KAMARAJ COLLEGE

SELF FINANCING COURSES
(Reaccredited with "A+" Grade by NAAC)
(Affiliated to Manonmaniam Sundaranar University, Tirunelveli.)
THOOTHUKUDI – 628003.

STUDY MATERIAL FOR BCA

DIGITAL DESIGN

SEMESTER – I

Academic Year 2022-2023

Prepared by

COMPUTER SCIENCE DEPARTMENT

UNIT-1

BINARY NUMBERS

A decimal number such as 7,392 represents a quantity equal to 7 thousand, plus 3 hundred, plus 9 tens, plus 2 units. The thousands, hundreds, etc., are powers of 10 implied by the position of the coefficients (symbols) in the number. To be more exact, 7,392 is a shorthand notation for what should be written as

$7 * 10^3 + 3 * 10^2 + 9 * 10^1 + 2 * 10^0$

For example,

$$1 * 2^4 + 1 * 2^3 + 0 * 2^2 + 1 * 2^1 + 0 * 2^0 + 1 * 2^{-1} + 1 * 2^{-2} = 26.75$$

$$(127.4)_8 = 1 * 8^2 + 2 * 8^1 + 7 * 8^0 + 4 * 8^{-1} = (87.5)_{10}$$

An example of a hexadecimal number is

$$(B65F)_{16} = 11 * 16^3 + 6 * 16^2 + 5 * 16^1 + 15 * 16^0 = (46,687)_{10}$$

. For example,

$(110101)_2 = 32 + 16 + 4 + 1 = (53)_{10}$

Table 1.1

Powers of Two

n	2^n	*n*	2^n	*n*	2^n
0	1	8	256	16	65,536
1	2	9	512	17	131,072
2	4	10	1,024 (1K)	18	262,144
3	8	11	2,048	19	524,288
4	16	12	4,096 (4K)	20	1,048,576 (1M)
5	32	13	8,192	21	2,097,152
6	64	14	16,384	22	4,194,304
7	128	15	32,768	23	8,388,608

NUMBER-BASE CONVERSIONS

Representations of a number in a different radix are said to be equivalent if they have the same decimal representation.

EG:

Convert decimal 41 to binary:

	Integer Quotient	Remainder		Coefficient
41>2 =	20	+	$\frac{1}{2}$	$a_0 = 1$
20>2 =	10	+	0	$a_1 = 0$
10>2 =	5	+	0	$a_2 = 0$
5>2 =	2	+	$\frac{1}{2}$	$a_3 = 1$
2>2 =	1	+	0	$a_4 = 0$
1>2 =	0	+	$\frac{1}{2}$	$a_5 = 1$

Therefore, the answer is $(41)_{10} = (101001)_2$

OCTAL AND HEXADECIMAL NUMBERS

The conversion from and to binary, octal, and hexadecimal plays an important role in digital computers. each octal digit corresponds to three binary digits and each hexadecimal digit corresponds to four binary digits. The first 16 numbers in the decimal, binary, octal, and hexadecimal number systems are listed in Table 1.2.

$(10\ 110\ 001\ 101\ 011\ \#\ 111\ 100\ 000\ 110)_2 = (26153.7406)_8$
2 6 1 5 3 7 4 0 6

Table 1.2
Numbers with Different Bases

Decimal (base 10)	Binary (base 2)	Octal (base 8)	Hexadecimal (base 16)
00	0000	00	0
01	0001	01	1
02	0010	02	2
03	0011	03	3
04	0100	04	4
05	0101	05	5
06	0110	06	6
07	0111	07	7
08	1000	10	8
09	1001	11	9
10	1010	12	A
11	1011	13	B
12	1100	14	C
13	1101	15	D
14	1110	16	E
15	1111	17	F

E.g.:

$(673.124)_8 = (110$	111	011	#001	010 100)$_2$
6	7	3	1	2 4
$(306.D)_{16} = (0011$	0000	0110		1101)$_2$
3	0	6		D

COMPLEMENTS

Complements are used in digital computers to simplify the subtraction operation and for logical manipulation. Simplifying operations leads to simpler, less expensive circuits to implement the operations. There are two types of complements for each base-*r* system: the radix complements and the diminished radix complement. The first is referred to as the *r*'s complement and the second as the (*r* - 1)›s complement. When the value of the base *r* is substituted in the name, the two types are referred to as the 2's complement and 1's complement for binary numbers and the 10's complement and 9's complement for decimal numbers.

Diminished Radix Complement

Given a number *N* in base *r* having *n* digits, the (*r* - 1)›s complement of *N*, i.e., its diminished radix complement, is defined as $(r^n - 1) - N$. For decimal numbers, $r = 10$ and $r - 1 = 9$, Here are some numerical examples:

The 9›s complement of 546700 is 999999 - 546700 = 453299.
The 9›s complement of 012398 is 999999 - 012398 = 987601.

1's complement of a binary number is formed by changing 1's to 0's and 0's to 1's. The following are some numerical examples:

The 1's complement of 1011000 is 0100111. The 1's complement of 0101101 is 1010010.

Radix Complement

The *r*'s complement of an *n*-digit number *N* in base *r* is defined as $r^n - N$ for $N \neq 0$ and as 0 for $N = 0$. Comparing with the (*r* – 1)›s complement, we note that the r's complement is obtained by adding 1 to the (*r* – 1)›s complement, since $r^n - N = [(r^n - 1) - N] + 1$.

the 10's complement of 012398 is 987602
and
the 10's complement of 246700 is 753300

Subtraction with Complements

The subtraction of two *n*-digit unsigned numbers *M* – *N* in base *r* can be done as follows:

1. Add the minuend *M* to the *r*'s complement of the subtrahend *N*. Mathematically, $M + (r^n - N) = M - N + r^n$.
2. If $M \geq N$, the sum will produce an end carry r^n, which can be discarded; what is left is the result *M* – *N*.
3. If $M < N$, the sum does not produce an end carry and is equal to $r^n - (N - M)$, which is the *r*'s complement of (*N* - *M*). To obtain the answer in a familiar form, take the *r*'s complement of the sum and place a negative sign in front.

The following examples illustrate the procedure:

Using 10's complement, subtract 72532 - 3250.

M =	72532
10›s complement of	+
N =	96750
Sum =	169282
Discard end carry	-
10^5 =	100000
Answer =	69282

BINARY STORAGE AND REGISTERS

A *binary cell* is a device that possesses two stable states and is capable of storing one bit (0 or 1) of information. The input to the cell receives excitation signals that set it to one of the two states

Registers

A *register* is a group of binary cells. A register with *n* cells can store any discrete quantity of information that contains *n* bits. The state of a register is an *n*-tuple of 1's and 0's, with each bit designating the state of one cell in the register.

A register with 16 cells can be in one of 2^{16} possible states. If one assumes that the content of the register represents a binary integer, then the register can store any binary number from 0 to $2^{16} - 1$

Register Transfer

In digital systems, a *register transfer* operation is a basic operation that consists of a transfer of binary information from one set of registers into another set of registers. The transfer may b45e direct, from one register to another, or may pass through data-processing circuits to perform an operation.

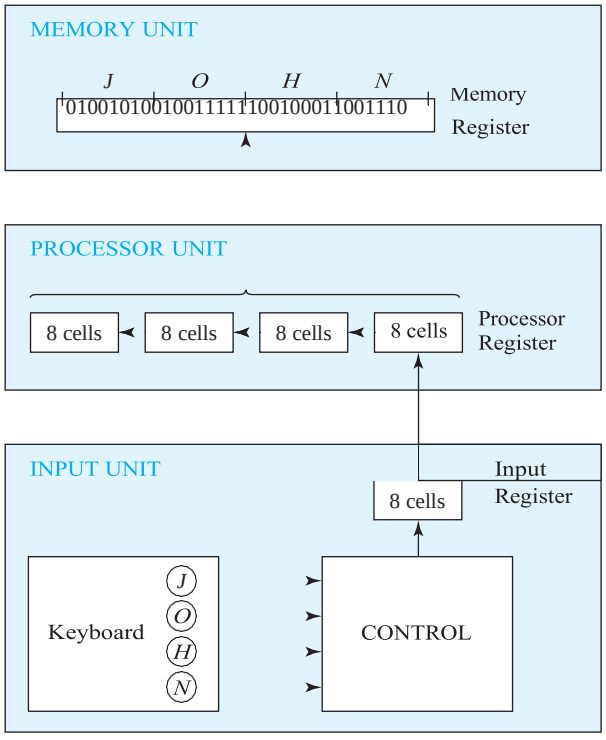


FIGURE 1.1
Transfer of information among register

AXIOMATIC DEFINITION OF BOOLEAN ALGEBRA

In 1854, George Boole developed an algebraic system now called *Boolean algebra*. In 1938, Claude E. Shannon introduced a two-valued Boolean algebra called *switching algebra* that represented the properties of bistable electrical switching circuits.

Comparing Boolean algebra with arithmetic and ordinary algebra we note the following differences:

1. Huntington postulates do not include the associative law. However, this law holds for Boolean algebra and can be derived from the other postulates.
2. The distributive law of + over # (i.e., $x + (y \# z) = (x + y) \# (x + z)$) is valid for Boolean algebra, but not for ordinary algebra.
3. Boolean algebra does not have additive or multiplicative inverses; therefore, there are no subtraction or division operations.
4. Postulate 5 defines an operator called the *complement* that is not available in ordinary algebra.
5. Ordinary algebra deals with the real numbers, which constitute an infinite set of elements. Boolean algebra deals with the as yet undefined set of elements, *B*, but in the two-valued Boolean algebra defined next, *B* is defined as a set with only two elements, 0 and 1.

Two-Valued Boolean Algebra

A two-valued Boolean algebra is defined on a set of two elements,

x	y	$x \# y$
0	0	0
0	1	0
1	0	0
1	1	1

x	y	x + y
0	0	0
0	1	1
1	0	1
1	1	1

x	x'
0	1
1	0

BASIC THEOREMS AND PROPERTIES OF BOOLEAN ALGEBRA

Duality

One part may be obtained from the other if the binary operators and the identity elements are interchanged. If the *dual* of an algebraic expression is desired, we simply interchange OR and AND operators and replace 1's by 0's and 0's by 1's.

Basic Theorems

Table 2.1 lists six theorems of Boolean algebra and four of its postulates. The notation is simplified by omitting the binary operator whenever doing so does not lead to confusion.

Table 2.1
Postulates and Theorems of Boolean Algebra

Postulate 2	(a)	$x + 0 = X$	(b) $x \# 1 = x$
Postulate 5	(a)	$x + x' = 1$	(b) $x \# x' = 0$
Theorem 1	(a)	$x + x = X$	(b) $\# x = x$
Theorem 2	(a)	$x + 1 = 1$	(b) $x \# 0 = 0$
Theorem 3, involution		$(x')' = X$	
Postulate 3, commutative	(a)	$x + y = y + x$	(b) $xy = yx$
Theorem 4, associative	(a)	$x + (y + z) = (x + y) + z$	(b) $x(yz) = (xy)z$
Postulate 4, distributive	(a)	$x(y + z) = xy + xz$	(b) $x + yz = (x + y)(x + z)$
Theorem 5, DeMorgan	(a)	$(x + y)' = x'y'$	(b) $(xy)' = x' + y'$
Theorem 6, absorption	(a)	$x + xy = X$	(b) $x(x + y) = x$

Operator Precedence

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR. The next operation that holds precedence is the complement.

UNIT:2

CANONICAL AND STANDARD FORMS

Minterms and Maxterms

A binary variable may appear either in its normal form (x) or in its complement form (x'). Now consider two binary variables x and y combined with an AND operation. Since each variable may appear in either form, there are four possible combinations: $x'y'$, $x'y$, xy', and xy. Each of these four AND terms is called a *minterm*, or a *standard product*.

A Boolean function can be expressed algebraically from a given truth table by forming a minterm for each combination of the variables that produces a 1 in the function and then taking the OR of all those terms.

For example,

$f_1 = x'y'z + xy'z' + xyz = m_1 + m_4 + m_7$

Table 2.3

Minterms and Maxterms for Three Binary Variables

			Minterms		Maxterms	
x	y	z	Term	Designation	Term	Designation
0	0	0	$x'y'z'$	m_0	$x + y + z$	M_0
0	0	1	$x'y'z$	m_1	$x + y + z'$	M_1
0	1	0	$x'yz'$	m_2	$x + y' + z$	M_2
0	1	1	$x'yz$	m_3	$x + y' + z'$	M_3
1	0	0	$xy'z'$	m_4	$x' + y + z$	M_4
1	0	1	$xy'z$	m_5	$x' + y + z'$	M_5
1	1	0	xyz'	m_6	$x' + y' + z$	M_6
1	1	1	xyz	m_7	$x' + y' + z'$	M_7

Table 2.4

Functions of Three Variables

x	Y	z	Function f_1	Function f_2
0	0	0	0	0
0	0	1	1	0
0	1	0	0	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1

Similarly, it may be easily verified that

$$f_2 = x'yz + xy'z + xyz' + xyz = m_3 + m_5 + m_6 + m_7$$

If we take the complement of $f^{=}$, we obtain the function f:

1 1

$f_1 = (x + y + z)(x + y' + z)(x' + y + z')(x' + y' + z)$

$= M_0 \# M_2 \# M_3 \# M_5 \# M_6$

Similarly, it is possible to read the expression for f_2 from the table:

$f_2 = (x + y + z)(x + y + z')(x + y' + z)(x' + y + z)$

$= M_0M_1M_2M_4$

Boolean functions expressed as a sum of minterms or product of maxterms are said to be in canonical form.

Sum of Minterms

1's under F for those combinations for which $A = 1$ and $BC = 01$. From the truth table, we can then read the five minterms of the function to be 1, 4, 5, 6, and 7.

Product of Maxterms

Each of the 2^{2^n} functions of n binary variables can be also expressed as a product ofmaxterms. To express a Boolean function as a product of maxterms, it must first be brought into a form of OR terms. This may be done by using the distributive law,$x + yz = (x + y)(x + z)$.

DIGITAL LOGIC GATES

Since Boolean functions are expressed in terms of AND, OR, and NOT operations, it is easier to implement a Boolean function with these types of gates. Factors to be weighed in considering the construction of other types of logic gates are (1) the feasibility and economy of producing the gate with physical components, (2) the possibility, more than two inputs, (3) the basic properties of the binary operator, such as commutativity and associativity, and (4) the ability of the gate to implement Boolean functions alone or in conjunction with other gates.

The small circle in the output of the graphic symbol of an inverter (referred to as a bubble) designates the logic complement. The triangle symbol by itself designates a buffer circuit. A buffer produces the transfer function, but does not produce a logic operation, since the binary value of the output is equal to the binary value of the input. This circuit is used for power amplification of the signal and is equivalent to two inverters connected in cascade.

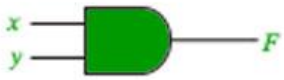
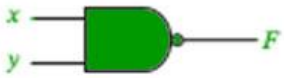
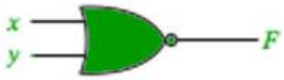
Name	Graphic symbol	Algebraic function	Truth table
AND		$F = x \cdot y$	x y | F 0 0 | 0 0 1 | 0 1 0 | 0 1 1 | 1
OR		$F = x + y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 1
Inverter		$F = x'$	x | F 0 | 1 1 | 0
Buffer		$F = x$	x | F 0 | 0 1 | 1
NAND		$F = (xy)'$	x y | F 0 0 | 1 0 1 | 1 1 0 | 1 1 1 | 0
NOR		$F = (x + y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 0
Exclusive-OR (XOR)		$F = xy' + x'y$ $= x \oplus y$	x y | F 0 0 | 0 0 1 | 1 1 0 | 1 1 1 | 0
Exclusive-NOR or equivalence		$F = xy + x'y'$ $= (x \oplus y)'$	x y | F 0 0 | 1 0 1 | 0 1 0 | 0 1 1 | 1

Figure 2.5
Digital Logic Gates

INTEGRATED CIRCUITS

An integrated circuit (IC) is fabricated on a die of a silicon semiconductor crystal, called a *chip,* containing the electronic components for constructing digital gates. The various gates are interconnected inside the chip to form the required circuit. Each IC has a numeric designation printed on the surface of the package for identification. Vendors provide data books, catalogs, and Internet websites that contain descriptions and information about the ICs that they manufacture.

Levels of Integration

The complexity of their circuits, as measured by the number of logic gates in a single package.

Small-scale integration (SSI) devices contain several independent gates in a single package. The inputs and outputs of the gates are connected directly to the pins in the package.

Medium-scale integration (MSI) devices have a complexity of approximately 10 to 1,000 gates in a single package.

Large-scale integration (LSI) devices contain thousands of gates in a single package. They include digital systems such as processors, memory chips, and programmable logic devices.

Very large-scale integration (VLSI) devices now contain millions of gates within a single package. Examples are large memory arrays and complex microcomputer chips. Because of their small size and low cost.

Digital Logic Families

The circuit technology is referred to as a *digital logic family*. Each logic family has its own basic electronic circuit upon which more complex digital circuits and components are developed. The basic circuit in each technology is a NAND, NOR, or inverter gate.

Many different logic families of digital integrated circuits have been introduced commercially. The following are the most popular:

TTL	transistor–transistor logic; ECL emitter-coupled logic;
MOS	metal-oxide semiconductor;
CMOS	complementary metal-oxide semiconductor.

FOUR-VARIABLE K-MAP

The map minimization of four-variable Boolean functions is similar to the methodused to minimize three-variable functions. Adjacent squares are defined to be squares next to each other. For example, m_0 and m_2 form adjacent squares, as do m_3 and m_{11}. The combination of adjacent squares that is useful during the simplification process is easily determined from inspection of the four-variable map:

One square represents one minterm, giving a term with four literals.Two adjacent squares represent a term with three literals.

Four adjacent squares represent a term with two literals.Eight adjacent squares represent a term with one literal.

Sixteen adjacent squares produce a function that is always equal to 1.

No other combination of squares can simplify the function. The next two examples show the procedure used to simplify four-variable Boolean functions.

Simplify the Boolean function

$F(w, x, y, z) = (0, 1,2, 4, 5, 6, 8, 9, 12, 13, 14)$

$F = y' + w'z' + xz'$

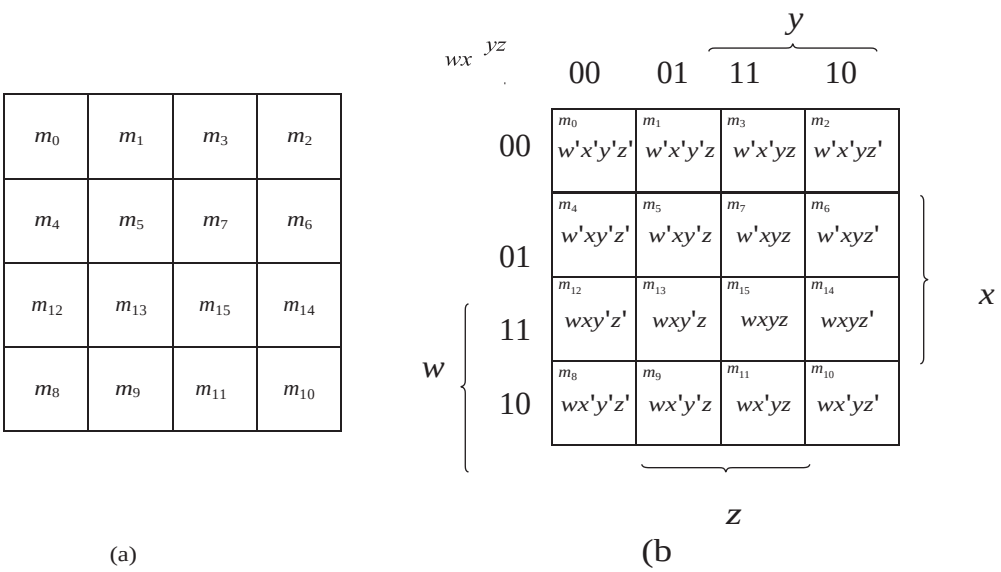


Figure 3.8

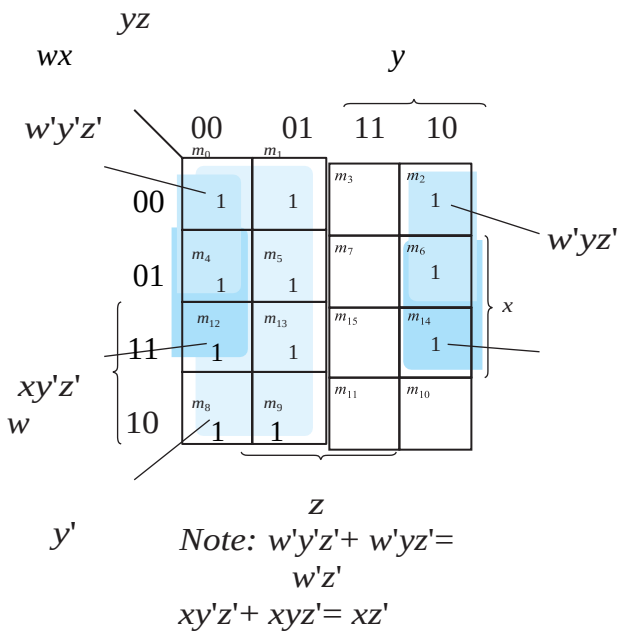


FIGURE 3.9

$F = y' + w'z' +$

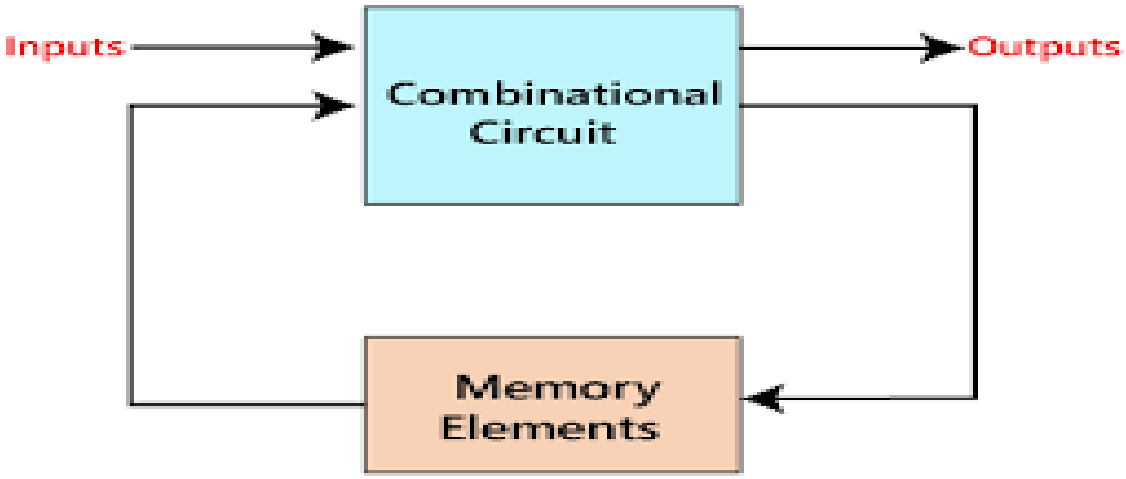
Five-Variable Map

Maps for more than four variables are not as simple to use as maps for four or fewer variables. A five-variable map needs 32 squares and a six-variable map needs 64 squares. When the number of variables becomes large, the number of squares becomes excessive and the geometry for combining adjacent squares becomes more involved. Maps for more than four variables are difficult to use.

UNIT-3

COMBINATIONAL CIRCUITS

A combinational circuit consists of an interconnection of logic gates. Combinational logic gates react to the values of the signals at their inputs and produce the value of the output signal, transforming binary information from the given input data to a required output data



DECIMAL ADDER

A decimal adder requires a minimum of nine inputs and five outputs, since four bits are required to code each decimal digit and the circuit must have an input and output carry.

BCD Adder

In examining the contents of the table, it becomes apparent that when the binary sum is equal to or less than 1001, the corresponding BCD number is identical, and therefore no conversion is needed. When the binary sum is greater than 1001, we obtain an invalid BCD representation. The addition of binary 6 (0110) to the binary sum converts it to the correct BCD representation and also produces an output carry as required.

Table 4.5
Derivation of BCD Adder

Binary Sum					BCD Sum					Decimal
K	Z_8	Z_4	Z_2	Z_1	*C*	S_8	S_4	S_2	S_1	
0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	1	0	0	0	0	1	1
0	0	0	1	0	0	0	0	1	0	2
0	0	0	1	1	0	0	0	1	1	3
0	0	1	0	0	0	0	1	0	0	4
0	0	1	0	1	0	0	1	0	1	5
0	0	1	1	0	0	0	1	1	0	6
0	0	1	1	1	0	0	1	1	1	7
0	1	0	0	0	0	1	0	0	0	8
0	1	0	0	1	0	1	0	0	1	9
0	1	0	1	0	1	0	0	0	0	10
0	1	0	1	1	1	0	0	0	1	11
0	1	1	0	0	1	0	0	1	0	12
0	1	1	0	1	1	0	0	1	1	13
0	1	1	1	0	1	0	1	0	0	14
0	1	1	1	1	1	0	1	0	1	15
1	0	0	0	0	1	0	1	1	0	16
1	0	0	0	1	1	0	1	1	1	17
1	0	0	1	0	1	1	0	0	0	18
1	0	0	1	1	1	1	0	0	1	19

The condition for a correction and an output carry can be expressed by the Boolean function

$$C = K + Z_8Z_4 + Z_8Z_2$$

When $C = 1$, it is necessary to add 0110 to the binary sum and provide an output carry for the next stage.

The two decimal digits, together with the input carry, are first added in the top four-bit adder to produce the binary sum. When the output carry is equal to 0, nothing is added to the binary sum. When it is equal to 1, binary 0110 is added to the binary sum through the bottom four-bit adder. The output carry generated from the bottom

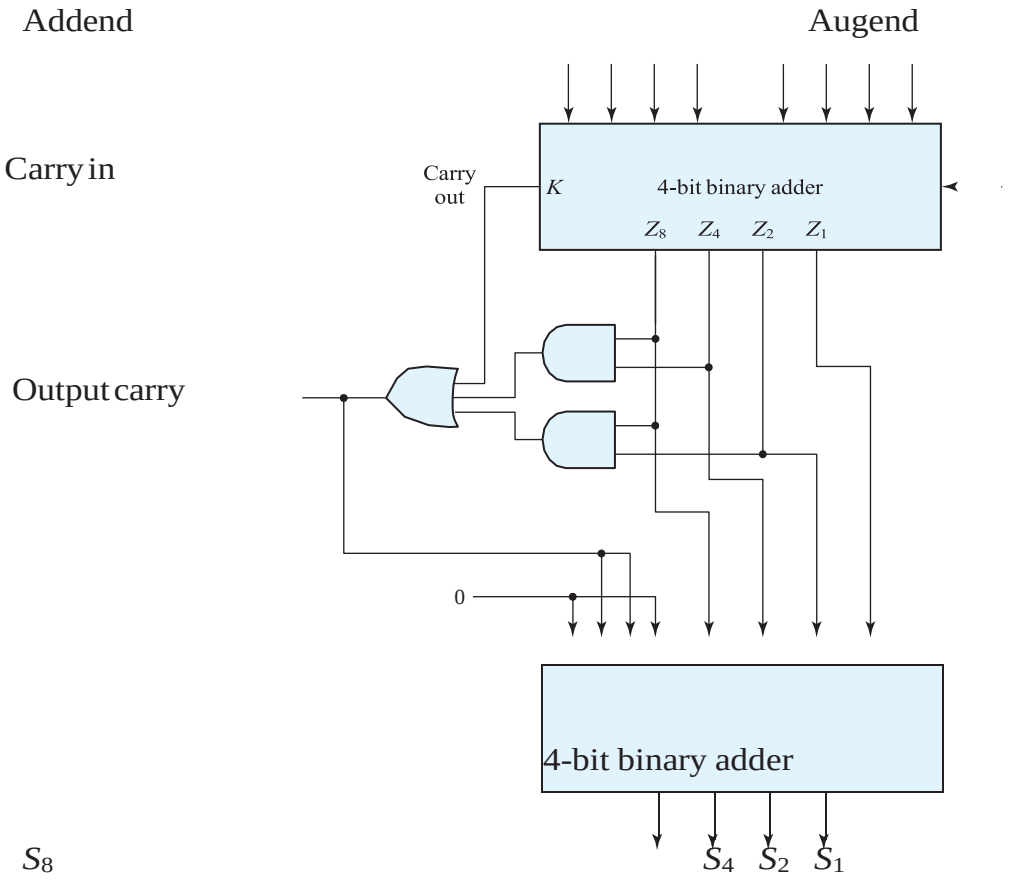
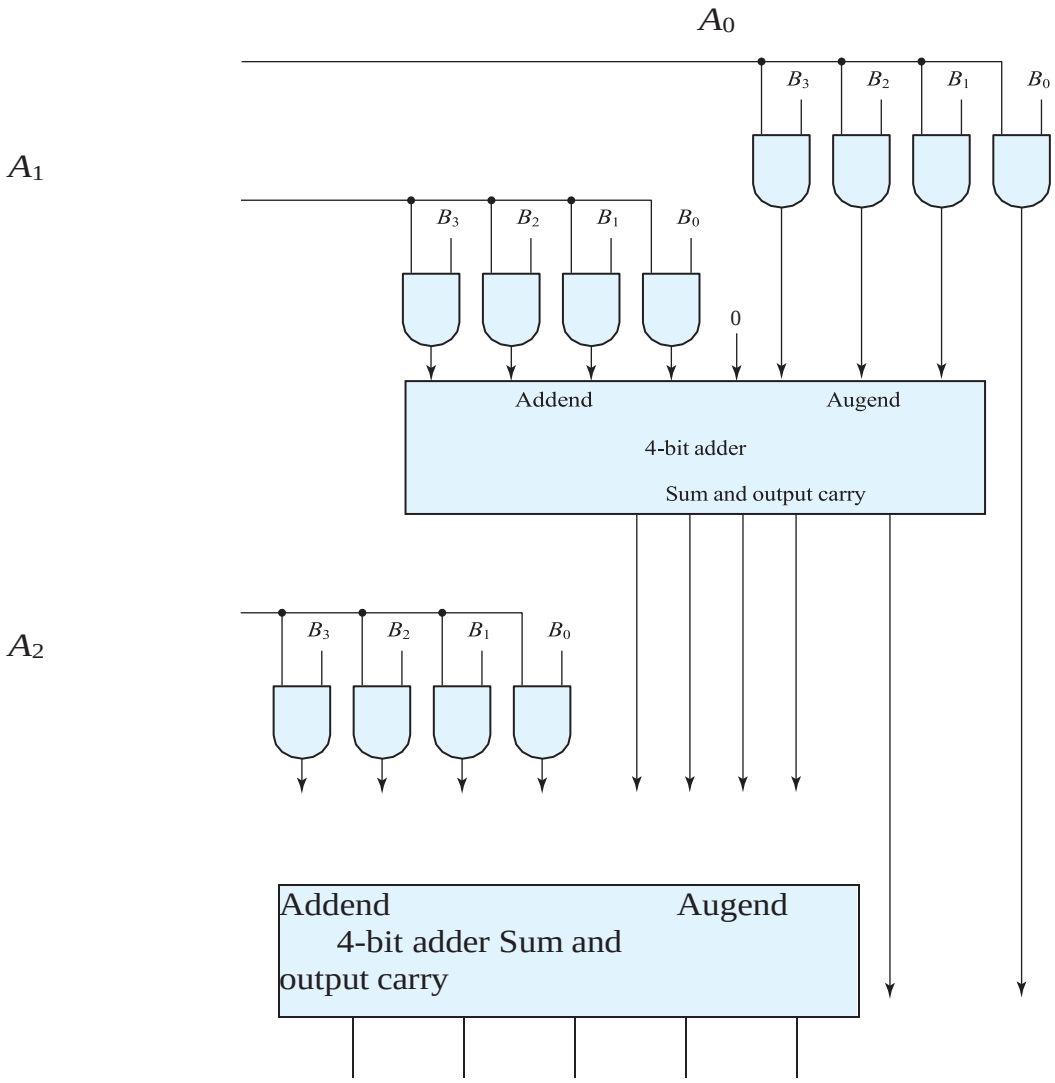


FIGURE 4.14
Block diagram of a BCD adder

adder can be ignored, since it supplies information already available at the output carry terminal.

BINARY MULTIPLIE R

The multiplicand is multiplied by each bit of the multiplier, starting from the least significant bit. Each such multiplication forms a partial product. Successive partial products are shifted one position to the left. The final product is obtained from the sum of the partial products.


A_0
A_1
A_2
B_3 B_2 B_1 B_0
0
Addend
Augend
4-bit adder
Sum and output carry
Addend
Augend
4-bit adder Sum and
output carry

UNIT-4

ENCODERS

An encoder is a digital circuit that performs the inverse operation of a decoder. An encoder has 2^n (or fewer) input lines and n output lines. The output lines, as an aggregate, generate the binary code corresponding to the input value.

Theseconditions can be expressed by the following Boolean output functions:
$z = D_1 + D_3 + D_5 + D_7 y = D_2 + D_3 + D_6 + D_7 x = D_4 + D_5 + D_6 + D_7$
The encoder can be implemented with three OR gates.

Table 4.7
Truth Table of an Octal-to-Binary Encoder

Inputs								Outputs		
D_0	D_1	D_2	D_3	D_4	D_5	D_6	D_7	X	y	z
1	0	0	0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0	0	0	1
0	0	1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	1	0	0	0	1	0	0
0	0	0	0	0	1	0	0	1	0	1
0	0	0	0	0	0	1	0	1	1	0
0	0	0	0	0	0	0	1	1	1	1

Priority Encoder

A priority encoder is an encoder circuit that includes the priority function. The operationof the priority encoder is such that if two or more inputs are equal to 1 at the same time,the input having the highest priority will take precedence.

Table 4.8
Truth Table of a Priority Encoder

Inputs				Outputs		
D_0	D_1	D_2	D_3	x	y	V
0	0	0	0	X	X	0
1	0	0	0	0	0	1
X	1	0	0	0	1	1
X	X	1	0	1	0	1
X	X	X	1	1	1	1

Although the table has only five rows, when each X in a row is replaced first by 0 and then by 1, we obtain all 16 possible input combination's. For example, the fourth row in the table, with inputs XX10, represents the four minterms 0010, 0110, 1010, and 1110. The simplified Boolean expressions for the priority encoder are obtained from the maps.

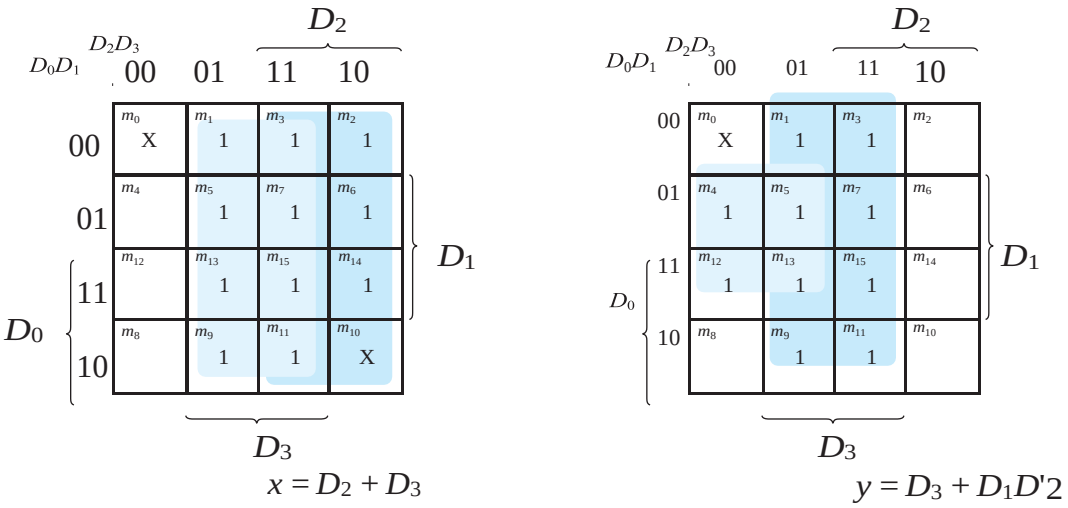


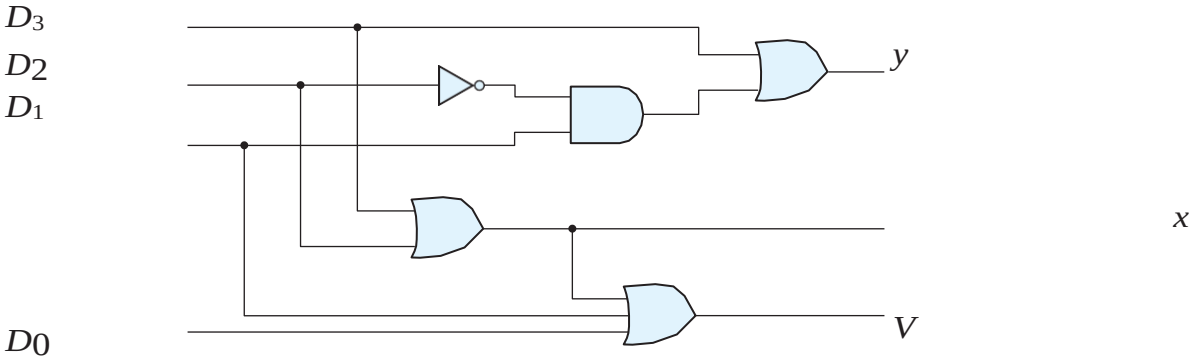
FIGURE 4.22
Maps for a priority encoder

The condition for output V is an OR function ofall the input variables.

$x = D_2 + D_3$

$y = D_3 + D_1 D'_2$

$V = D_0 + D_1 + D_2 + D_3$



Four-input priority encoder

C_5 C_4 C_2 C_1 C_0
C_3

SEQUENTIAL CIRCUITS

The storage elements are devices capable of storing binary information. The binary information stored in these elements at any given time defines the *state* of the sequential circuit atthat time. These external inputs also determine the condition for changing the state in the storage elements. The block diagram demonstrates that the outputs in a sequential circuit are a function not only of the inputs, but also of the present state of the storage elements. The next state of the storage elements is also a function of external inputs and the present state. Thus**,** a sequential circuit is specified by a time sequence of inputs, outputs, and internal states**.** In contrast, the outputs of combinational logic depend only on the present values of the inputs.

There are two main types of sequential circuits, and their classification is a function ofthe timing of their signals. A *synchronous* sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time. The behavior of an *asynchronous* sequential circuit depends upon the input signals at any instant of time *and* the order in which the inputs change. The storage elements commonly used in asynchronous sequential circuits are time-delay devices.

A synchronous sequential circuit employs signals that affect the storage elements at only discrete instants of time. Synchronization is achieved by a timing device called a *clock generator*, which provides a clock signal having the form of a periodic train of *clock pulses*. The clock signal is commonly denoted by the identifiers *clock* and *clk*. The clockpulses are distributed throughout the system in such a way that storage elements are affected only with the arrival of each pulse.

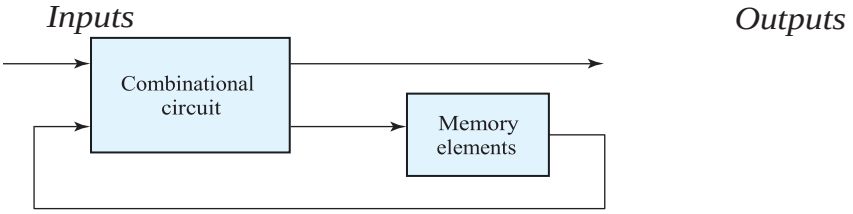
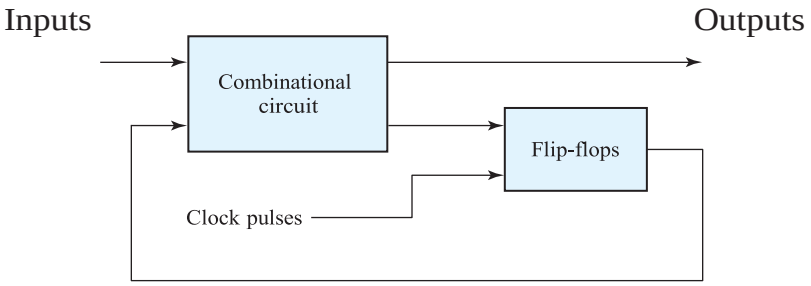


FIGURE 5.1
Block diagram of sequential circuit

There are two main types of sequential circuits, and their classification is a function ofthe timing of their signals. A *synchronous* sequential circuit is a system whose behavior can be defined from the knowledge of its signals at discrete instants of time. The behavior of an *asynchronous* sequential circuit depends upon the input signals at any instant of time *and* the order in which the inputs change. The storage elements commonly used in asynchronous sequential circuits are time-delay devices.

A synchronous sequential circuit employs signals that affect the storage elements at only discrete instants of time. Synchronization is achieved by a timing device called a *clock generator*, which provides a clock signal having the form of a periodic train of *clock pulses*. The clock signal is commonly denoted by the identifiers *clock* and *clk*. The clockpulses are distributed throughout the system in such a way that storage elements are affected only with the arrival of each pulse.



(a) Block diagram

(b) Timing diagram of clock pulses

FIGURE 5.2

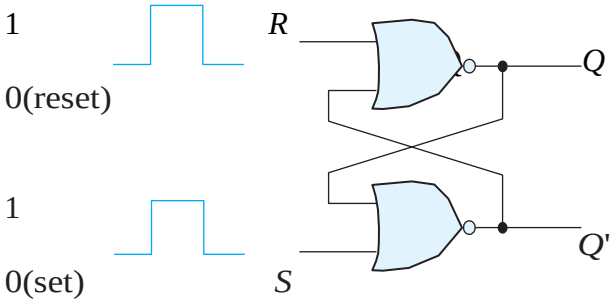
Synchronous clocked sequential circuit

STORAGE ELEMENTS: LATCHES

A storage element in a digital circuit can maintain a binary state indefinitely until directed by an input signal to switch states. The major differences among various types of storage elements are in the number of inputs they possess and in the manner in which the inputs affect the binary state. *Storage elements that operate with signal levels (rather than signal transitions) are referred to as latches*; *those controlled by a clock transition are flip-flops*. Latches are said to be level sensitive devices; flip-flops are edge-sensitive devices. The two types of storage elements are related because latches are the basic circuits from which all flip-flops are con- structed. Although latches are useful for storing binary information and for the design of asynchronous sequential circuits, they are not practical for use as storage elements in synchronous sequential circuits

SR Latch

The *SR* latch is a circuit with two cross-coupled NOR gates or two cross-coupled NAND gates, and two inputs labeled *S* for set and *R* for reset. The *SR* latch constructed with two cross-coupled NOR gates. The latch has two useful states. When output $Q = 1$ and $Q' = 0$, the latch is said to be in the *set state*. When $Q = 0$ and $Q' = 1$, it is in the *reset state*. Outputs Q and Q' are normally the complement of each other. However, when both inputs are equal to 1 at the same time, a condition in which both outputs are equal to 0 (rather than be mutually complementary) occurs. If both inputs are then switched to 0 simultaneously, the device will enter an unpredictable or undefined state or a meta- stable state.



S	R	Q	Q'
1	0	1	0
0	0	1	0
0	1	0	1
0	0	0	1
1	1	0	0

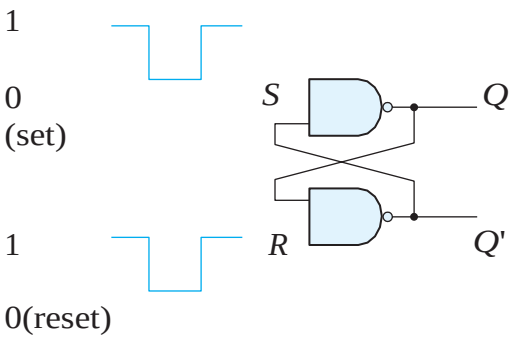
(after $S = 1$, $R = 0$)

(after $S = 0$, $R = 1$)
(forbidden)

(a) Logic diagram (b) Function table

FIGURE 5.3

_SR_ latch with NOR gates



S	R	Q	Q'
1	0	0	1
1	1	0	1 (
0	1	1	0
1	1	1	0 (
0	0	1	1 (

after $S = 1$, $R = 0$)

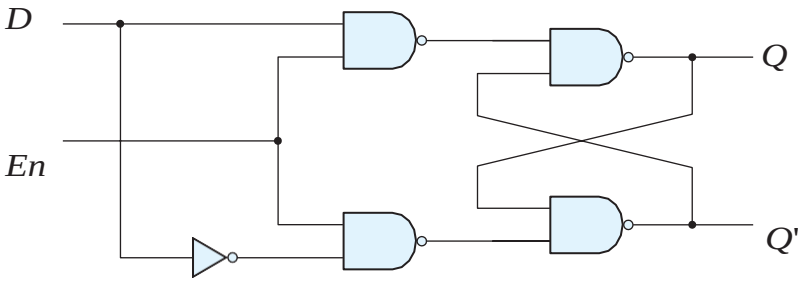
after $S = 0$, $R = 1$)
forbidden)

(a) Logic diagram (b) Function table

FIGURE 5.4

_D_ Latch (Transparent Latch)

One way to eliminate the undesirable condition of the indeterminate state in the *SR* latch is to ensure that inputs *S* and *R* are never equal to 1 at the same time. This is done in the *D* latch, this latch has only two inputs: *D* (data) and *En* (enable).



En	*D*	Next state of *Q*
0	X	No change
1	0	$Q = 0$; reset state
1	1	$Q = 1$; set state

(a) Logic diagram (b) Function table

FIGURE 5.6

_D_ latch

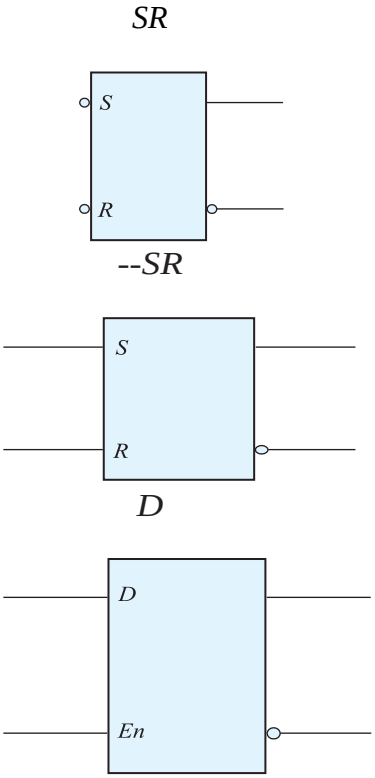


FIGURE 5.7
Graphic symbols for latches

The *D* input goes directly to the *S* input, and its complement is appliedto the *R* input. As long as the enable input is at 0, the cross-coupled *SR* latch has both inputs at the 1 level and the circuit cannot change state regardless of the value of *D*.The *D* input is sampled when *En* = 1. If *D* = 1, the *Q* output goes to 1, placing the circuit in the set state. If *D* = 0, output *Q* goes to 0, placing the circuit in the resetstate.

The *D* latch receives that designation from its ability to hold *data* in its internal storage. It is suited for use as a temporary storage for binary information between a unit and its environment. The binary information present at the data input of the *D* latch istransferred to the *Q* output when the enable input is asserted. The output follows changes in the data input as long as the enable input is asserted. This situation providesapath from input *D* to the output, and for this reason, the circuit is often called a *transparent* latch. When the enable input signal is de-asserted, the binary information that was present at the data input at the time the transition occurred is retained (i.e., stored)at the *Q* output until the enable input is asserted again. Note that an inverter could beplaced at the enable input. Then, depending on the physical circuit, the external enablingsignal will be a value of 0 (active low) or 1 (active high).

SHIFT REGISTERS

A register capable of shifting the binary information held in each cell to its neighboring cell, in a selected direction, is called a *shift register*. The logical configuration of a shift register consists of a chain of flip-flops in cascade, with the output of one flip-flop connected to the input of the next flip-flop. All flip-flops receive common clock pulses, which activate the shift of data from one stage to the next.

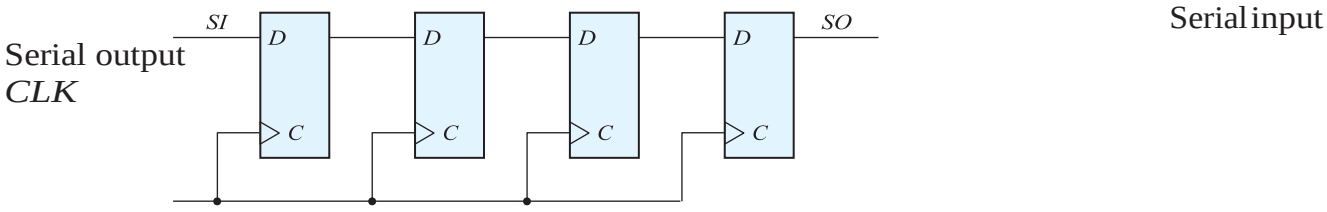
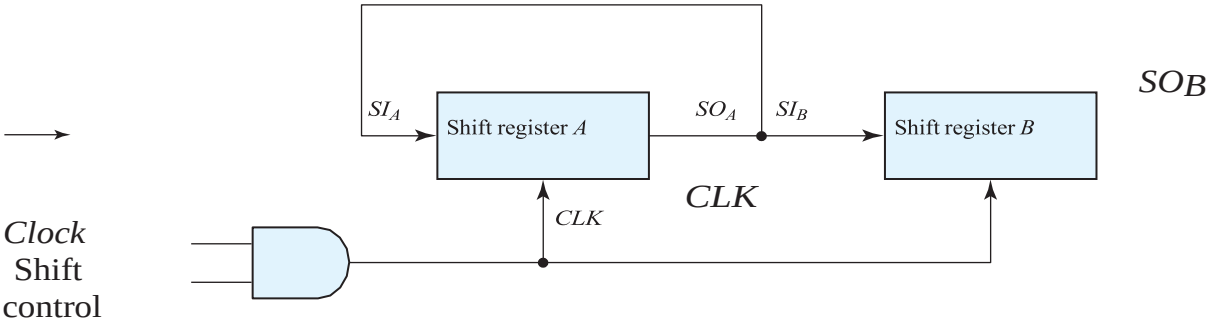


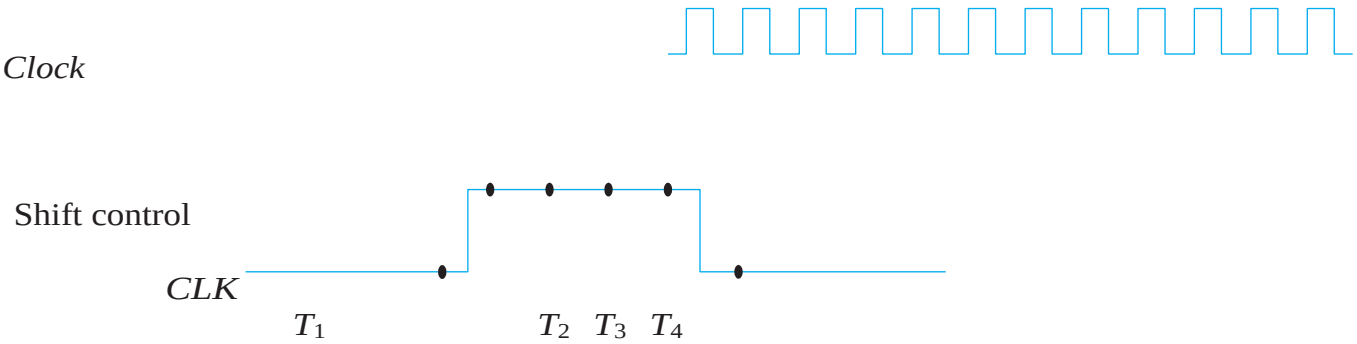
FIGURE6.3

The *serial input* determines what goes into the leftmost flip-flop during the shift. The *serial output* is taken from the output of the rightmost flip-flop. Sometimes it is necessary tocontrol the shift so that it occurs only with certain pulses, but not with others.

Serial Transfer

The data path of a digital system is said to operate in serial mode when information is transferred and manipulated one bit at a time. Information is transferred one bit at a time by shifting the bits out of the source register and into the destination register. This type of transfer is in contrast to parallel transfer, whereby all the bits of the register are transferred at the same time.





Serial transfer from register *A* to register *B*

Serial-Transfer Example

Timing Pulse	Shift Register *A*	Shift Register *B*
Initial value	1 0 1 1	0 0 1 0
After T_1	1 1 0 1	1 0 0 1
After T_2	1 1 1 0	1 1 0 0
After T_3	0 1 1 1	0 1 1 0
After T_4	1 0 1 1	1 0 1 1

The registers have a single serial input and a single serial output. The information is transferred one bit at a time while the registers are shifted in the same direction.

Serial Addition

Operations in digital computers are usually done in parallel because that is a faster mode of operation. Serial operations are slower because a data path operation takes several clock cycles, but serial operations have the advantage of requiring fewer hardware components. In VLSI circuits, they require less silicon area on a chip.

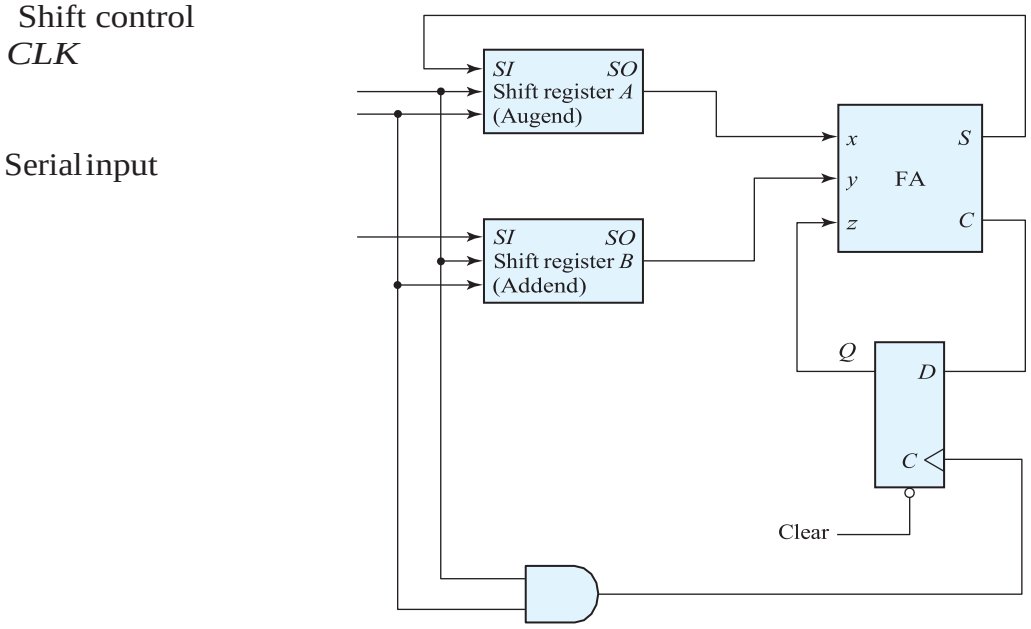


FIGURE 6.5
Serial adder

To it through its serial input. The second number is then added to the contents of register *A*, while a third number is transferred serially into register *B*. This can be repeated to perform the addition of two, three, or more four-bit numbers and accumulate their sum in register *A*.

Table 6.2
State Table for Serial Adder

Present State	Inputs	Next State	Output	Flip-Flop Inputs
Q	x Y	Q	S	J_Q K_Q
0	0 0	0	0	0 X
0	0 1	0	1	0 X
0	1 0	0	1	0 X
0	1 1	1	0	1 X
1	0 0	0	1	X 1
1	0 1	1	0	X 0
1	1 0	1	0	X 0
1	1 1	1	1	X 0

$J_Q = xy$
$K_Q = x'y' = (x + y)'$
$S = x \{ y \{ Q\}\}$

The circuit diagram is shown in Fig. 6.6. The circuit consists of three gates and a *JK* flip-flop. The two shift registers are included in the diagram to show the complete serial adder. Note that output S is a function not only of x and y, but also of the present state of Q. The next state of Q is a function of the present state of Q and of the values of x and y that come out of the serial outputs of the shift registers.

Universal Shift Register

If the flip-flop outputs of a shift register are accessible, then information entered serially by shifting can be taken out in parallel from the outputs of the flip-flops. If a parallel load capability is added to a shift register, then data entered in parallel can be taken out in serial fashion by shifting the data stored in the register.

The most general shift register has the following capabilities:

1. A *clear* control to clear the register to 0.
2. A *clock* input to synchronize the operations.

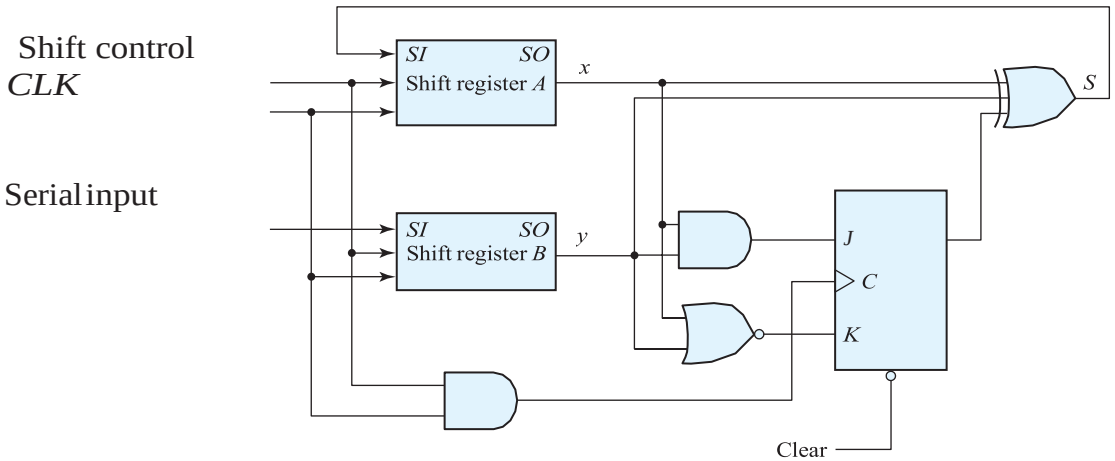


FIGURE 6.6
Second form of serial adder

3. A *shift-right* control to enable the shift-right operation and the *serial input* and *output* lines associated with the shift right.
4. A *shift-left* control to enable the shift-left operation and the *serial input* and *output* lines associated with the shift left.
5. A *parallel-load* control to enable a parallel transfer and the *n* input lines associated with the parallel transfer.
6. *n* parallel output lines.
7. A control state that leaves the information in the register unchanged in responseto the clock. Other shift registers may have only some of the preceding functions, with at least one shift operation.

A register capable of shifting in one direction only is a *unidirectional* shift register. One that can shift in both directions is a *bidirectional* shift register. If the register hasboth shifts and parallel-load capabilities, it is referred to as a *universal shift register.*

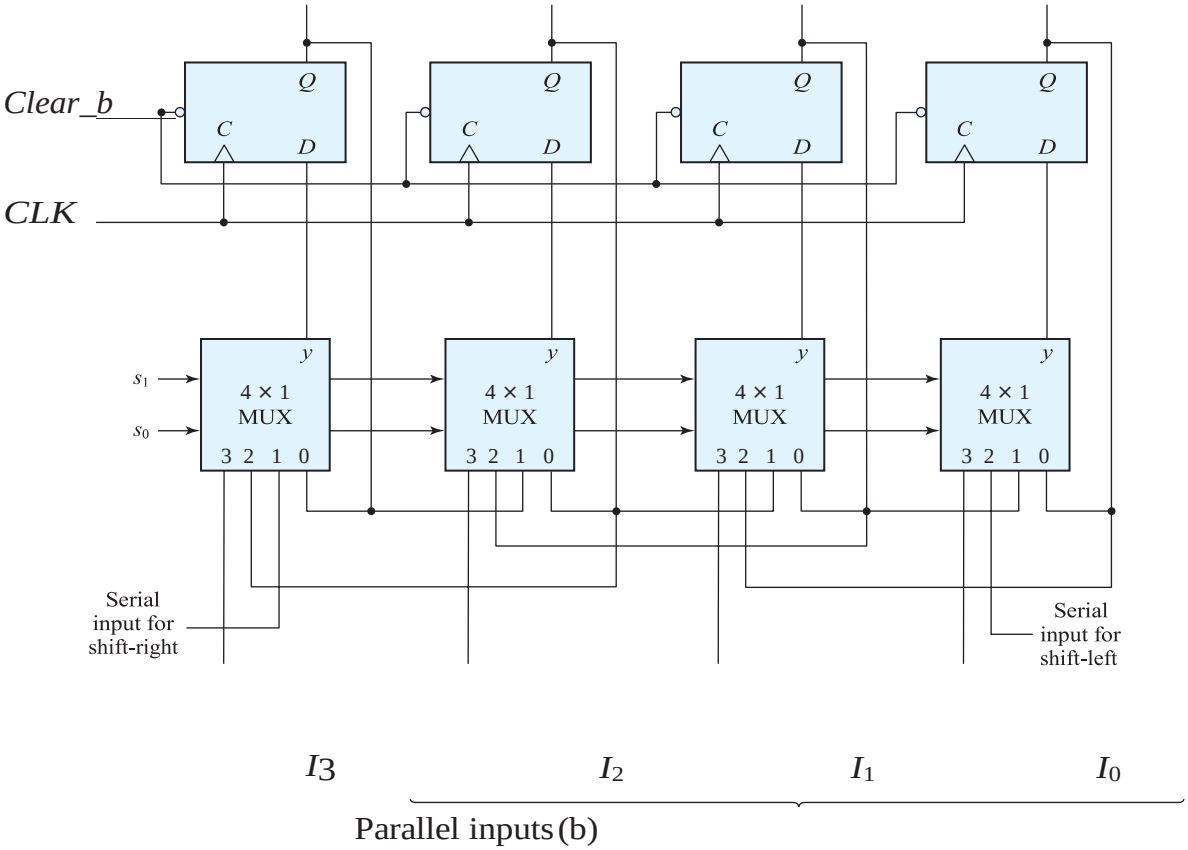


FIGURE 6.7
Four-bit universal shift register

Table 6.3

Function Table for the Register of Fig. 6.7

Mode Control

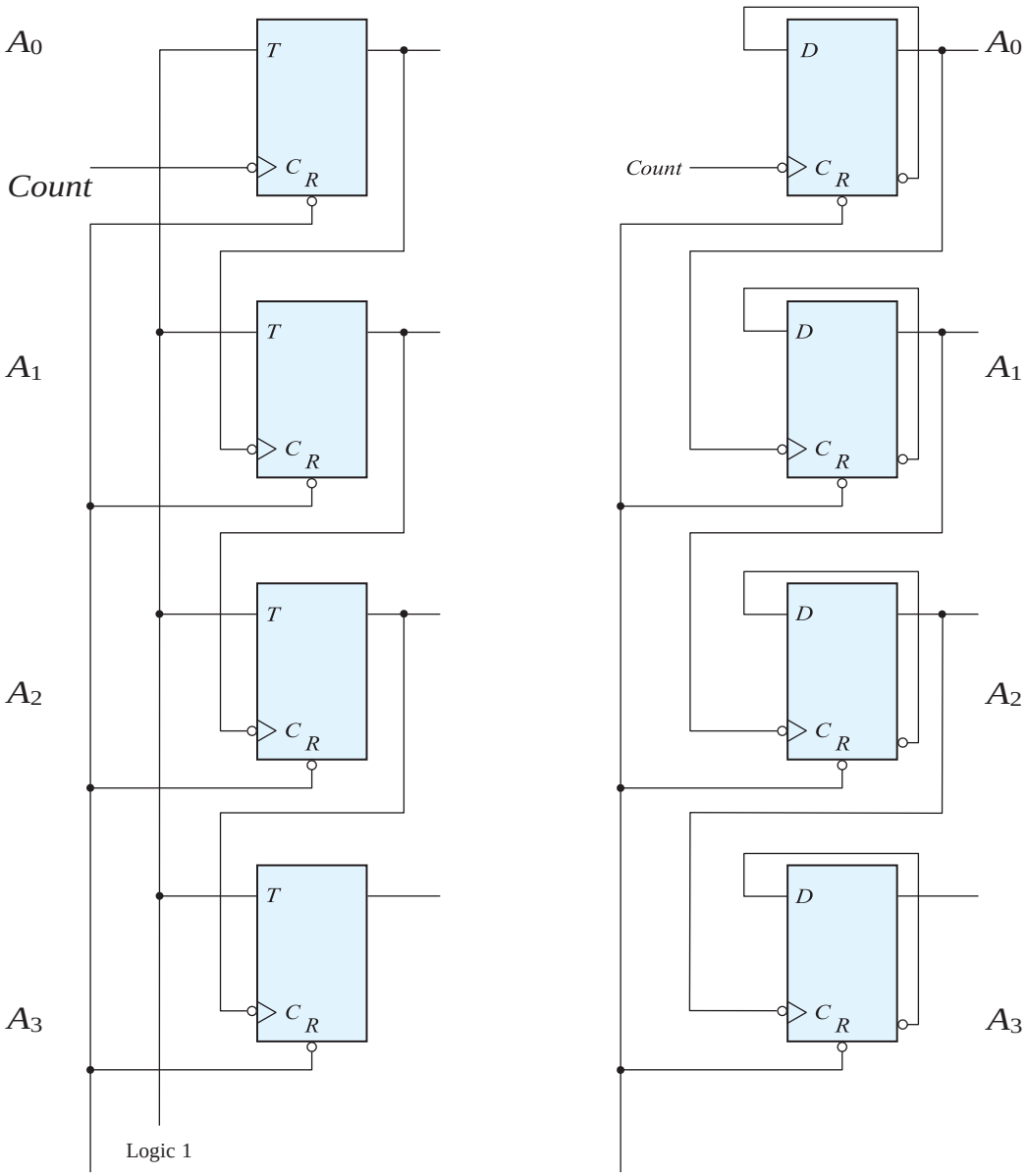
s_1	s_0	Register Operation
0	0	No change
0	1	Shift right
1	0	Shift left
1	1	Parallel load

When s_1s_0 = 01, terminal 1 of the multiplexer inputs has a path to the *D* inputs of the flip-flops. This causes a shift-right operation, with the serial input transferred into flip-flop A_3. When s_1s_0 = 10, a shift-left operation results, with the other serial input going into flip-flop A_0. Finally, when s_1s_0 = 11, the binary information on the parallel input lines is transferred into the register simultaneously during the next clock edge. Note that data enters *MSB in* for a shift-right operation and enters *LSB_in* for a shift-left operation. *Clear_b* is an active-low signal that clears all of the flip-flops.

RIPPLE COUNTERS

A register that goes through a prescribed sequence of states upon the application of input pulses is called a *counter*.

In a ripple counter, a flip-flop output transition serves as a source for triggering other flip-flops. In other words, the *C* input of some or all flip-flops are triggered, not by the common clock pulses, but rather by the transition that occurs in other flip-flop outputs.
A binary ripple counter consists of a series connection of complementing flip-flops, with the output of each flip-flop connected to the *C* input of the next higher order flip-flop. The flip-flop holding the least significant bit receives the incoming count pulses. A complementing flip-flop can be obtained from a *JK* flip-flop with the *J* and *K* inputs tied together or from a *T* flip-flop. A third possibility is to use a *D* flip-flop with the complement output connected to the *D* input.



(a) With *T* flip-flops (b) With *D* flip-flops

FIGURE 6.8

A binary counter with a reverse count is called a *binary countdown counter*. In a countdown counter, the binary count is decremented by 1 with every input count pulse. The count of a four-bit countdown counter starts from binary 15 and continues to binary counts 14, 13, 12, . . . , 0 and then back to 15. A list of the count sequence of a binary countdown counter shows that the least significant bit is complemented with every count pulse.

BCD Ripple Counter

A decimal counter follows a sequence of 10 states and returns to 0 after the count of 9. Such a counter must have at least four flip-flops to represent each decimal digit, sincea decimal digit is represented by a binary code with at least four bits. A decimal counter is similar to a binary counter, except that the state after 1001 (the code for decimal digit 9) is 0000 (the code for decimal digit 0).

A ripple counter is an asynchronous sequential circuit. Signals that affect the flip-floptransition depend on the way they change from 1 to 0. The operation of the counter can

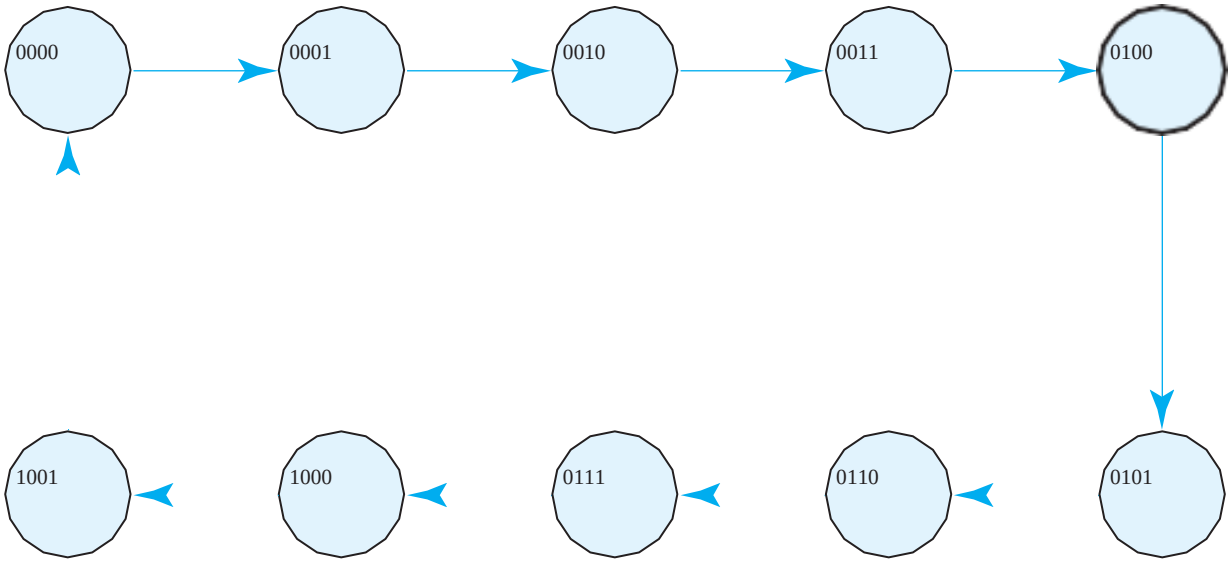
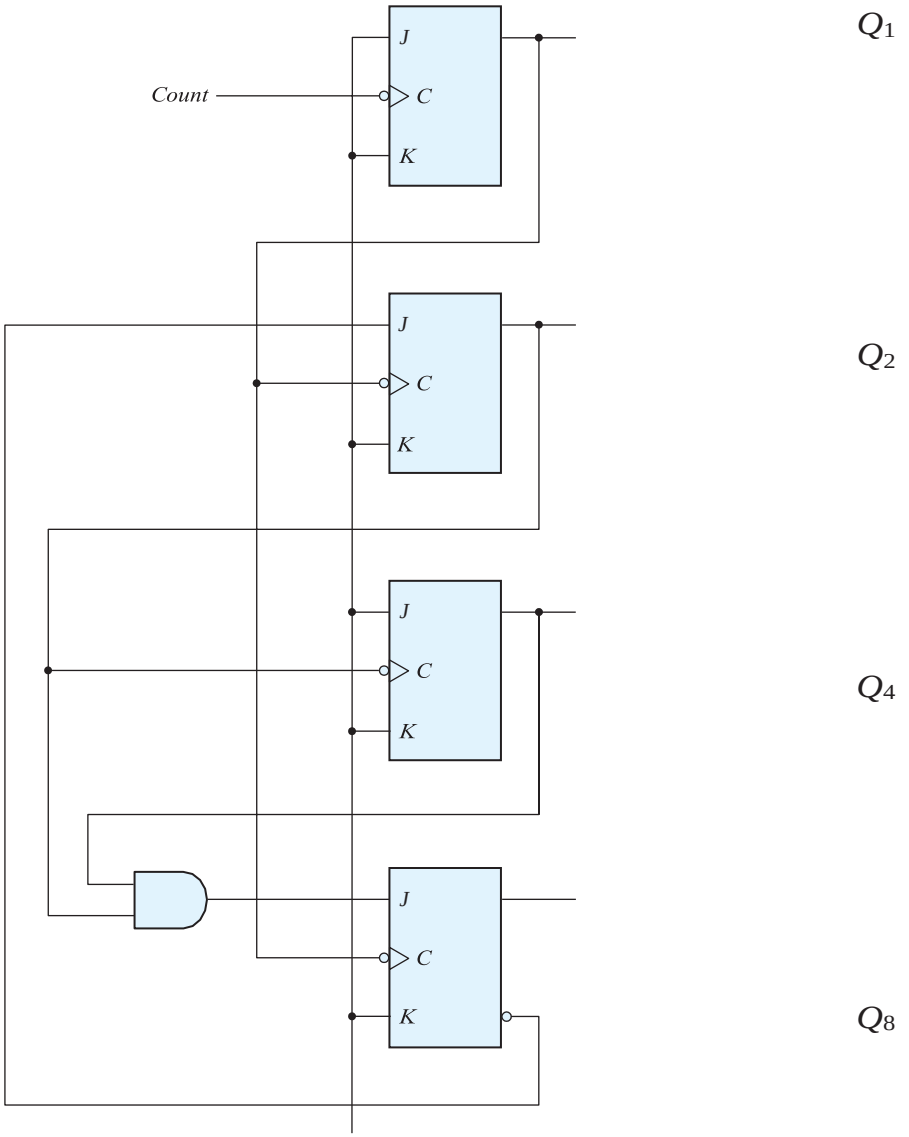


FIGURE 6.9

State diagram of a decimal BCD counter



Logic 1

FIGURE 6.10

BCD ripple counter

These conditions are derived from the logic diagram and from knowledge of how a *JK* flip-flop operates. Remember that when the *C* input goes from 1 to 0, the flip-flop is set if $J = 1$, is cleared if $K = 1$, is complemented if $J = K = 1$, and is left unchanged if $J = K = 0$.

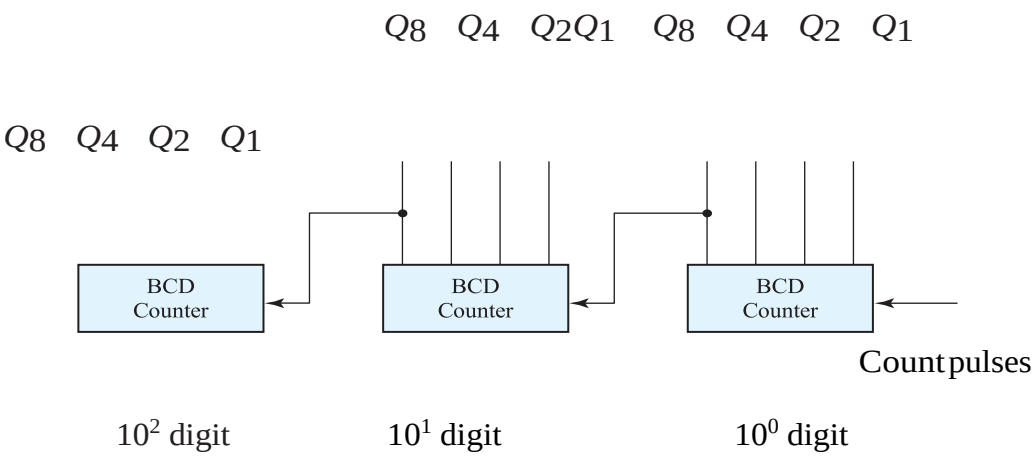


FIGURE 6.11

Block diagram of a three-decade decimal BCD counter