 9, 10, 11, 14, 15)$$

by using the Quine-McClusky tabular method. The table is constructed, as shown, in three columns in Fig. 6.51.

Index	Column 1						Column 2						Column 3					
	Decimal Number	Binary Form A B C D					Decimal Number	First Reduction A B C D					Decimal Number	Second Reduction A B C D				
0	0	0	0	0	0	✓	0, 8	–	0	0	0	Ⓢ	8, 9, 10, 11 1 0 – – Ⓢ 10, 11, 14, 15 1 – 1 – Ⓣ					
1	8	1	0	0	0	✓	8, 9	1	0	0	–	✓						
							8,10	1	0	–	0	✓						
2	5	0	1	0	1	✓	5, 7	0	1	–	1	ⓧ						
	9	1	0	0	1	✓	9, 11	1	0	–	1	✓						
	10	1	0	1	0	✓	10, 11	1	0	1	–	✓						
							10, 14	1	–	1	0	✓						
3	7	0	1	1	1	✓	7, 15	–	1	1	1	Ⓢ						
	11	1	0	1	1	✓	11, 15	1	–	1	1	✓						
	14	1	1	1	0	✓	14, 15	1	1	1	–	✓						
4	15	1	1	1	1	✓												

Fig. 6.51

In column 1, the minterms are represented in *binary form* and are written in ascending order according to the index. The index indicates the *number of 1's* in the binary form of the minterm. The minterms have been written in four groups separated by horizontal lines. Their *equivalent decimal numbers* are also written alongside.

Any two minterms *differing from each other in one variable* are combined together. This results in formation of pairs (0, 8), (8, 9), (8, 10) etc. When any two minterms are *same in all bit positions but one*, then a check (✓) is written towards the right side of both these minterms to indicate that these minterms have been used to form the pairs. The pair which results due to this combination is also written along with its decimal equivalent in column 2 of the table. A variable which is '0' in one minterm and '1' in the other is eliminated as a result of this combination and replaced by a dash (–) in its bit position.

Now all the terms in column 2 have three variables and one dash (–). Any two pairs of columns *differing from each other in one variable only* are combined together to form quads of four minterms. For example, (8, 9) = (100–) is combined with pair (10, 11) = (101–), as they differ only in one position. These combinations result in formation of quads (8, 9, 10, 11) and (10, 11, 14, 15). A variable which is '0' in one pair and '1' in the other is eliminated as a result of this combination and is replaced by a dash (–) in its bit position. The quads which result due to this combination are written along with their decimal equivalent in column 3. Now any two pairs which are the same in all bit positions but one, are marked with a check (✓) toward their right side in column 2 to indicate that these pairs have been used to form the quads shown in column 3. For (8, 9), (10, 11) are combined (8, 10), (9, 11) are combined, (10, 14) (11, 15) are combined (10, 11), (14, 15) are also combined. The process of combining is repeated to combine quads to form an *octet*, if any. The quads or pairs which are not marked with check (✓), thus cannot be combined and have been marked *u, v, w, x, y*, respectively. Thus, the terms corresponding to *u, v, w, x, y* represent the *prime implicants*.

Construct the prime implicant table to find the *essential prime-implicants* of the function, out of the prime implicants *u, v, w, x, y* found earlier. Construct a table as shown in Fig. 6.52. Each column in this table has a decimal number at the top. This decimal number corresponds to one of the minterms in the canonical SOP form of the given switching function. The columns of the table are named by these decimal numbers in ascending order as shown. Each row of the table corresponds to one of the prime implicants represented by *u, v, w, x, y* on the left.

Prime Implicants	Minterms									
	0	5	7	8	9	10	11	14	15	
u						*	*	*	*	
v				*	*	*	*			
w			*						*	
x		*	*							
y	*			*						

Fig. 6.52

Put a cross under each decimal number, which is a term included in the prime implicant represented by that row. For example, for prime implicant *u* which includes 10, 11, 14, 15 we mark a cross under each of these column numbers in that row of prime implicant *u*. For prime implicant *v*, which includes 8, 9, 10, 11 mark a cross under each of these numbers in that row, and so on for prime implicants *w, x, y*, the process is repeated. Search all the columns which contain a single *x* i.e. only one cross under them and mark a circle around them. Also mark a circle around those prime implicants shown at the left of each of these rows in which the cross has been circled as shown around *u, v, x, and y*. These prime implicants, which are circled, represent the *essential prime implicants*.

Switching function $f(A, B, C, D)$

$$= U + V + X + Y$$

$$= (1-1-) + (10--) + (01-1) + (-000)$$

$$= AC + AB\bar{B} + \bar{A}BD + \bar{B}\bar{C}\bar{D}$$

which is the same as that found by using Karnaugh map.

SUMMARY

- The canonical expressions are usually not the most economical implementation of the logic functions.
- Simplification means finding an expression with fewer terms or literals.
- Minimising means finding an expression that is best by some minimisation criteria.
- The graphical method gives quick and positive results.
- The Karnaugh map is also called the Veitch diagram.
- The permitted sizes of loops are any powers of two with the provision that the loop must either be square or rectangular.
- Don't care conditions are guaranteed never to occur.
- Don't cares are like trump cards because you can use them wherever you feel convenient.

- A pair eliminates one variable that changes form.
- A quad eliminates two variables and their complements.
- An octet eliminates three variables and their complements.
- End-around and four-corner loops are permitted.
- A Karnaugh map can be considered as another way of writing a truth table.
- Every Boolean equation has a dual form obtained by changing OR to AND and AND to OR (0 to 1 and 1 to 0).
- A sum-of-products equation always results in an AND-OR circuit or its equivalent NAND-NAND circuits.
- Karnaugh map is a trial-and-error procedure.
- The tabulation method can be applied to problems with many variables.
- Each of the basic gates (AND, OR, NAND, NOR) can be used to enable/disable the passage of an input signals to its output.



Test your
understanding

REVIEW QUESTIONS

1. Identify each expression as either SOP or POS:
 - (a) $X = AB + CD + EF$
 - (b) $X = (A + B)(C + D)(E + F)$,
 - (c) $X = \bar{A}BC + A\bar{B}C + ABC + \bar{A}\bar{B}\bar{C}$
2. Determine the logic gate required to implement each of the following terms:
 - (a) ABC , (b) $A + B + C$, (c) \overline{ABC} (d) $\overline{A + B + C}$
3. Write the SOP expression for a circuit with four inputs and an output that is to be HIGH only when input A is LOW at the same time that exactly two other inputs are LOW.
4. If the SOP expression in 3 is implemented using all NAND gates, how many gates are required?
5. A logic designer needs an INVERTER and all that is available is an XOR gate from a 7486 chip. Does he need another chip?
6. Convert $\bar{A}B + C$ to minterms.
7. Convert $AB + \bar{C}D + A\bar{B}C$ to minterms.
8. Find the minterm designations for the following minterms:
 - (a) $\bar{A}\bar{B}C$, (b) $A\bar{B}C\bar{D}$, (c) $AB\bar{C}\bar{D}E$
9. How many gate inputs are required for the following equations?
 - (a) $W = A\bar{B}C + \bar{A}B + A\bar{C}D$,
 - (b) $X = R\bar{S} + \bar{R}TUV + RSTUV + \bar{T}UV$,
 - (c) $Y = (A + B + C)(A + \bar{B} + D)(\bar{A} + \bar{C}) + (A + \bar{B} + \bar{C} + D)$
10. Design an SOP circuit that will output a 1 any time the Gray codes 5 through 12, appear at the inputs and a 0 for all other cases.
11. Explain the terms: literal, product term, sum term, and normal term.
12. Draw the truth table for a three-input OR gate. Write the overall expression for the logic function as the sum of minterms.
13. Using NAND gates involves what steps?
14. Draw a pair, a quad, and an octet on K-map.
15. Describe redundancy with the help of an illustration.
16. How many inverters could be formed using a 7400 quad NAND IC?
17. Draw the Karnaugh map for AND and OR functions.
18. What is a Don't care condition?
19. Explain the term looping.
20. Briefly explain the simplification process in a K-map.
21. What is the difference in mapping a POS expression and an SOP expression?
22. What is the standard sum term for a 0 in cell 1011?
23. What is the standard product term for a 1 in cell 0010?
24. What is a prime implicant?

SUPPLEMENTARY PROBLEMS

Test your understanding

25. Suppose a truth table has a LOW output for the first three input conditions: 000, 001 and 010. If all other outputs are HIGH, what is the POS circuit?
26. Draw the SOP circuit for the Karnaugh-map. (See Fig. 6.53)
27. Draw the POS circuit for the Karnaugh-map in problem 26. (See Fig. 6.54)

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	0	0	0	0
$\bar{A}B$	0	0	0	1
AB	1	1	1	1
$A\bar{B}$	1	1	1	1

Fig. 6.53 $Y = A + B\bar{C}\bar{D}$.

	$\bar{C}\bar{D}$	$\bar{C}D$	CD	$C\bar{D}$
$\bar{A}\bar{B}$	1	1	1	1
$\bar{A}B$	1	1	1	0
AB	0	0	0	0
$A\bar{B}$	0	0	0	0

Fig. 6.54 $Y = \bar{A}\bar{B} + \bar{A}\bar{C} + \bar{A}\bar{D}$.

28. Draw the Karnaugh map for the given expression and make the groupings.

$$\bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}\bar{B}C + \bar{A}BC + ABC$$

29. Draw the Karnaugh map for the given expression and make the groupings.

$$\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD + \bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + \bar{A}BCD$$

30. Chart on a truth table and map $C = \bar{A}\bar{B} + A\bar{B}$
31. Chart and map the equation $X = ABC + A\bar{B}C + A\bar{B}\bar{C}$.
32. Chart and map $X = \Sigma m(1, 2, 3, 4, 9, 11, 12, 13, 15)$.
33. Simplify the Boolean expression by K-mapping.

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

34. Simplify the Boolean expression by K-mapping.

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD$$

35. Simplify the Boolean expression by K-mapping.

$$Y = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}\bar{B}CD$$

36. Find out the minimal expression for the switching function given below by K-map.
 $f(A, B, C, D) = \Sigma(1, 3, 6, 7, 9, 13, 14, 15)$.
37. Obtain the minimal expression for the function by using the K-map. (Refer Fig. 6.55)
38. Obtain the minimal POS and SOP expressions for the switching function given below using a four-variable K-map $f(A, B, C, D) = \Pi(3, 4, 6, 7, 11, 12, 13, 14, 15)$ (Refer Fig. 6.56).

$\backslash AB$	00	01	11	10
00	0	1 4	1 12	8
01	1 1	1 5	1 13	1 9
11	1 3	7	15	1 11
10	2	6	14	10

Fig. 6.55

$\backslash AB$	00	01	11	10
00	0	0 4	1 12	8
01	1	5	1 13	9
11	0 3	0 7	0 15	0 11
10	2	0 6	0 14	10

Fig. 6.56

39. Find the minimal expression for the Boolean function
 $f(W, X, Y, Z) = \Sigma(0, 1, 2, 3, 4, 7, 8, 11, 12, 14, 15)$
40. Minimise the given function by K-mapping: $f(A, B, C, D) = \Pi M(0, 1, 2, 3, 4, 7, 8, 11, 12, 14, 15)$

41. From the given truth table obtain the sum-of-products using K-map AND realise it using NAND gates (refer Fig. 6.57).

A	B	C	Y
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Fig. 6.57

42. Determine the simplified Boolean equation from the truth table given in Fig. 6.58.

A	B	C	D	Y
0	0	0	0	1
0	0	0	1	1
0	0	1	0	×
0	0	1	1	×
0	1	0	0	1
0	1	0	1	1
0	1	1	0	×
0	1	1	1	×
1	0	0	0	×
1	0	0	1	×
1	0	1	0	×
1	0	1	1	×
1	1	0	0	1
1	1	0	1	1
1	1	1	0	×
1	1	1	1	×

Fig. 6.58

Test your
understanding

OBJECTIVE TYPE QUESTIONS

Fill in the Blanks

43. Karnaugh mapping requires that you reduce the equation to a _____ format.
44. A Karnaugh map is similar to a _____.
45. Karnaugh mapping is a graphical method of applying the laws of _____ and absorption.
46. An n -variable requires _____ cells.
47. Adjacent cells in a Karnaugh must not differ by _____ one variable.
48. The variable that _____ between the cells is dropped.
49. The permitted sizes of loops are any power of _____.
50. The loops must be either rectangular or _____.
51. The _____ of two loops is allowed.
52. End-around and _____ loops are permitted.
53. A larger loop is defined by _____ variables.
54. Don't care conditions never _____ as long as the system is working properly.
55. You can't let a _____ condition either 1 OR 0.
56. A quad eliminates two variables and their _____.
57. An octet eliminates _____ variables and their complements.
58. With the POS method, the _____ sum produces an output 0 for the corresponding input condition.
59. Complementing the Karnaugh map is the same as complementing the _____ of the truth table.

60. NAND gates can be replaced by _____ OR gates.
 61. Karnaugh map depends on the ability of the human user to recognise certain _____.
 62. A logic function can be expressed in two _____ forms.
 63. Simplifying an expression means finding an expression with _____ terms or literals.
 64. Don't care conditions can be used on a map to provide further _____ of the function.

True/False Questions

65. The process of unreducing a Boolean expression is called expansion.
 66. The Karnaugh map method of simplification starts with converting the karnaugh map to a truth table.
 67. Every Boolean equation has a dual form obtained by changing AND to OR, OR to AND, 0 to 1, and 1 to 0.
 68. In a Karnaugh map you cannot eliminate the variable that changes form.
 69. You can eliminate the variable that appears in both complemented and uncomplemented form.
 70. A quad eliminates two variables and their complements.
 71. An octet eliminates four variables and their complements.
 72. System complexity and size are indirectly related to the complexity of the corresponding logic expressions and equations.
 73. Most common, for a large number of variables, is the K-map.
 74. Adjacent cells represent minterms which differ by at least one variable.
 75. The minterm represented by a cell is determined by the binary assignments of the variables for that cell.
 76. Many logic gates combine two or more of the basic logic functions.
 77. A single NAND gate performs an OR operation on its uncomplemented input terminals.
 78. A single NOR gate performs an AND operation on its complemented input terminals.
 79. The Quine-McClusky tabular method of minimisation is especially useful for functions with several variables but it cannot be programmed for a computer.
 80. Complex functions can be constructed by successively applying operators to functions.
 81. Precedence rules are not needed to specify unambiguously the order in which operators should be applied.
 82. The Quine-McClusky approach is algorithmic.

Multiple Choice Questions

83. Karnaugh map is used to
 (a) minimise the number of flip-flops in a digital circuits
 (b) to design gates
 (c) to minimise the number of gates in only a digital circuit
 (d) to minimise the number of gates and fan-in requirements of the gates in a digital circuit.
 84. The SOP form of logical expression is most suitable for designing logic circuits using only
 (a) XOR gates (b) NOR gates
 (c) NAND gates (d) AND gates
 85. The POS form of logical expression is most suitable for designing logic circuits using only
 (a) XOR gates (b) NOR gates
 (c) NAND gates (d) OR gates
 86. Total number of cells in the Karnaugh map of a switching function (A, B, C) consisting of only three variables is
 (a) 4 (b) 8 (c) 10 (d) 12
 87. The Karnaugh map method of minimisation of switching functions is very convenient and effective, if the number of variables in the switching function is
 (a) 8 (b) 4 (c) 5 (d) 6
 88. The minimised expression for the given K-map is (See Fig. 6.59)
 (a) $\overline{BCD} + \overline{BCD} + \overline{CD}$
 (b) $\overline{BD} + \overline{CD}$
 (c) $\overline{ABCD} + \overline{ABCD} + \overline{ABC} + \overline{ABC}$
 (d) $\overline{CD} + \overline{ABC} + \overline{ABCD} + \overline{ABC}$

		AB			
		00	01	11	10
CD	00	1			1
	01	1	1	1	1
	11				
	10	1			1

Fig. 6.59

89. The minimised expression for the given K-map is (See Fig. 6.60)

- (a) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}CD + \bar{A}\bar{B}C\bar{D}$ (b) $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}CD + \bar{A}BC$
 (c) $\bar{B}\bar{C}\bar{D} + CD + \bar{A}BC$ (d) $\bar{B}\bar{C}\bar{D} + CD + BC$

90. The minimised expression for the given K-map is (See Fig. 6.61)

- (a) $\bar{A} \cdot (A + B)$ (b) $\bar{A} + B$ (c) $\bar{A} \cdot B$ (d) $(\bar{A} + \bar{B})(\bar{A} + B)(A + B)$

AB \ CD	00	01	11	10
00		1	×	
01			×	
11	1	1	×	×
10	1		×	×

Fig. 6.60

AB \ CD	00	01	11	10
00	0		0	0
01	0		0	0
11	0		0	0
10	0		0	0

Fig. 6.61

91. The minimised expression for the given K-map is (See Fig. 6.62)

- (a) $(C + D)(\bar{C} + \bar{D})(A + \bar{B})$
 (b) $(\bar{B} + C + D)(A + \bar{B} + C)(A + B + \bar{C} + \bar{D})(\bar{A} + \bar{B} + \bar{C} + \bar{D})$
 (c) $(\bar{B} + C + D)(A + \bar{B} + C + D)(A + B + \bar{C} + \bar{D})(A + B + \bar{C} + \bar{D})$
 (d) $(\bar{A} + B + C)(\bar{A} + B + C + \bar{D})(A + B + \bar{C} + D)(\bar{A} + \bar{B} + \bar{C} + \bar{D})$

92. The number of cells in a six-variable K-map is

- (a) 4 (b) 16 (c) 32 (d) 64

93. In the given K-map, the values of P , Q , R , and S are, respectively (See Fig. 6.63).

- (a) 10, 11, 10, 11 (b) 10, 11, 11, 10 (c) 11, 10, 10, 11 (d) 11, 10, 11, 10

AB \ CD	00	01	11	10
00	×	0	0	×
01		0		×
11	0	×	0	×
10		×		×

Fig. 6.62

AB \ CD	00	01	P	Q
00				
01				
R				
S				

Fig. 6.63

94. In K-map simplification, a group of eight adjacent ones leads to a term with

- (a) one literal less than the total number of variables
 (b) two literals less than the total number of variables
 (c) three literals less than the total number of variables
 (d) four literals less than the total number of variables

ANSWERS

- (a) SOP, (b) POS, (c) SOP.
- (a) 3-input AND, (b) 3-input OR, (c) 3-input NAND, (d) 3-input NOR.
- $\bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{A}BC\bar{D}$.
- Eight.
- No. The available XOR gate can be used as an Inverter by connecting one of its inputs to a constant HIGH.

6. $\bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + ABC$
 7. $\bar{A}\bar{B}\bar{C}D + \bar{A}\bar{B}C\bar{D} + \bar{A}B\bar{C}D + \bar{A}BC\bar{D} + A\bar{B}\bar{C}D + A\bar{B}C\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$
 8. (a) 5, (b) 10, (c) 25.
 9. (a) 11, (b) 18, (c) 16.
 10. $B\bar{C} + BD + AC\bar{D}$

Term	Definition
Literal	Variable or its complement (A , \bar{A} , B , \bar{B} , etc.)
Product term	Series of literals related by AND, e.g., $A\bar{B}D$, $AC\bar{D}E$, etc.
Sum term	Series of literals related by OR, e.g., $\bar{A} + B + \bar{C} + D + \bar{E} + F$, etc.
Normal term	Product or sum term in which no variable appears more than once.

12. (1) A minterm is generated for each column in which a 1 appears in the truth table.
 (2) The minterm contains each input variable in turn; the input being non-inverted if it is a 1 in the truth table and inverted if it is a 0.
 13. Using NAND gates involves the following steps:
 1. Start with a minterm (sum-of-products) Boolean expression.
 2. Draw the AND-OR logic diagram using AND, OR, and NOT symbols.
 3. Substitute NAND symbols for each AND and OR symbol, keeping all connections the same.
 4. Substitute NAND symbols with all inputs tied together for each inverter.
 5. Test the logic containing all NAND gates to determine if it generates the proper truth table.

14.

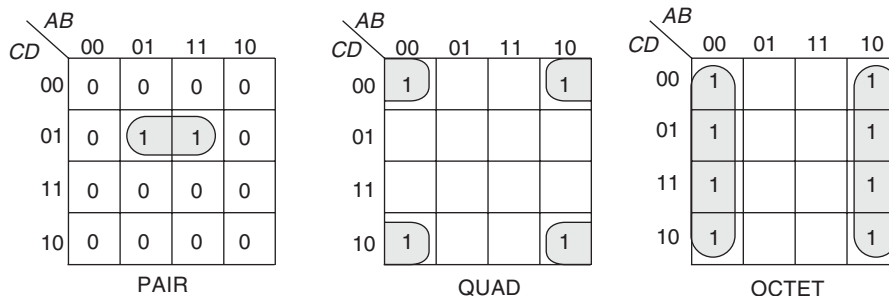


Fig. 6.64

15. While forming groups, overlapping of groups is allowed i.e. two redundant (not allowed) groups can have one or more 1's in common. At the same time redundancy is not allowed i.e. a group whose all 1's are overlapped by other groups. (See Fig. 6.65)
 16. 4
 17. (See Fig. 6.66)
 18. An input condition for which there is no specific required output condition i.e. we are free to make it 0 or 1.

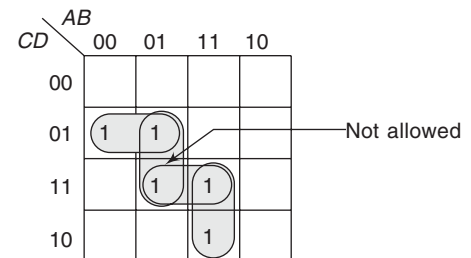


Fig. 6.65

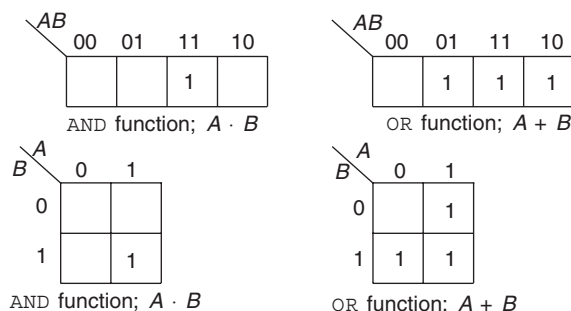
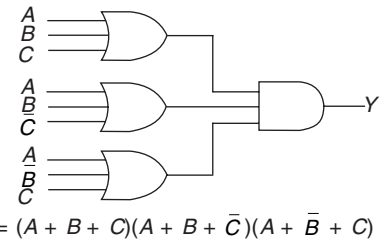
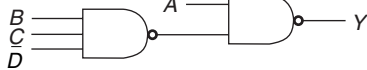
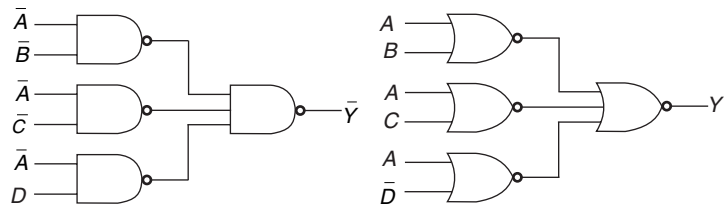
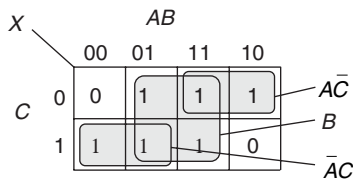
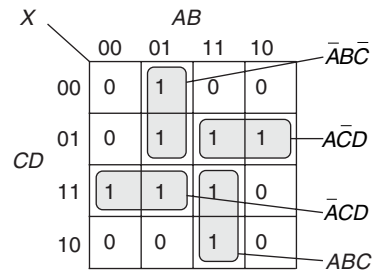


Fig. 6.66 Solution for Problem 17.

19. The expression for output X can be simplified by properly combining those squares in the K-map which contain 1's. The process for combining these 1's is called looping.
20. When a variable appears in both complemented and uncomplemented form within a loop, that variable is eliminated from the expression. Variables that are the same for all squares of the loop must appear in the final expression.
21. In mapping a POS expression, 0's are placed in cells whose value makes the standard sum term zero, and in mapping an SOP expression 1's are placed in cells having the same values as the product terms.
22. 0 in 1011 cell: $\bar{A} + B + \bar{C} + \bar{D}$.
23. 1 in the 0010 cell: $\bar{A}\bar{B}\bar{C}\bar{D}$
24. A tabular reduced term incapable of being reduced further.
25. (See Fig. 6.67)
26. (See Fig. 6.68)
27. (See Fig. 6.69)
28. (See Fig. 6.70)
29. (See Fig. 6.71)

**Fig. 6.67** Solution for Problem 25**Fig. 6.68** Solution for Problem 26.**Fig. 6.69** Solution for Problem 27.**Fig. 6.70** Solution for Problem 28.**Fig. 6.71** Solution for Problem 29.

30.

INPUTS			OUTPUT
A	B	C	
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	0

$\bar{A}\bar{B}$ is m_0 .
 AB is m_2 .

A	B		
	0	1	
0	1	0	1
1	1	0	3

(a) Charting

(b) Mapping

Fig. 6.72 Solution for Problem 30.

31.

INPUTS			OUTPUT
A	B	C	X
0	0	0	1
1	0	1	1
1	0	0	1

ABC is m_7
 ABC is m_5
 $A\bar{B}\bar{C}$ is m_4

BC	A			
	00	01	11	10
0	0	0	0	0
1	1	1	1	0

(a) Charting

(b) Mapping

Fig. 6.73 Solution for Problem 31.

32.

INPUTS				OUTPUT
A	B	C	D	E
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	1
0	1	1	0	1
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	0
1	0	1	1	0
1	1	0	0	0
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

(a) Charting

CD \ AB	CD			
	00	01	11	10
00	0 0	1 1	1 3	1 2
01	1 4	0 5	0 7	0 6
11	0 8	1 9	1 11	0 10
10	1 12	1 13	1 15	0 14

(b) Mapping

Fig. 6.74 Solution for Problem 32.

33. The Karnaugh map is considered to be wrapped in a cylinder, with the left side adjacent to the right side. Also notice the elimination of the A and \bar{A} and C and \bar{C} terms.

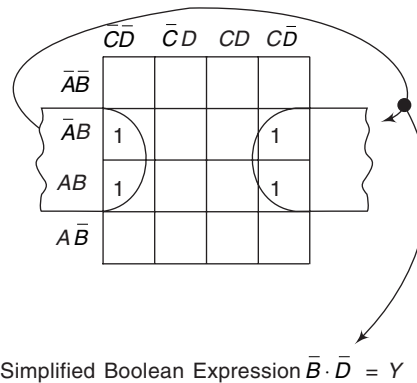


Fig. 6.75 Solution for Problem 33.

34. The K-map is considered to be a horizontal cylinder. In this way the four 1's can be looped. (See Fig. 6.76)
 35. The K-map is considered to be a ball. In this way, the 1's at the four corners can be enclosed in a single loop. (See Fig. 6.77).

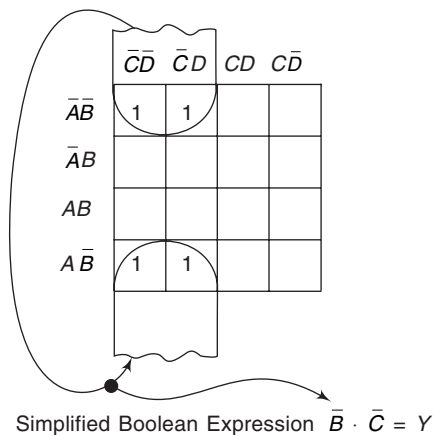


Fig. 6.76 Solution for Problem 34.

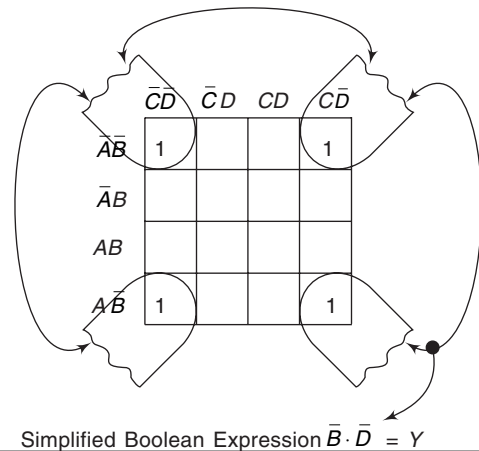


Fig. 6.77 Solution for Problem 35.

36. $f(A, B, C, D) = \bar{A}\bar{B}D + A\bar{C}D + BC$

C \ AB	00	01	11	10
	0	4	12	8
00	0	4	12	8
01	1 ₁	5	1 ₁₃	9
11	1 ₃	1 ₇	1 ₁₅	1 ₁₁
10	2	1 ₆	1 ₁₄	1 ₁₀

Fig. 6.78 Solution for Problem 36.

37. $f(A, B, C, D) = B\bar{C} + \bar{B}D$

38. POS, $f(A, B, C, D) = (\bar{A} + \bar{B})(\bar{B} + D)(\bar{C} + D)$; SOP, $f(A, B, C, D) = \bar{B}\bar{C} + \bar{B}D + \bar{A}\bar{C}D$

39. $f(W, X, Y, Z) = \bar{W}\bar{X} + \bar{X}\bar{Z} + \bar{W}YZ + W\bar{X}\bar{Y} + WXYZ$ (See Fig. 6.79)

40. $f(A, B, C, D) = (A + B)(C + D)(\bar{C} + \bar{D})(\bar{A} + \bar{B} + \bar{C})$ (See Fig. 6.80)

YZ \ WX	00	01	11	10
	0	4	12	8
00	1 ₀	1 ₄	1 ₁₂	1 ₈
01	1 ₁	5	1 ₁₃	9
11	1 ₃	7	1 ₁₅	1 ₁₁
10	1 ₂	6	1 ₁₄	1 ₁₀

Fig. 6.79 Solution for Problem 39.

C \ AB	00	01	11	10
	0	4	12	8
00	0	0	0	0
01	0	1	5	9
11	0	0	0	0
10	0	2	6	10

Fig. 6.80 Solution for Problem 40.

41. SOP expression $Y = AB + BC + AC$

$$\bar{Y} = \overline{AB + BC + AC}$$

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

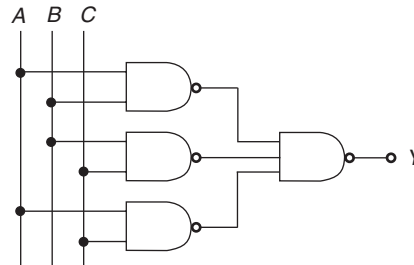
$$Y = \overline{\bar{A} \cdot \bar{B}\bar{C} \cdot \bar{A}\bar{C}}$$

42. Minimised expression

$$Y = \bar{A}\bar{B} + BD + AB + \bar{C}\bar{D}$$

CD \ AB	00	01	11	10
	0	4	12	8
00	1	1	1	x
01	1	1	1	x
11	x	x	x	x
10	x	x	x	x

(a)



(b) NAND-NAND Realisation

Fig. 6.82 Solution for Problem 42.

C \ AB	00	01	11	10
	0	4	12	8
0	0	0	1	0
1	0	1	1	1

Fig. 6.81 Solution for Problem 41.

43. SOP

47. More than

51. Intersection

55. Don't care

44. Truth table

48. Changes

52. Four corner

56. Complements

45. Complementation

49. Two

53. Fewer

57. Three

46. 2ⁿ

50. Square

54. Occur

58. Fundamental

- | | | | |
|-------------|--------------------|--------------|---------------|
| 59. Output | 60. Bubbled | 61. Patterns | 62. Canonical |
| 63. Minimum | 64. Simplification | 65. True | 66. False |
| 67. True | 68. False | 69. True | 70. True |
| 71. False | 72. False | 73. False | 74. False |
| 75. True | 76. True | 77. False | 78. True |
| 79. False | 80. True | 81. False | 82. True |
| 83. (d) | 84. (c) | 85. (b) | 86. (b) |
| 87. (b) | 88. (b) | 89. (d) | 90. (c) |
| 91. (a) | 92. (d) | 93. (d) | 94. (c) |