DIGITAL LOGIC DESIGN

GATE 2023

1.1-1.8,
2.1-2.7,
3,
4,
6,
7.1-7.7

See each and every diagram while revising

Topic	Sub Topics
Boolean Algebra	Minimal Expression
	K-Maps
	SOP and POS
Number System	Fixed Point Representation
	IEEE Representation
	Number Representation
Combinational Circuits	Adders
	Digital Circuit
	Multiplexer
Sequential Circuits	Circuit Output
	Digital Counter
	Flip Flops

JANUARY 12, 2022
QUANTUM CITY
Nowhere

1. DIGITAL SYSTEM AND BINARY NUMBERS

1.1) Digital System : Any set that is restricted to a finite number of elements contains discrete information. Examples of discrete sets are the 10 decimal digits, the 26 letters of the alphabet, the 52 playing cards, and the 64 squares of a chessboard. Early digital computers were used for numeric computations. In this case, the discrete elements were the digits.

Discrete elements of information are represented in a digital system by physical quantities called *signals*.

The signals in most present-day electronic digital systems use just two discrete values and are therefore said to be binary. A binary digit, called a bit, has two values: 0 and 1. Discrete elements of information are represented with groups of bits called binary codes. For example, the decimal digits 0 through 9 are represented in a digital system with a code of four bits (e.g., the number 7 is represented by 0111).

1.2) Binary Numbers :

In general, a number with a decimal point is represented by a series of coefficients:

a5a4a3a2a1a0. a-1a-2a-3

The coefficients aj are any of the 10 digits (0, 1, 2, …, 9), and the subscript value j gives the place value and, hence, the power of 10 by which the coefficient must be multiplied. Thus, the preceding decimal number can be expressed as

105 a5 + 104 a4 + 103 a3 + 102 a2 + 101 a1 + 100 a0 + 10-1 a-1 + 10-2 a-2 + 10-3 a-3.

NOTE : The letters of the alphabet are used to supplement the 10 decimal digits when the base of the number is greater than 10.

augend:	101101	minuend:	101101	multiplicand:	1011
addend:	+100111	subtrahend:	−100111	multiplier:	× 101
sum:	1010100	difference:	000110		1011
					0000
				partial product:	1011
				product:	110111

Any carry obtained in a given significant position is used by the pair of digits one significant position higher. Subtraction is slightly more complicated. The rules are still the same as in decimal, except that the borrow in a given significant position adds 2 to a minuend digit. (A borrow in the decimal system adds 10 to a minuend digit.) Multiplication is simple: The multiplier digits are always 1 or 0; therefore, the partial products are equal either to a shifted (left) copy of the multiplicand or to 0.

(x*y)r = (x)r * (y)r, where * is multiplication operator.

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

1.3) Binary base conversions :

Convert decimal 41 to binary. First, 41 is divided by 2 to give an integer quotient of 20 and a remainder of $\frac{1}{2}$. Then the quotient is again divided by 2 to give a new quotient and remainder. The process is continued until the integer quotient becomes 0. The *coefficients* of the desired binary number are obtained from the *remainders* as follows:

	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 $(41)_{10} = (a_5a_4a_3a_2a_1a_0)_2 = (101001)_2$.

The arithmetic process can be manipulated more conveniently as follows:

Integer	Remainder	
41		
20	1	
10	0	
5	0	
2	1	
1	0	
0	1	101001 = answer

Conversion from decimal integers to any base-*r* system is similar to this example, except that division is done by *r* instead of 2.

Convert $(0.513)_{10}$ to octal.

$$0.513 \times 8 = 4.104$$
$$0.104 \times 8 = 0.832$$
$$0.832 \times 8 = 6.656$$
$$0.656 \times 8 = 5.248$$
$$0.248 \times 8 = 1.984$$
$$0.984 \times 8 = 7.872$$

The answer, to seven significant figures, is obtained from the integer part of the products:

$$(0.513)_{10} = (0.406517\ldots)_8$$

Convert decimal 153 to octal. The required base *r* is 8. First, 153 is divided by 8 to give an integer quotient of 19 and a remainder of 1. Then 19 is divided by 8 to give an integer quotient of 2 and a remainder of 3. Finally, 2 is divided by 8 to give a quotient of 0 and a remainder of 2. This process can be conveniently manipulated as follows:

153	
19	1
2	3
0	2 = $(231)_8$

The conversion of a decimal *fraction* to binary is accomplished by a method similar to that used for integers. However, multiplication is used instead of division, and integers instead of remainders are accumulated. Again, the method is best explained by example.

The conversion of decimal numbers with both integer and fraction parts is done by converting the integer and the fraction separately and then combining the two answers. Using the results of Examples 1.1 and 1.3, we obtain

$$(41.6875)_{10} = (101001.1011)_2$$

From Examples 1.2 and 1.4, we have

$$(153.513)_{10} = (231.406517)_8$$

1.4) Octal to hexadecimal :

The conversion from binary to octal is easily accomplished by partitioning the binary number into groups of three digits each, starting from the binary point and proceeding to the left and to the right. The corresponding octal digit is then assigned to each group.

$$\begin{array}{cccccccccc} (10 & 110 & 001 & 101 & 011 & \cdot & 111 & 100 & 000 & 110)_2 = (26153.7406)_8 \\ 2 & 6 & 1 & 5 & 3 & & 7 & 4 & 0 & 6 \end{array}$$

$$\begin{array}{cccccccc} (10 & 1100 & 0110 & 1011 & \cdot & 1111 & 0010)_2 = (2C6B.F2)_{16} \\ 2 & C & 6 & B & & F & 2 \end{array}$$

Each octal digit is converted to its three-digit binary equivalent. Similarly, each hexadecimal digit is converted to its four-digit binary equivalent. The procedure is illustrated in the following examples:

$$\begin{array}{cccccccccc} (673.124)_8 = (110 & 111 & 011 & \cdot & 001 & 010 & 100)_2 \\ 6 & 7 & 3 & & 1 & 2 & 4 \end{array}$$

$$\begin{array}{cccccc} (306.D)_{16} = (0011 & 0000 & 0110 & \cdot & 1101)_2 \\ 3 & 0 & 6 & & D \end{array}$$

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

1.5) Complement of numbers :

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.

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, so the 9's complement of N is $(10^n - 1) - N$. In this case, 10 n represents a number that consists of a single 1 followed by n 0's. 10 n - 1 is a number represented by n 9's. For example, if n = 4, we have 104 = 10,000 and 104 - 1 = 9999. It follows that the 9's complement of a decimal number is obtained by subtracting each digit from 9.
The 9's complement of 546700 is 999999 - 546700 = 453299.
The 9's complement of 012398 is 999999 - 012398 = 987601.

For binary numbers, r = 2 and r - 1 = 1, so the 1's complement of N is $(2^n - 1) - N$. Again, 2^n is represented by a binary number that consists of a 1 followed by n 0's. $2^n - 1$ is a binary number represented by n 1's. For example, if n = 4, we have $2^4 = (10000)_2$ and $2^4 - 1 = (1111)_2$. Thus, the 1's complement of a binary number is obtained by subtracting each digit from 1.
The 1's complement of 1011000 is 0100111. The 1's complement of 0101101 is 1010010. The (r - 1)'s complement of octal or hexadecimal numbers is obtained by subtracting each digit from 7 or F (decimal 15), respectively.

Radix Complement : The r's complement of an n-digit number N in base r is defined as $r^n - N$ for N != 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 $r^n - N = [(r^n - 1) - N] + 1$.
Thus, the 10's complement of decimal 2389 is 7610 + 1 = 7611 and is obtained by adding 1 to the 9's complement value.
The 2's complement of binary 101100 is 010011 + 1 = 010100 and is obtained by adding 1 to the 1's-complement value.

$10^n - N$ which is the 10's complement of N, can be formed also by leaving all least significant 0's unchanged, *subtracting the first nonzero least significant digit from 10, and subtracting all higher significant digits from 9*. Thus, the 10's complement of 012398 is 987602 and the 10's complement of 246700 is 753300.

Similarly, the 2's complement can be formed by leaving all least significant 0's and the first 1 unchanged and replacing 1's with 0's and 0's with 1's in all other higher significant digits. For example, the 2's complement of 1101100 is 0010100 and the 2's complement of 0110111 is 1001001.

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.

Using 10's complement, subtract 72532 − 3250.

$$\begin{aligned} M &= \quad 72532 \\ \text{10's complement of } N &= + \quad 96750 \\ \text{Sum} &= \quad 169282 \\ \text{Discard end carry } 10^5 &= - \quad 100000 \\ \textit{Answer} &= \quad 69282 \end{aligned}$$

Note that M has five digits and N has only four digits. Both numbers must have the same number of digits, so we write N as 03250. Taking the 10's complement of N produces a 9 in the most significant position. The occurrence of the end carry signifies that $M \geq N$ and that the result is therefore positive.

Using 10's complement, subtract 3250 − 72532.

$$\begin{aligned} M &= \quad 03250 \\ \text{10's complement of } N &= + \quad 27468 \\ \text{Sum} &= \quad 30718 \end{aligned}$$

There is no end carry. Therefore, the answer is −(10's complement of 30718) = −69282.

Note that since 3250 < 72532, the result is negative. Because we are dealing with unsigned numbers, there is really no way to get an unsigned result for this case. When subtracting with complements, we recognize the negative answer from the absence of the end carry and the complemented result. When working with paper and pencil, we can change the answer to a signed negative number in order to put it in a familiar form.

Subtraction with complements is done with binary numbers in a similar manner, using the procedure outlined previously.

Given the two binary numbers $X = 1010100$ and $Y = 1000011$, perform the subtraction **(a)** $X - Y$ and **(b)** $Y - X$ by using 2's complements.

(a)

$$\begin{aligned} X &= \quad 1010100 \\ \text{2's complement of } Y &= + \quad 0111101 \\ \text{Sum} &= \quad 10010001 \\ \text{Discard end carry } 2^7 &= - \quad 10000000 \\ \textit{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

(b)

$$\begin{aligned} Y &= \quad 1000011 \\ \text{2's complement of } X &= + \quad 0101100 \\ \text{Sum} &= \quad 1101111 \end{aligned}$$

There is no end carry. Therefore, the answer is $Y - X = -$(2's complement of 1101111) = −0010001.

Repeat Example 1.7, but this time using 1's complement.

(a) $X - Y = 1010100 - 1000011$

$$\begin{aligned} X &= \quad 1010100 \\ \text{1's complement of } Y &= + \quad 0111100 \\ \text{Sum} &= \quad 10010000 \\ \text{End-around carry} &= + \quad 1 \\ \textit{Answer: } X - Y &= \quad 0010001 \end{aligned}$$

(b) $Y - X = 1000011 - 1010100$

$$\begin{aligned} Y &= \quad 1000011 \\ \text{1's complement of } X &= + \quad 0101011 \\ \text{Sum} &= \quad 1101110 \end{aligned}$$

There is no end carry. Therefore, the answer is $Y - X = -$(1's complement of 1101110) = −0010001.

1.6) Signed Binary Numbers :

It is customary to represent the sign with a bit placed in the leftmost position of the number. The convention is to make the sign bit 0 for positive and 1 for negative.

For example, the string of bits 01001 can be considered as 9 (unsigned binary) or as +9 (signed binary) because the leftmost bit is 0. The string of bits 11001 represents the binary equivalent of 25 when considered as an unsigned number and the binary equivalent of -9 when considered as a signed number.

The representation of the signed numbers in the last example is referred to as the *signed-magnitude convention*. Whereas the *signed-magnitude system* negates a number by changing its sign, the *signed-complement system* negates a number by taking its complement.

Table 1.3
Signed Binary Numbers

Decimal	Signed-2's Complement	Signed-1's Complement	Signed Magnitude
+7	0111	0111	0111
+6	0110	0110	0110
+5	0101	0101	0101
+4	0100	0100	0100
+3	0011	0011	0011
+2	0010	0010	0010
+1	0001	0001	0001
+0	0000	0000	0000
−0	—	1111	1000
−1	1111	1110	1001
−2	1110	1101	1010
−3	1101	1100	1011
−4	1100	1011	1100
−5	1011	1010	1101
−6	1010	1001	1110
−7	1001	1000	1111
−8	1000	—	—

The range of n-bit 2's complement number = $-(2^{n-1})$ to $+(2^{n-1}-1)$. (0)
The range of n-bit sign magnitude number = $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$. (-0, +0)
The range of n-bit 1's complement number = $-(2^{n-1}-1)$ to $+(2^{n-1}-1)$. (-0, +0)

+ 6	00000110	− 6	11111010
+13	00001101	+13	00001101
+19	00010011	+ 7	00000111

+ 6	00000110	− 6	11111010
−13	11110011	−13	11110011
− 7	11111001	−19	11101101

Arithmetic Addition :

The addition of two signed binary numbers with negative numbers represented in signed-2's-complement form is obtained from the addition of the two numbers, including their sign bits. A carry out of the sign-bit position is discarded.

Arithmetic Subtraction :

To see this, consider the subtraction (-6) - (-13) = +7. In binary with eight bits, this operation is written as (11111010 - 11110011). The subtraction is changed to addition by taking the 2's complement of the subtrahend (-13), giving (+13). In binary, this is 11111010 + 00001101 = 100000111. Removing the end carry, we obtain the correct answer: 00000111 (+7).

2's complement of 101101.1 = 010010.1, Thus in floating number also we start from right hand side.

Overflow rule for addition :

If 2 Two's Complement numbers are added, remember you do not have to convert number into 2's complement if it is given in 2's complement. Rule is given below : (C_{out} XOR C_{in} = 1 then overflow)

1 1 0 0 0	0 0 1 1 0	1 0 1 0 1
1 1 0 0 0	0 1 0 0 1	1 1 0 1 0
① 1 0 0 0 0	0 1 1 1 1	1 0 1 1 1 1
MSBs are same, no overflow	MSBs are same, no overflow	MSB of output is different, overflow occurred
Carry, not MSB		
0 1 0 0 0		
0 1 0 0 0		
1 0 0 0 0		
MSB of output is different overflow occurred		

1.7) Binary Codes :

BCD Addition : When the binary sum is equal to or less than 1001 (without a carry), the corresponding BCD digit is correct. However, when the binary sum is greater than or equal to 1010, the result is an invalid BCD digit. The addition of 6 = (0110)2 to the binary sum converts it to the correct digit and also produces a carry as required.

4	0100	4	0100	8	1000	BCD		1	1	
+5	+0101	+8	+1000	+9	1001		0001	1000	0100	184
9	1001	12	1100	17	10001		+0101	0111	0110	+576
			+0110		+0110	Binary sum	0111	10000	1010	
			10010		10111	Add 6		0110	0110	
						BCD sum	0111	0110	0000	760

Decimal Arithmetic : Consider the addition (+375) + (-240) = +135, done in the signed-complement system. The 9 in the leftmost position of the second number represents a minus, and 9760 is the 10's complement of 0240. The two numbers are added and the end carry is discarded to obtain +135.

0	375
+9	760
0	135

Other Decimal Codes :

Table 1.5
Four Different Binary Codes for the Decimal Digits

Decimal Digit	BCD 8421	2421	Excess-3	8, 4, −2, −1
0	0000	0000	0011	0000
1	0001	0001	0100	0111
2	0010	0010	0101	0110
3	0011	0011	0110	0101
4	0100	0100	0111	0100
5	0101	1011	1000	1011
6	0110	1100	1001	1010
7	0111	1101	1010	1001
8	1000	1110	1011	1000
9	1001	1111	1100	1111
	1010	0101	0000	0001
Unused bit combi-nations	1011	0110	0001	0010
	1100	0111	0010	0011
	1101	1000	1101	1100
	1110	1001	1110	1101
	1111	1010	1111	1110

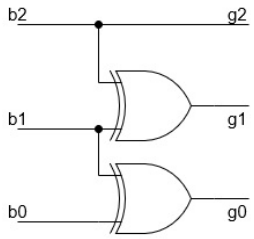
Table 1.6
Gray Code

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

Gray Code : For example, in going from 7 to 8, the Gray code changes from 0100 to 1100. Only the first bit changes, from 0 to 1; the other three bits remain the sam

e. By contrast, with binary numbers the change from 7 to 8 will be from 0111 to 1000, which causes all four bits to change values.

Binary to Grey Code Convertor :



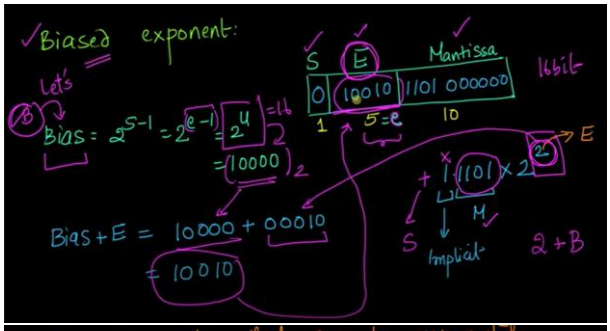
The 8421, 2421 and the excess-3 codes are examples of **self-complementing codes**. Such codes have the property that the 9's complement of a decimal number is obtained directly by changing 1's to 0's and 0's to 1's (i.e., by complementing each bit in the pattern). For example, decimal 395 is represented in the excess-3 code as 0110 1100 1000. The 9's complement of 604 is represented as 1001 0011 0111, which is obtained simply by complementing each bit of the code (as with the 1's complement of binary numbers). **Excess-3 is an unweighted code in which each coded combination is obtained from the corresponding binary value plus 3.**

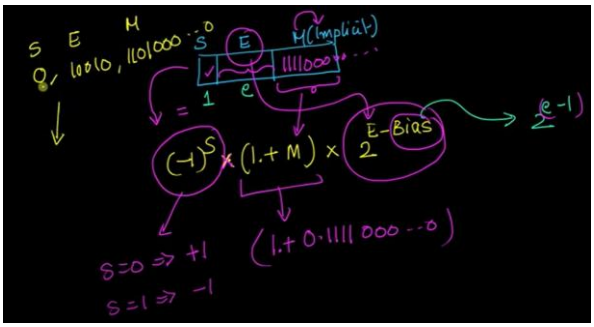
1.8) ***Register***: A *binary cell* is a device that possesses two stable states and is capable of storing one bit (0 or 1) of information. A *register* is a group of binary cells. A register with n cells can store any discrete quantity of information that contains n bits. The state of a register is an n-tuple of 1's and 0's, with each bit designating the state of one cell in the register. Consider, for example, a 16-bit register with the following binary content: 1100001111001001.

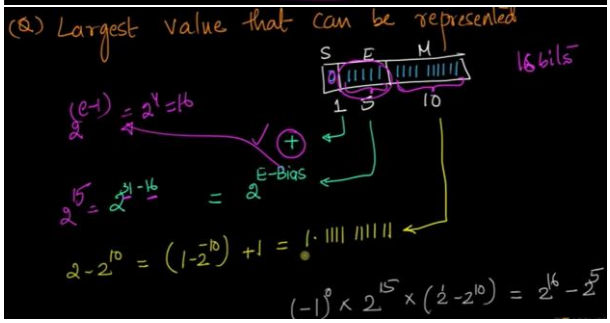
More on Number system form Applied gate :

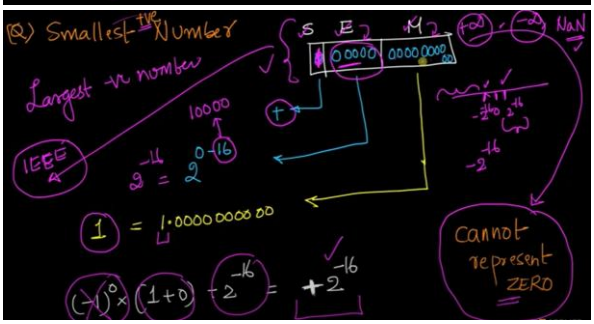
1) Floating point representation : **See carefully (1.+M) you need to add 1 to mantissa.**

1.1101 is implicit normalized mantissa representation and 0.11101 is explicit normalized mantissa representation.

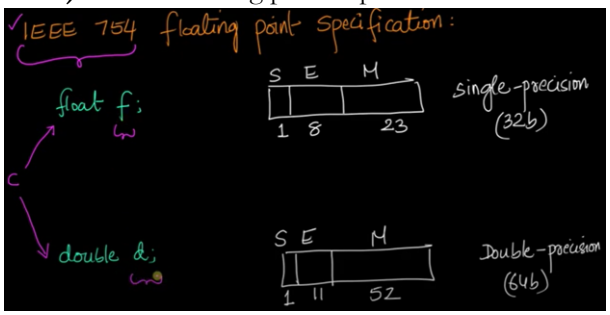








2) IEEE floating point representation :



S(1)	E(8)	M(23)	Value	Expr
0/1	0000 0000	000 ··· 0	±0	-
0/1	1111 1111	000 ··· 0	±∞	-
0/1	E≠0, E≠255	M=xxx··x	Implicit value	$(-1)^S \times (1.M)_2 \times 2^{E-127}$
0/1	E=0	M≠0	fractional	$(-1)^S \times (0.M)_2 \times 2^{-126}$
0/1	E=255	M≠0	NAN	

←imp

Here 127 is taken because its according to IEEE standard do not use $2^E - 1$.

IEEE 754 is excess 127. That is why adding 127 to every power, of two. And then convert into binary. now you can solve any type of problem.

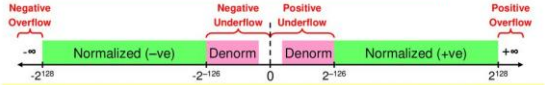
7.2 in IEEE formate is 402CCCCCH this is partially correct we use rounding and we make 402CCCCDH because we cannot form exact 7.2 by this method. So, first representation is actually 7.19 something so we have to add one at the end so by doing rounding we can have 7.2

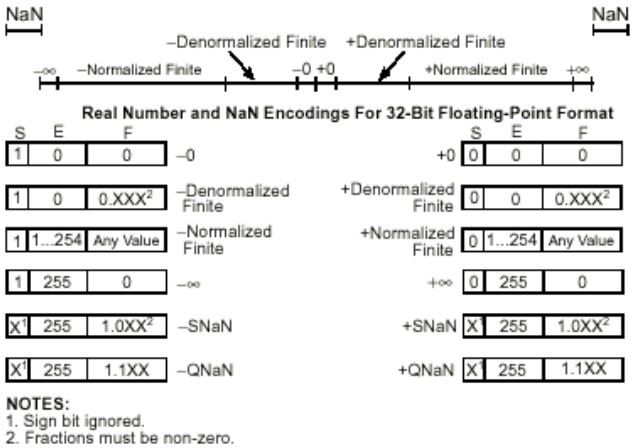
Denormalized means 0.mentisa and **normalized** means 1.mentisa

Denormalized Numbers

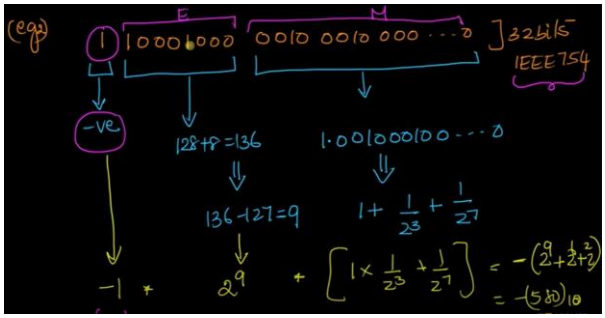
- IEEE standard uses denormalized numbers to …
  - Fill the gap between 0 and the smallest normalized float
  - Provide gradual underflow to zero
- Denormalized: exponent field E is 0 and fraction $F \neq 0$
  - Implicit 1. before the fraction now becomes 0. (not normalized)
- Value of denormalized number (S, 0, F)

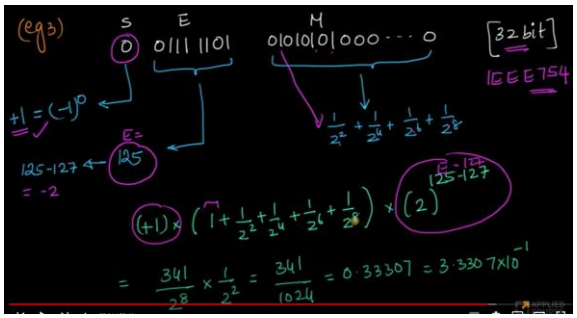
Single precision: $(-1)^S \times (0.F)_2 \times 2^{-126}$
Double precision: $(-1)^S \times (0.F)_2 \times 2^{-1022}$





←IMP

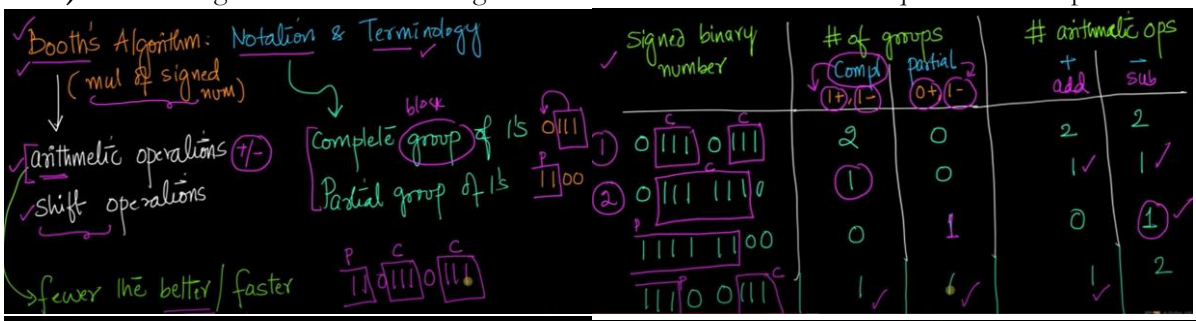


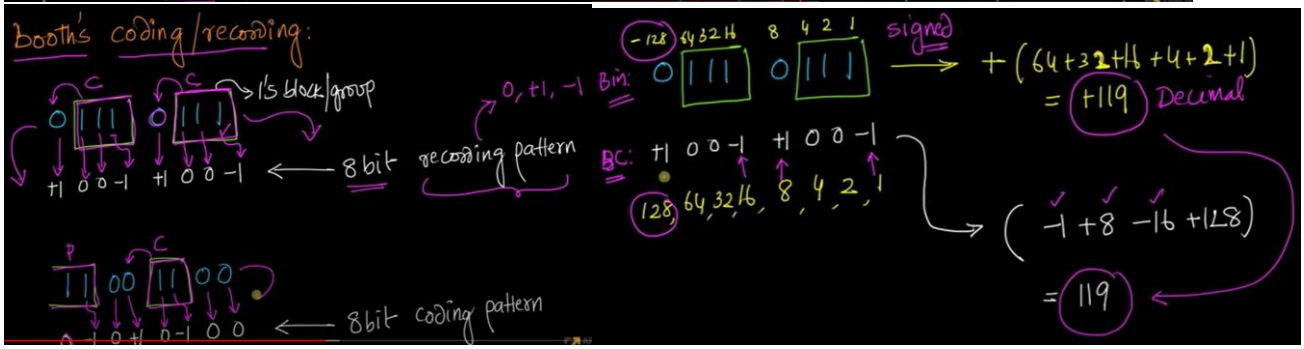


3) Addition and multiplication of signed and unsigned fixed-point numbers :

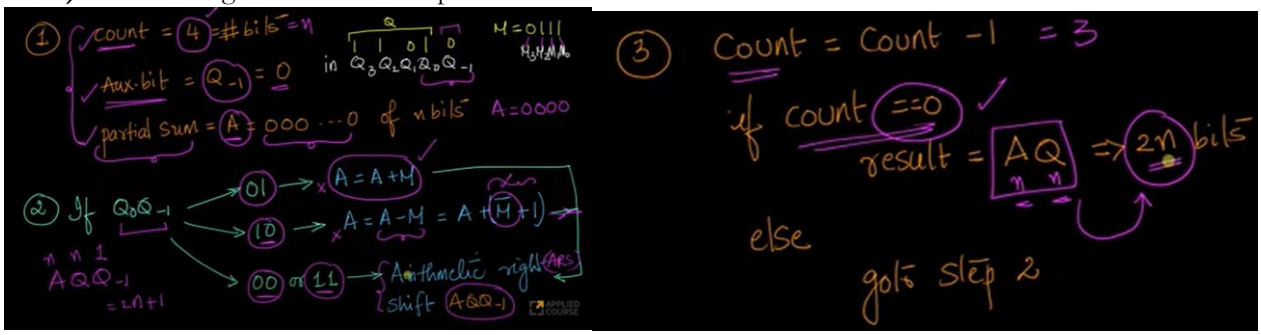
Convert any 2's complement number into decimal formate.

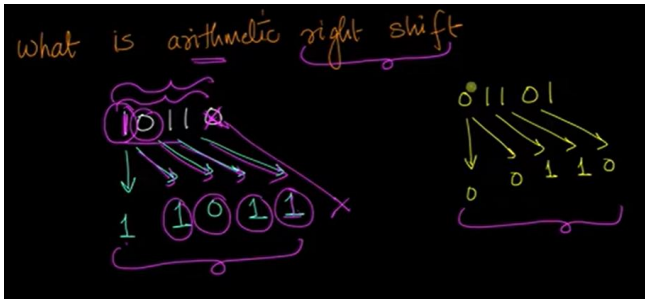
4) Booth's algorithms it's terminologies and notation : remember here sequence of bit represents multiplier.

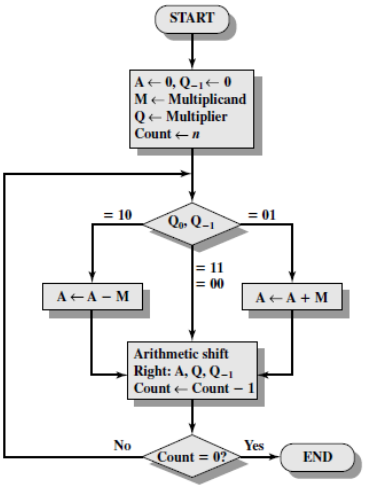




5) Booth's Algorithm with example :

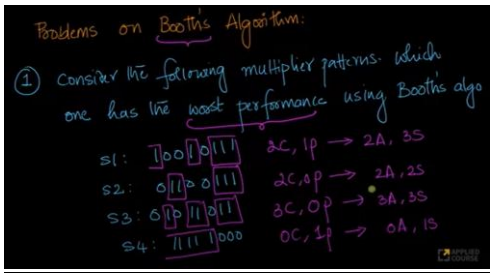


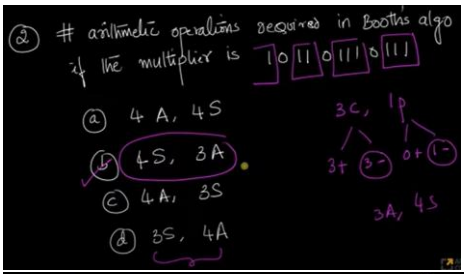


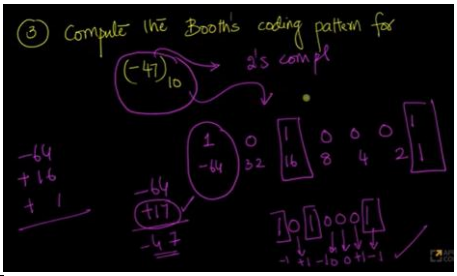


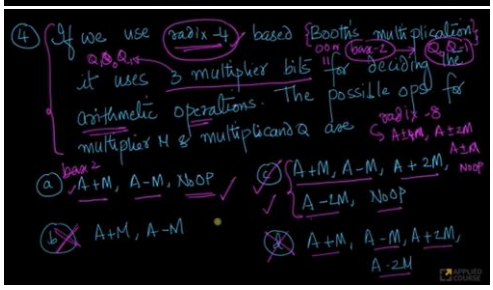
Now the choice of multiplicand and multiplier is not arbitrary so if you have two numbers. Write booth coding/sequence of both numbers. Number having minimum zeros in booth coding/sequence will be multiplier. The Booth's technique that helps reduce the number of summands to n / 2 for n-bit operands.

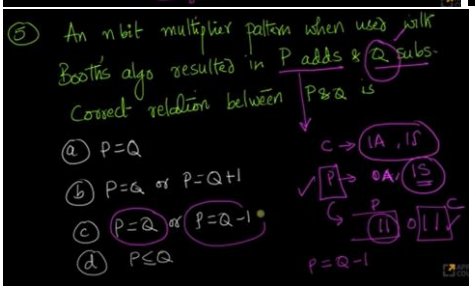
6) Solved example :











Question on Number System :

1) Consider an excess 50 - representation for floating point numbers with BCD digit mantissa and BCD digit exponent in normalised form. The minimum and maximum positive numbers that can be represented are _____ and _____ respectively.

Answer :

In binary we have normalized number of the form $(-1)^S \times 1.M \times 2^{E-\text{Bias}}$ where,

- S : sign bit
- M : Mantissa
- E : Exponent

Similarly for for Binary Coded Decimal (BCD) numbers the normalized number representation will be : $(-1)^S \times 1.M \times 10^{E-\text{Bias}}$

Here bias is given to be excess - 50 meaning that we need to subtract 50 from the base exponent field to get the actual exponent.

So, maximum mantissa value with 4 BCD digits $= 9999$

Maximum base exponent value with 2 BCD digits $= 99$

So, maximum actual exponent value possible with 2 BCD digits $=$ 99 - Bias

$= 99 - 50$

$= 49$

So, the magnitude of the largest positive number $= 9.9999 \times 10^{49}$

Similarly,

To get a minimum positive number, we have to set mantissa $= 0$ and exponent field $= 0$

So, doing that we get exponent $= 0 - 50 = -50$

So, magnitude of minimum positive number $= 1.0000 \times 10^{50}$

Therefore, maximum positive number $= 9.9999 \times 10^{49}$

Minimum positive number $= 1.0000 \times 10^{-50}$

2)

A 36 bit floating-point binary number has 8 bits plus a sign bit for exponent. The remaining bits are used to store mantissa along with a sign bit. The mantissa is store d as a normalized fraction (without leading 1). The negative numbers in the mantissa and the exponent are in signed-magnitude representation.

Format of this:

SM	SE	Exponent (8 bit)	Mantissa
1 bit	1 bit		26 bit

What are the largest and smallest positive quantities (excluding zero) that can be represented in the above system?

(a) $(2^{256}-2^{229})$ and 2^{-255}

(b) $(2^{256}-2^{229})$ and 1

(c) 2^{256} and 2^{-255}

(d) None of these

Solution:

Smallest quantity: -

0	1	11111111	00000000000000000000000000

Exponent =11111111 =255

=-255, because exponent bit is 1.

So smallest positive quantity excluding zero.

$=2^{-255}$

Largest quantity

0	0	11111111	11111111111111111111111111

$(2^{27}-1) * 2^{-26} * 2^{255}$

$(2^{27}-1) * 2^{229}$

$(2^{256} - 2^{229})$

Answer is A

2. BOOLEAN ALGEBRA AND LOGIC GATES

2.1) Basic Definitions :

Boolean algebra, like any other deductive mathematical system, may be defined with a set of elements, a set of operators, and a number of unproved axioms or postulates. A *binary operator* defined on a set S of elements is a rule that assigns, to each pair of elements from S, a unique element from S. As an example, consider the relation a * b = c. We say that * is a binary operator if it specifies a rule for finding c from the pair (a, b) and also if a, b, c Є S. However, * is not a binary operator if a, b Є S, and if c € S. The most common postulates used to formulate various algebraic structures are as follows:

1. *Closure*. A set S is closed with respect to a binary operator if, for every pair of elements of S, the binary operator specifies a rule for obtaining a unique element of S. For example, the set of natural numbers N = {1, 2, 3, 4, …} is closed with respect to the binary operator + by the rules of arithmetic addition.
2. *Associative law*. A binary operator * on a set S is said to be associative whenever (x * y) * z = x * (y * z) for all x, y, z, Є S
3. *Commutative law*. A binary operator * on a set S is said to be commutative whenever x * y = y * x for all x, y Є S and many more…

The operator precedence for evaluating Boolean expressions is (1) parentheses, (2) NOT, (3) AND, and (4) OR

Algebraic Manipulation :

We define a literal to be a single variable within a term, in complemented or uncomplemented form.(x, x', y, z',…) Dual of function is obtained by changing AND to OR and vice versa.

Simplify the following Boolean functions to a minimum number of literals.

1. $x(x' + y) = xx' + xy = 0 + xy = xy.$
2. $x + x'y = (x + x')(x + y) = 1(x + y) = x + y.$
3. $(x + y)(x + y') = x + xy + xy' + yy' = x(1 + y + y') = x.$
4. $xy + x'z + yz = xy + x'z + yz(x + x')$
   $= xy + x'z + xyz + x'yz$
   $= xy(1 + z) + x'z(1 + y)$
   $= xy + x'z.$
5. $(x + y)(x' + z)(y + z) = (x + y)(x' + z)$, by duality from function 4.

Assume function f has values $(a_0, a_1, \ldots, a_m)$, where $m = 2^n - 1$.

Then function f^d will have values $(\overline{a_m}, \overline{a_{m-1}}, \ldots, \overline{a_0})$

And function f^c will have values $(\overline{a_0}, \overline{a_1}, \ldots, \overline{a_m})$

Complement of a Function :

Find the complement of the functions $F_1 = x'yz' + x'y'z$ and $F_2 = x(y'z' + yz)$. By applying DeMorgan's theorems as many times as necessary, the complements are obtained as follows:

$$F_1' = (x'yz' + x'y'z)' = (x'yz')'(x'y'z)' = (x + y' + z)(x + y + z')$$
$$F_2' = [x(y'z' + yz)]' = x' + (y'z' + yz)' = x' + (y'z')'(yz)'$$
$$= x' + (y + z)(y' + z')$$
$$= x' + yz' + y'z$$

A simpler procedure for deriving the complement of a function is to take the dual of the function and complement each literal. This method follows from the generalized forms of DeMorgan's theorems. Remember that the dual of a function is obtained from the interchange of AND and OR operators and 1's and 0's.

Find the complement of the functions F_1 and F_2 of Example 2.2 by taking their duals and complementing each literal.

1. $F_1 = x'yz' + x'y'z.$
   The dual of F_1 is $(x' + y + z')(x' + y' + z).$
   Complement each literal: $(x + y' + z)(x + y + z') = F_1'.$
2. $F_2 = x(y'z' + yz).$
   The dual of F_2 is $x + (y' + z')(y + z).$
   Complement each literal: $x' + (y + z)(y' + z') = F_2'.$

2.2) Canonical Form and Standard Form :

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*. In a similar manner, *n variables can be combined to form 2^n minterms*.

In a similar fashion, n variables forming an OR term, with each variable being primed or unprimed, provide 2^n possible combinations, called *maxterms, or standard sums*.

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, the function f1 in Table 2.4 is determined by expressing the combinations 001, 100, and 111 as x'y'z, xy'z', and xyz, respectively.

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

Sum of Minterms :

for n binary variables, one can obtain 2^n distinct minterms and that any Boolean function can be expressed as a sum of minterms.

An alternative procedure for deriving the minterms of a Boolean function is to obtain the truth table of the function directly from the algebraic expression and then read the minterms from the truth table.

Express the Boolean function $F = A + B'C$ as a sum of minterms. The function has three variables: A, B, and C. The first term A is missing two variables; therefore,

$$A = A(B + B') = AB + AB'$$

This function is still missing one variable, so

$$A = AB(C + C') + AB'(C + C')$$
$$= ABC + ABC' + AB'C + AB'C'$$

The second term $B'C$ is missing one variable; hence,

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

Combining all terms, we have

$$F = A + B'C$$
$$= ABC + ABC' + AB'C + AB'C' + A'B'C$$

But $AB'C$ appears twice, and according to theorem 1 ($x + x = x$), it is possible to remove one of those occurrences. Rearranging the minterms in ascending order, we finally obtain

$$F = A'B'C + AB'C + AB'C + ABC' + ABC$$
$$= m_1 + m_4 + m_5 + m_6 + m_7$$

When a Boolean function is in its sum-of-minterms form, it is sometimes convenient to express the function in the following brief notation:

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

Table 2.5
Truth Table for $F = A + B'C$

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

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 of maxterms. 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). Then any missing variable x in each OR term is ORed with xx'. The procedure is clarified in the following example.

Express the Boolean function $F = xy + x'z$ as a product of maxterms. First, convert the function into OR terms by using the distributive law:

$$F = xy + x'z = (xy + x')(xy + z)$$
$$= (x + x')(y + x')(x + z)(y + z)$$
$$= (x' + y)(x + z)(y + z)$$

The function has three variables: x, y, and z. Each OR term is missing one variable; therefore,

$$x' + y = x' + y + zz' = (x' + y + z)(x' + y + z')$$
$$x + z = x + z + yy' = (x + y + z)(x + y' + z)$$
$$y + z = y + z + xx' = (x + y + z)(x' + y + z)$$

Combining all the terms and removing those which appear more than once, we finally obtain

$$F = (x + y + z)(x + y' + z)(x' + y + z)(x' + y + z')$$
$$= M_0M_2M_4M_5$$

A convenient way to express this function is as follows:

$$F(x, y, z) = \Pi(0, 2, 4, 5)$$

The product symbol, Π, denotes the ANDing of maxterms; the numbers are the indices of the maxterms of the function.

The complement of a function expressed as the sum of minterms equals the sum of minterms missing from the original function. This is because the original function is expressed by those minterms which make the function equal to 1, whereas its complement is a 1 for those minterms for which the function is a 0. As an example, consider the function

$$F(A, B, C) = \Sigma(1, 4, 5, 6, 7)$$

This function has a complement that can be expressed as

$$F'(A, B, C) = \Sigma(0, 2, 3) = m_0 + m_2 + m_3$$

Now, if we take the complement of F' by DeMorgan's theorem, we obtain F in a different form:

$$F = (m_0 + m_2 + m_3)' = m_0' \cdot m_2' \cdot m_3' = M_0M_2M_3 = \Pi(0, 2, 3)$$

The last conversion follows from the definition of minterms and maxterms as shown in Table 2.3. From the table, it is clear that the following relation holds:

$$m_j' = M_j$$

That is, the **maxterm with subscript j is a complement of the minterm with the same subscript j and vice versa.**

Table 2.6
Truth Table for $F = xy + x'z$

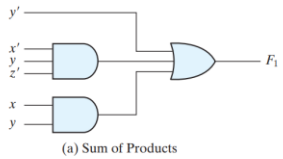
x	y	z	F
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	1
1	1	1	1

Minterms

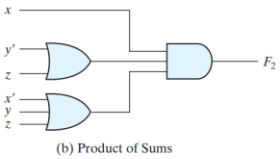
Maxterms

(A+A'B) = (A+A')(A+B) = (A+B)

Standard Forms : In this configuration, the terms that form the function may contain one, two, or any number of literals. There are two types of standard forms: the sum of products and products of sums. The sum of products is a Boolean expression containing AND terms, called product terms, with one or more literals each. The sum denotes the ORing of these terms.

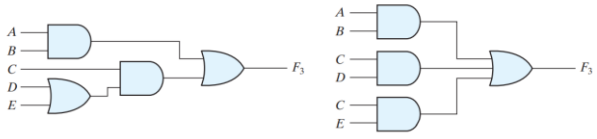


(a) Sum of Products



(b) Product of Sums

FIGURE 2.3
Two-level implementation



(a) $AB + C(D + E)$ (b) $AB + CD + CE$

FIGURE 2.4
Three- and two-level implementation

2.3) Other Logical Operators : Previously we stated that there are 2^{2^n} functions for n binary variables. Thus, for two variables, n = 2, and the number of possible Boolean functions is 16. Therefore, the AND and OR functions are only 2 of a total of 16 possible functions formed with two binary variables.

Table 2.8
Boolean Expressions for the 16 Functions of Two Variables

Boolean Functions	Operator Symbol	Name	Comments
$F_0 = 0$		Null	Binary constant 0
$F_1 = xy$	$x \cdot y$	AND	x and y
$F_2 = xy'$	x/y	Inhibition	x, but not y
$F_3 = x$		Transfer	x
$F_4 = x'y$	y/x	Inhibition	y, but not x
$F_5 = y$		Transfer	y
$F_6 = xy' + x'y$	$x \oplus y$	Exclusive-OR	x or y, but not both
$F_7 = x + y$	$x + y$	OR	x or y
$F_8 = (x + y)'$	$x \downarrow y$	NOR	Not-OR
$F_9 = xy + x'y'$	$(x \oplus y)'$	Equivalence	x equals y
$F_{10} = y'$	y'	Complement	Not y
$F_{11} = x + y'$	$x \subset y$	Implication	If y, then x
$F_{12} = x'$	x'	Complement	Not x
$F_{13} = x' + y$	$x \supset y$	Implication	If x, then y
$F_{14} = (xy)'$	$x \uparrow y$	NAND	Not-AND
$F_{15} = 1$		Identity	Binary constant 1

Table 2.7
Truth Tables for the 16 Functions of Two Binary Variables

x	y	F_0	F_1	F_2	F_3	F_4	F_5	F_6	F_7	F_8	F_9	F_{10}	F_{11}	F_{12}	F_{13}	F_{14}	F_{15}
0	0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0	1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

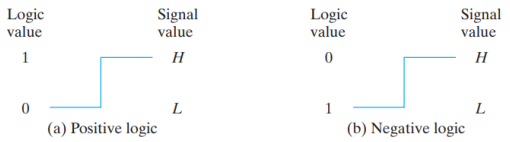


FIGURE 2.9
Signal assignment and logic polarity

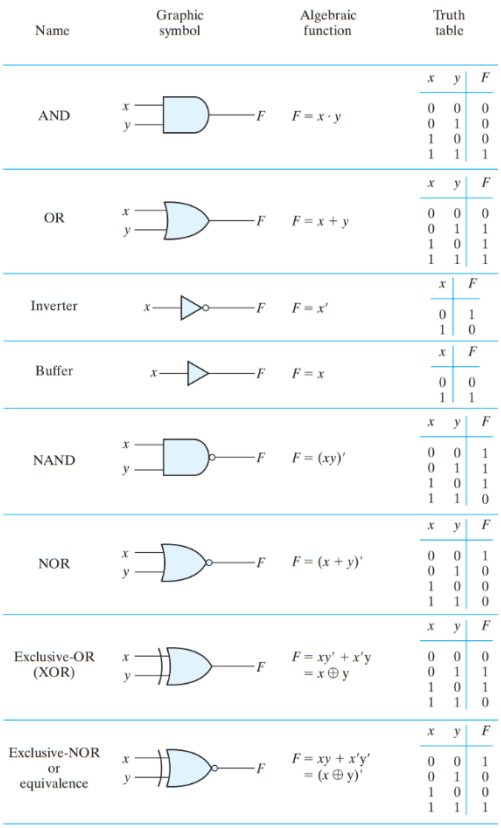


FIGURE 2.5
Digital logic gates

2.4) 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.

Levels of Integration :

Small-scale integration (SSI) devices contain several independent gates in a single package. The number of gates is usually fewer than 10 and is limited by the number of pins available in the IC.

Medium-scale integration (MSI) devices have a complexity of approximately 10 to 1,000 gates in a single package. They usually perform specific elementary digital operations. MSI digital functions are introduced in Chapter 4 as decoders, adders, and multiplexers and in Chapter 6 as registers and counters.

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.

Digital Logic Families :

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

TTL is a logic family that has been in use for 50 years and is considered to be standard. ECL has an advantage in systems requiring high-speed operation. MOS is suitable for circuits that need high component density, and CMOS is preferable in systems requiring low power consumption, such as digital cameras, personal media players, and other handheld portable devices.

Some Extra Definition :

Fan-out specifies the number of standard loads that the output of a typical gate can drive without impairing its normal operation.
Fan-in is the number of inputs available in a gate.
Power dissipation is the power consumed by the gate that must be available from the power supply.
Propagation delay is the average transition delay time for a signal to propagate from input to output.
Noise margin is the maximum external noise voltage added to an input signal that does not cause an undesirable change in the circuit output.

2.5) Functional Completeness :

Definition : A system S of Boolean functions (or, alternatively, logical operators) is functionally complete if every Boolean function (or, alternatively, every compound proposition) can be expressed in terms of the functions from S.

First, let us describe those five properties of boolean functions :

- **Property 1**: We say that boolean **function f preserves zero**, if on the 0-input it produces 0. By the 0-input we mean such an input, where every input variable is 0 (this input usually corresponds to the first row of the truth table). We denote the class of zero-preserving boolean functions as T0 and write f ∈ T0.

x	$\neg x$
0	1
1	0

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

We see that on 0-input, both ∧ and ∨ produce 0, while ¬ produces 1. Thus, ∧ and ∨ preserve zero, and ¬ does not. We write ∧ ∈ T0 and ∨ ∈ T0.

- **Property 2**: Similarly to T0, we say that boolean **function f preserves one**, if on 1-input, it produces 1. The 1-input is the input where all the input variables are 1 (this input usually corresponds to the last row of the truth table). We denote the class of one-preserving boolean functions as T1 and write f ∈ T1.

x	$\neg x$
0	1
1	0

x_1	x_2	$x_1 \wedge x_2$
0	0	0
0	1	0
1	0	0
1	1	1

x_1	x_2	$x_1 \vee x_2$
0	0	0
0	1	1
1	0	1
1	1	1

Thus, ∧ ∈ T1 and ∨ ∈ T1.

- **Property 3**: We say that boolean function f is **linear** <u>if one of</u> the following two statements holds for f:

⇨ For every 1-value of f, the number of 1's in the corresponding input is odd, and for every 0-value of f, the number of 1's in the corresponding input is even. **or**

⇨ For every 1-value of f, the number of 1's in the corresponding input is even, and for every 0-value of f, the number of 1's in the corresponding input is odd.

In the case of ¬, for every 1-value of this function, the number of 1's among the input variables is always even. ∧ is not linear because on 0 outputs, the number of 1's in the corresponding inputs is sometimes even and sometimes odd. Thus, neither of two conditions from our definition can hold for ∧. Using the same reasoning, we can show that neither ∧ nor ∨ is linear. Thus, ¬ ∈ L.

- **Property 4**: Property 4: We say that boolean function f is **monotone** if for every input, switching any input variable from 0 to 1 can only result in the function's switching its value from 0 to 1, and never from 1 to 0. We denote the class of monotone boolean functions with M and write f ∈ M. We have ∧ ∈ M and ∨ ∈ M.
- **Property 5**: We say that boolean function f(x1, . . . , xn) is **self-dual** if f(x1, . . . , xn) = ¬f(¬x1, . . . , ¬xn). The function on the right in the equality above (the one with negations) is called the dual of f. We will call the class of self-dual boolean functions S and write f ∈ S.

The dual for ¬x is ¬(¬(¬x)) = ¬x. Thus, it is clear that ¬ is self-dual. The dual for x ∧ y is ¬((¬x) ∧ (¬y)), and the latter, according to one of De Morgan's laws is x ∨ y. It is clear that x ∧ y and its dual x ∨ y has different values on some inputs and, thus, are not equivalent. Thus, ∧ is not self-dual. Using the same reasoning, we can show that ∨ is also not self-dual. Thus, ¬ ∈ S.

Theorem. A system of boolean functions is functionally complete if and only if this system does not entirely belong to any of T0, T1, L, M, S.

According to the Emily post's condition A system of boolean functions to be functionally complete, this system should have

- at least one function that does not preserve zero (i.e. it is not in T0), and
- at least one function that does not preserve one (i.e. it is not in T1), and
- at least one function that is not linear (i.e. it is not in L), and
- at least one function that is not monotone (i.e. it is not in M), and
- at least one function that is not self-dual (i.e. it is not in S).

Example :

	not in T_0	not in T_1	not in L	not in M	not in S
$\wedge$	−	−	+	−	+
$\vee$	−	−	+	−	+
$\neg$	+	+	−	+	−

	not in T_0	not in T_1	not in L	not in M	not in S
$\wedge$	−	−	+	−	+
$\vee$	−	−	+	−	+

According to Post's criterion, for a system {∧, ∨, ¬} of boolean functions to be functionally complete. Every column of the table contains at least one +. The columns corresponding to classes T0, T1, and M have no +'es, which means that system {∧, ∨} is not functionally complete.

Questions on Boolean algebra :

1) Two numbers are chosen independently and uniformly at random from the set {1, 2, ..., 13}. The probability (rounded off to 3 decimal places) that their 4-bit (unsigned) binary representations have the same most significant bit is _____.

0.502
0.461
0.402
0.561

Answer : The probability that their 4-bit binary representations have the same most significant bit is
= P(MSB is 0) + P(MSB is 1) = (7×7)/(13×13) + (6×6)/(13×13) = (49+36)/169 = 85/169 = 0.502

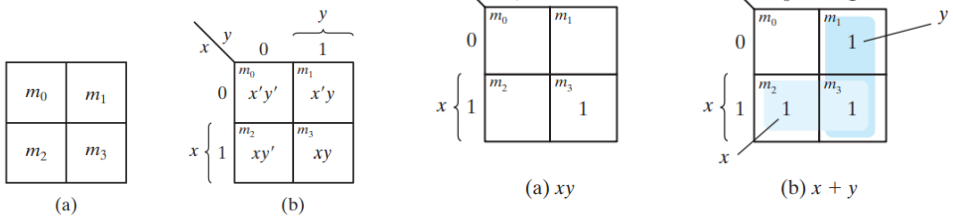
3. GATE LEVEL MINIMIZATION

Gate-level minimization is the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit.

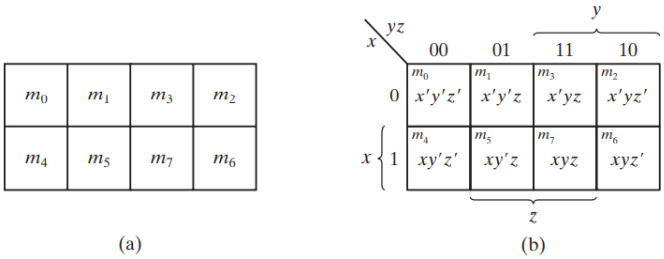
3.1) Map Method : This method may be regarded as a pictorial form of a truth table. The map method is also known as the Karnaugh map or K-map.

Two-Variable K-Map : There are four minterms for two variables; hence, the map consists of four squares, one for each minterm. The map is redrawn in (b) to show the relationship between the squares and the two variables x and y . The 0 and 1 marked in each row and column designate the values of variables. Variable x appears primed in row 0 and unprimed in row 1. Similarly, y appears primed in column 0 and unprimed in column 1.

$$m_1 + m_2 + m_3 = x'y + xy' + xy = x + y$$



Three-Variable K-Map :



Simplify the Boolean function

$$F(x, y, z) = \Sigma(3, 4, 6, 7)$$

The map for this function is shown in Fig. 3.5. There are four squares marked with 1's, one for each minterm of the function. Two adjacent squares are combined in the third column to give a two-literal term yz. The remaining two squares with 1's are also adjacent by the new definition. These two squares, when combined, give the two-literal term xz'. The simplified function then becomes

$$F = yz + xz'$$

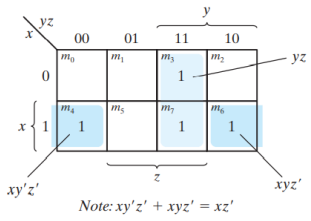


FIGURE 3.5
Map for Example 3.2, $F(x, y, z) = \Sigma(3, 4, 6, 7) = yz + xz'$

Simplify the Boolean function

$$F(x, y, z) = \Sigma(0, 2, 4, 5, 6)$$

The map for F is shown in Fig. 3.6. First, we combine the four adjacent squares in the first and last columns to give the single literal term z'. The remaining single square, representing minterm 5, is combined with an adjacent square that has already been used once. This is not only permissible, but rather desirable, because the two adjacent squares give the two-literal term xy' and the single square represents the three-literal minterm $xy'z$. The simplified function is

$$F = z' + xy'$$

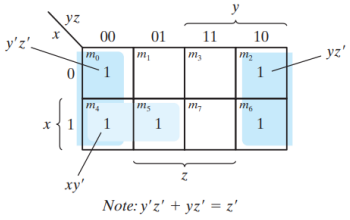


FIGURE 3.6
Map for Example 3.3, $F(x, y, z) = \Sigma(0, 2, 4, 5, 6) = z' + xy'$

Example :

Simplify the Boolean function

$$F(x, y, z) = \Sigma(2, 3, 4, 5)$$

First, a 1 is marked in each minterm square that represents the function. This is shown in Fig. 3.4, in which the squares for minterms 010, 011, 100, and 101 are marked with 1's. The next step is to find possible adjacent squares. These are indicated in the map by two shaded rectangles, each enclosing two 1's. The upper right rectangle represents the area enclosed by $x'y$. This area is determined by observing that the two-square area is in row 0, corresponding to x', and the last two columns, corresponding to y. Similarly, the lower left rectangle represents the product term xy'. (The second row represents x and the two left columns represent y'.) The sum of four minterms can be replaced by a sum of

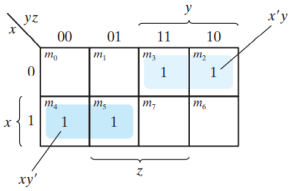


FIGURE 3.4
Map for Example 3.1, $F(x, y, z) = \Sigma(2, 3, 4, 5) = x'y + xy'$

For the Boolean function

$$F = A'C + A'B + AB'C + BC$$

(a) Express this function as a sum of minterms.
(b) Find the minimal sum-of-products expression.

$$F(A, B, C) = \Sigma(1, 2, 3, 5, 7)$$

The sum-of-products expression, as originally given, has too many terms. It can be simplified, as shown in the map, to an expression with only two terms:

$$F = C + A'B$$

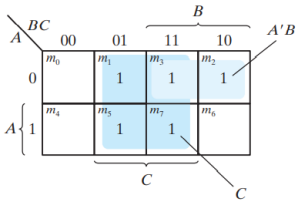
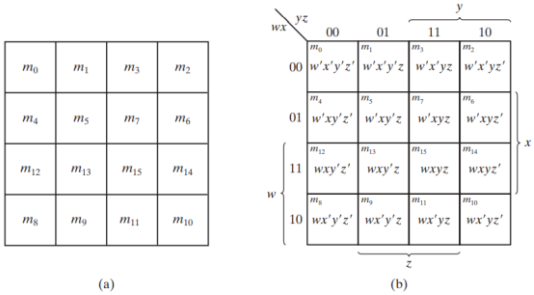


FIGURE 3.7
Map of Example 3.4, $A'C + A'B + AB'C + BC = C + A'B$

3.2) Four variable K-Map :



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

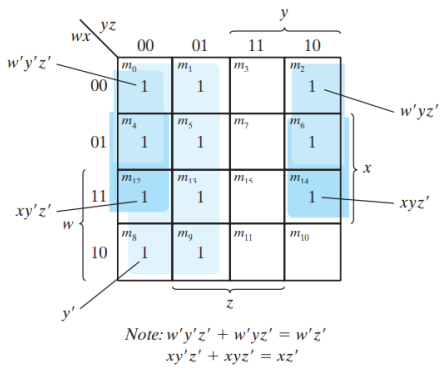


FIGURE 3.9
Map for Example 3.5, $F(w, x, y, z) = \Sigma(0,1,2,4,5,6,8,9,12,13,14) = y' + w'z' + xz'$

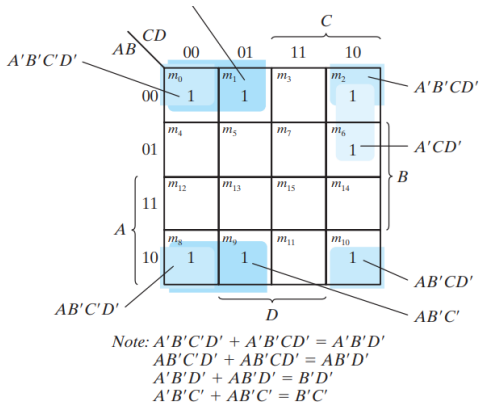
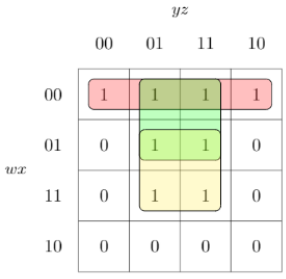
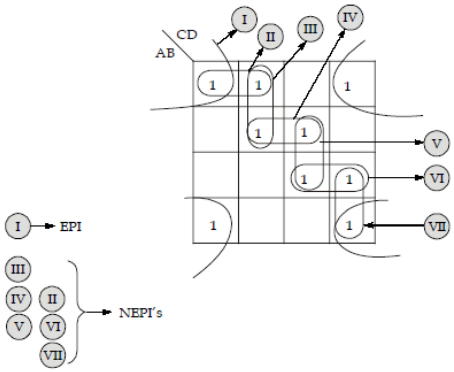


FIGURE 3.10
Map for Example 3.6, $A'B'C' + B'CD' + A'BCD' + AB'C' = B'D' + B'C' + A'CD'$

Prime Implicants : A *prime implicant* is a product term obtained by combining the maximum possible number of adjacent squares in the map. If a minterm in a square is covered by only one prime implicant, that prime implicant is said to be *essential.* The prime implicants of a function can be obtained from the map by combining all possible maximum numbers of squares. This means that a single 1 on a map represents a prime implicant if it is not adjacent to any other 1's. Two adjacent 1's form a prime implicant, provided that they are not within a group of four adjacent squares. Four adjacent 1's form a prime implicant if they are not within a group of eight adjacent squares, and so on. The essential prime implicants are found by looking at each square marked with a 1 and checking the number of prime implicants that cover it. *The prime implicant is essential if it is the only prime implicant that covers the minterm. If square is covered do not try to breck it down. NEPI should also cover all don't care terms.*



EPI = Essential Prime Implicant
NEPI = Non Essential Prime Implicant



Now, we can count the number of implicants:

- The number of implicants of size $2^0 = 8$
- The number of implicants of size $2^1 = 10$
- The number of implicants of size $2^2 = 3$

$\therefore$ The total number of implicants $= 8 + 10 + 3 = 21$.

Prime implicants $= 3$, and essential prime implicants $= 2$.

Five-Variable Map : In general, *rows and column in n variable K-map is* $2^{\lfloor n/2 \rfloor}$ *and* $2^{\lceil n/2 \rceil}$ *respectively.* A five-variable map needs 32 squares and a six-variable map needs 64 squares.

3.3) Sum of product simplification :

Simplify the following Boolean function into (a) sum-of-products form and (b) product-of-sums form:

$$F(A, B, C, D) = \Sigma(0, 1, 2, 5, 8, 9, 10)$$

The 1's marked in the map of Fig. 3.12 represent all the minterms of the function. The squares marked with 0's represent the minterms not included in F and therefore denote the complement of F. Combining the squares with 1's gives the simplified function in sum-of-products form:

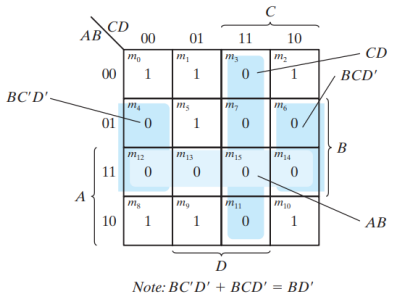
(a) $F = B'D' + B'C' + A'C'D$

If the squares marked with 0's are combined, as shown in the diagram, we obtain the simplified complemented function:

$$F' = AB + CD + BD'$$

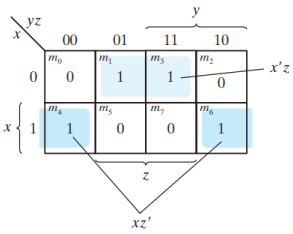
Applying DeMorgan's theorem (by taking the dual and complementing each literal as described in Section 2.4), we obtain the simplified function in product-of-sums form:

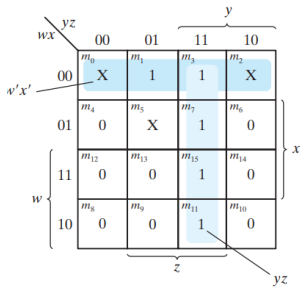
(b) $F = (A' + B')(C' + D')(B' + D)$



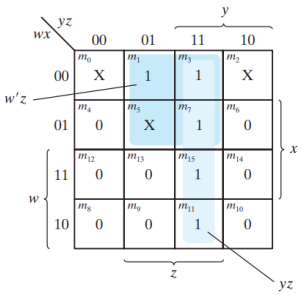
Note: $BC'D' + BCD' = BD'$

FIGURE 3.12
Map for Example 3.7, $F(A, B, C, D) = \Sigma(0,1, 2, 5, 8, 9,10) = B'D' + B'C' + A'C'D = (A' + B')(C' + D')(B' + D)$





(a) $F = yz + w'x'$



(b) $F = yz + w'z$

FIGURE 3.15
Example with don't-care conditions

← Try this.

Consensus Theorem : AB + A'C + BC = AB +A'C … Draw k-map

3.4) Don't care condition : Functions that have unspecified outputs for some input combinations are called *incompletely specified functions* . In most applications, we simply *don't care* what value is assumed by the function for the unspecified minterms. For this reason, it is customary to call the unspecified minterms of a function *don't-care conditions*.

Simplify the Boolean function

$$F(w, x, y, z) = \Sigma(1, 3, 7, 11, 15)$$

which has the don't-care conditions

$$d(w, x, y, z) = \Sigma(0, 2, 5)$$

3.5) NAND and NOR implementation :

NAND Circuits : Complement can be obtained with NAND gates alone.

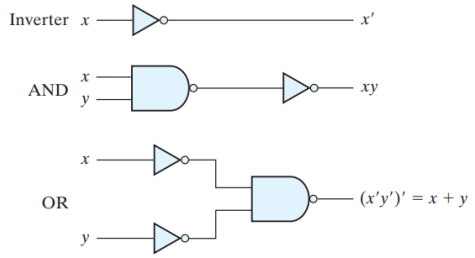
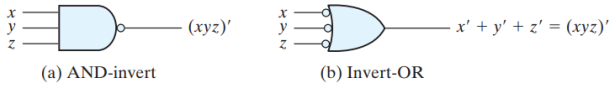
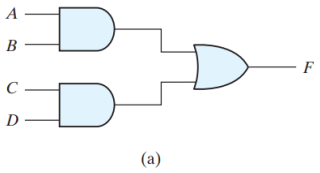


FIGURE 3.16
Logic operations with NAND gates

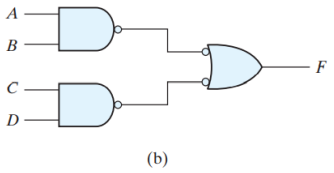


(a) AND-invert (b) Invert-OR

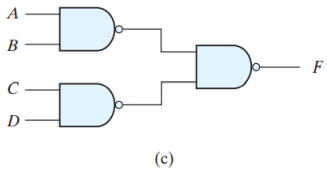
FIGURE 3.17
Two graphic symbols for a three-input NAND gate



(a)



(b)



(c)

FIGURE 3.18
Three ways to implement $F = AB + CD$

Two-Level Implementation : The implementation of Boolean functions with NAND gates requires that the functions be in sum-of-products form. Fig 3.18, 3.19

Example : Implement the following Boolean function with NAND gates: F (x, y, z) = (1, 2, 3, 4, 5, 7)

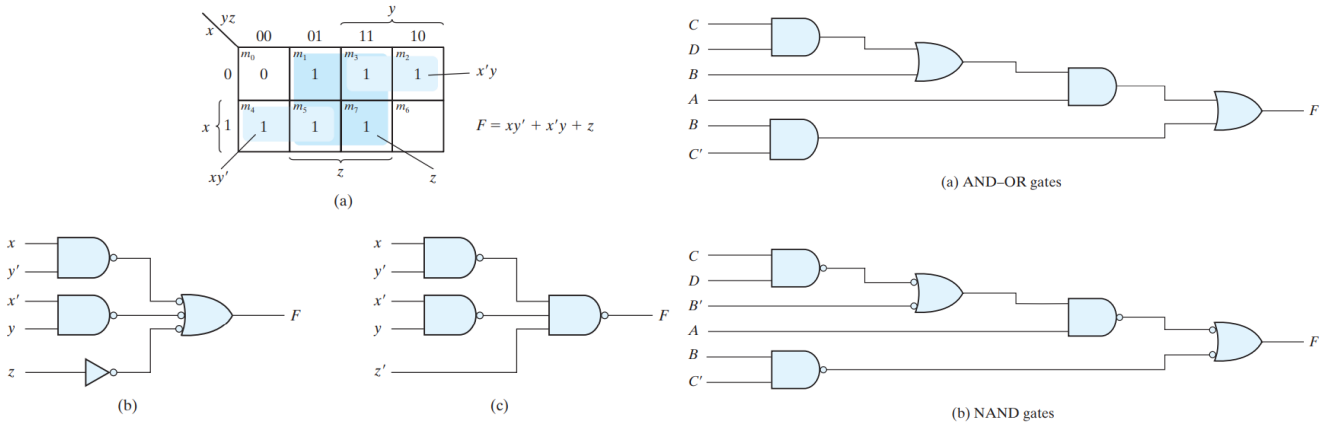


FIGURE 3.19
Solution to Example 3.9

FIGURE 3.20
Implementing $F = A(CD + B) + BC'$

Multilevel NAND Circuits : figure 3.20. In general

1. Convert all AND gates to NAND gates with AND-invert graphic symbols.
2. Convert all OR gates to NAND gates with invert-OR graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NAND gate) or complement the input literal. Try F = (AB' + A'B)(C + D')

NOR Implementation :

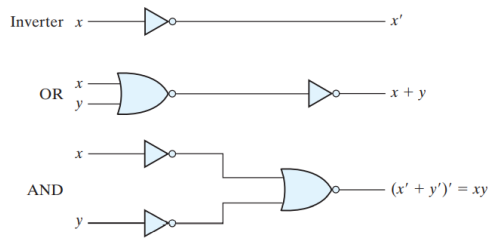


FIGURE 3.22
Logic operations with NOR gates

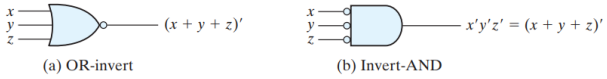


FIGURE 3.23
Two graphic symbols for the NOR gate

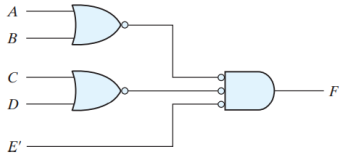


FIGURE 3.24
Implementing $F = (A + B)(C + D)E$

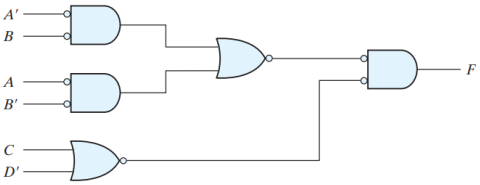


FIGURE 3.25
Implementing $F = (AB' + A'B)(C + D')$ with NOR gates

In general

1. Convert all OR gates to NOR gates with OR-invert graphic symbols.
2. Convert all AND gates to NOR gates with invert-AND graphic symbols.
3. Check all the bubbles in the diagram. For every bubble that is not compensated by another small circle along the same line, insert an inverter (a one-input NOR gate) or complement the input literal.

3.6) Exclusive OR Function :

The exclusive-OR (XOR), denoted by the symbol Θ, is a logical operation that performs the following Boolean operation: x Θ y = xy' + x'y The exclusive-OR is equal to 1 if only x is equal to 1 or if only y is equal to 1 (i.e., x and y differ in value), but not when both are equal to 1 or when both are equal to 0. The exclusive NOR, also known as equivalence, performs the following Boolean operation: (x Θ y)' = xy + x'y' The exclusive-NOR is equal to 1 if both x and y are equal to 1 or if both are equal to 0.

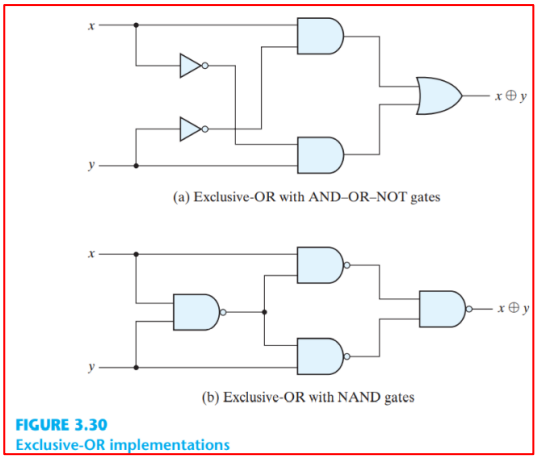
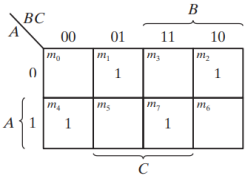


FIGURE 3.30
Exclusive-OR implementations

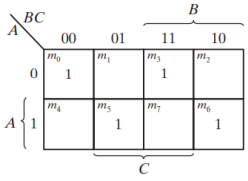
Odd Function :

$$A \oplus B \oplus C = (AB' + A'B)C' + (AB + A'B')C$$
$$= AB'C' + A'BC' + ABC + A'B'C$$
$$= \Sigma(1, 2, 4, 7)$$

In general, an n -variable exclusive-OR function is an odd function defined as the logical sum of the 2^n /2 minterms whose binary numerical values have an odd number of 1's.

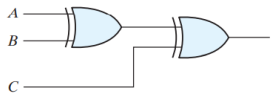


(a) Odd function $F = A \oplus B \oplus C$

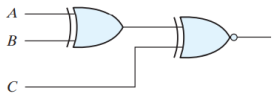


(b) Even function $F = (A \oplus B \oplus C)'$

FIGURE 3.31
Map for a three-variable exclusive-OR function

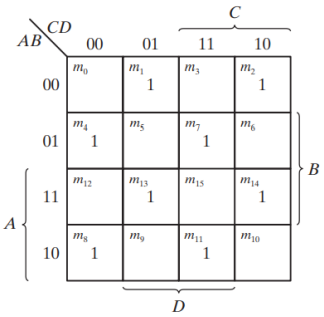


(a) 3-input odd function

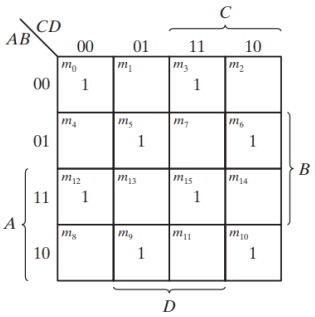


(b) 3-input even function

FIGURE 3.32
Logic diagram of odd and even functions



(a) Odd function $F = A \oplus B \oplus C \oplus D$



(b) Even function $F = (A \oplus B \oplus C \oplus D)'$

FIGURE 3.33
Map for a four-variable exclusive-OR function

1. ***Parity Generation and Checking :*** The circuit that generates the parity bit in the transmitter is called a parity generator. The circuit that checks the parity in the receiver is called a parity checker.

NOTE :

1) The prime implicant whose 1 covered by at least one EPI is called **redundant prime implicant**. And a prime implicant which is neither essential nor redundant is called **selective prime implicant**.
2) Any essential prime implicant cannot be split into two different PI.
3) You can also solve question of no of NAND/NOR gate requires by taking negation of expression.
4) Surprisingly, **A⊕B⊕C=A⊙B⊙C**, you can check

4. Combinational Circuits

4.1) Introduction :

Logic circuits for digital systems may be combinational or sequential. A combinational circuit consists of logic gates whose outputs at any time are determined from only the present combination of inputs. In contrast, sequential circuits employ storage elements in addition to logic gates. Their outputs are a function of the inputs and the state of the storage elements. Because the state of the storage elements is a function of previous inputs, the outputs of a sequential circuit depend not only on present values of inputs, but also on past inputs, and the circuit behaviour must be specified by a time sequence of inputs and internal states.

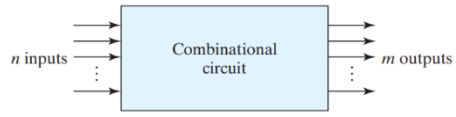


FIGURE 4.1
Block diagram of combinational circuit

For n-input variables, there are 2^n possible combinations of the binary inputs. For each possible input combination, there is one possible value for each output variable.

4.2) Analysis of Combinational circuit :

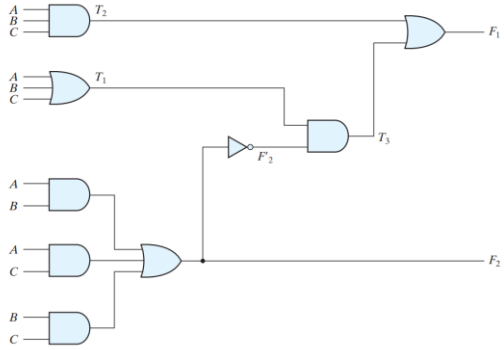


FIGURE 4.2
Logic diagram for analysis example

The analysis of the combinational circuit of Fig. 4.2 illustrates the proposed procedure. We note that the circuit has three binary inputs—A, B, and C—and two binary outputs—F_1 and F_2. The outputs of various gates are labeled with intermediate symbols. The outputs of gates that are a function only of input variables are T_1 and T_2. Output F_2 can easily be derived from the input variables. The Boolean functions for these three outputs are

$$F_2 = AB + AC + BC$$
$$T_1 = A + B + C$$
$$T_2 = ABC$$

Next, we consider outputs of gates that are a function of already defined symbols:

$$T_3 = F'_2 T_1$$
$$F_1 = T_3 + T_2$$

To obtain F_1 as a function of A, B, and C, we form a series of substitutions as follows:

$$\begin{aligned} F_1 &= T_3 + T_2 = F'_2 T_1 + ABC = (AB + AC + BC)'(A + B + C) + ABC \\ &= (A' + B')(A' + C')(B' + C')(A + B + C) + ABC \\ &= (A' + B'C')(AB' + AC' + BC' + B'C) + ABC \\ &= A'BC' + A'B'C + AB'C' + ABC \end{aligned}$$

Table 4.1
Truth Table for the Logic Diagram of Fig. 4.2

A	B	C	F_2	F'_2	T_1	T_2	T_3	F_1
0	0	0	0	1	0	0	0	0
0	0	1	0	1	1	0	1	1
0	1	0	0	1	1	0	1	1
0	1	1	1	0	1	0	0	0
1	0	0	0	1	1	0	1	1
1	0	1	1	0	1	0	0	0
1	1	0	1	0	1	0	0	0
1	1	1	1	0	1	1	0	1

4.3) Design procedure :

A truth table for a combinational circuit consists of input columns and output columns. The input columns are obtained from the 2^n binary numbers for the n input variables. The binary values for the outputs are determined from the stated specifications.

Code Conversion Example :

It is sometimes necessary to use the output of one system as the input to another. A conversion circuit must be inserted between the two systems if each uses different codes for the same information. Thus, a *code converter* is a circuit that makes the two systems compatible even though each uses a different binary code. The design procedure will be illustrated by an example that converts binary coded decimal (BCD) to the excess-3 code for the decimal digits.

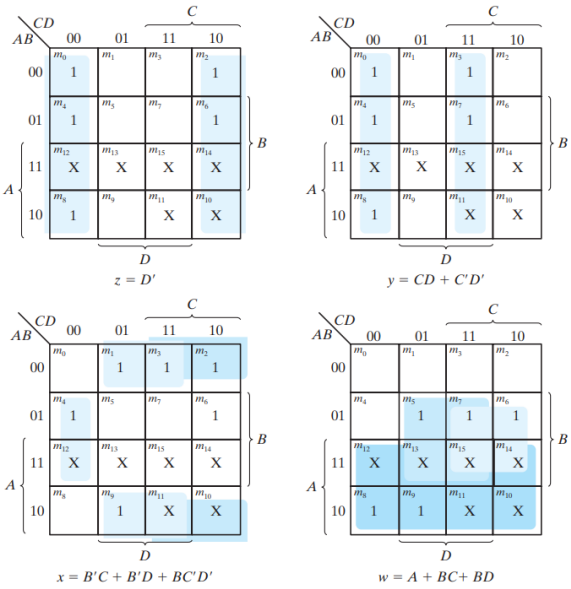


FIGURE 4.3
Maps for BCD-to-excess-3 code converter

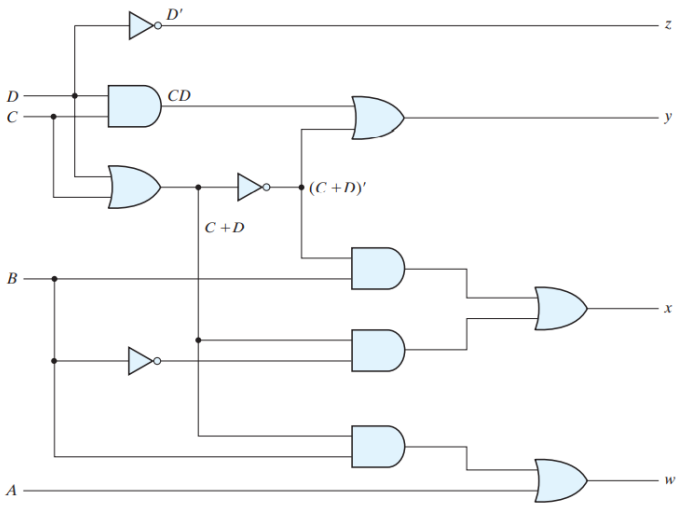


FIGURE 4.4
Logic diagram for BCD-to-excess-3 code converter

Table 4.2
Truth Table for Code Conversion Example

Input BCD				Output Excess-3 Code			
A	B	C	D	w	x	y	z
0	0	0	0	0	0	1	1
0	0	0	1	0	1	0	0
0	0	1	0	0	1	0	1
0	0	1	1	0	1	1	0
0	1	0	0	0	1	1	1
0	1	0	1	1	0	0	0
0	1	1	0	1	0	0	1
0	1	1	1	1	0	1	0
1	0	0	0	1	0	1	1
1	0	0	1	1	1	0	0

implemented with three or more levels of gates:

$$z = D'$$
$$y = CD + C'D' = CD + (C + D)'$$
$$x = B'C + B'D + BC'D' = B'(C + D) + BC'D'$$
$$= B'(C + D) + B(C + D)'$$
$$w = A + BC + BD = A + B(C + D)$$

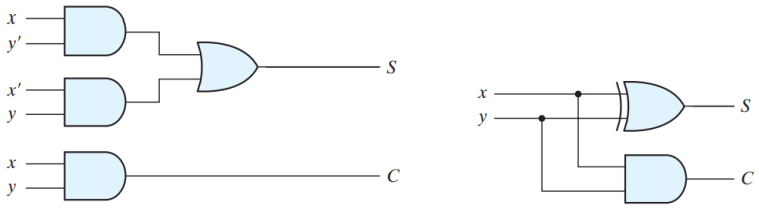
4.4) Binary Adder and Subtractor :

A binary adder–subtractor is a combinational circuit that performs the arithmetic operations of addition and subtraction with binary numbers.

Half Adder:

Table 4.3
Half Adder

x	y	C	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0



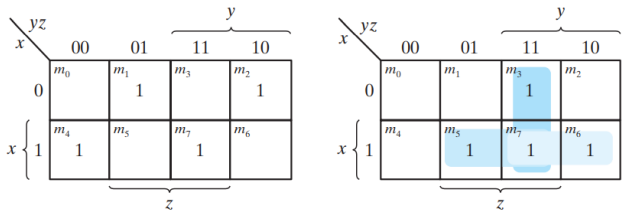
(a) $S = xy' + x'y$, $C = xy$ (b) $S = x \oplus y$, $C = xy$

FIGURE 4.5
Implementation of half adder

Full Adder: A full adder is a combinational circuit that forms the arithmetic sum of three bits. It consists of three inputs and two outputs. Two of the input variables, denoted by x and y , represent the two significant bits to be added. The third input, z , represents the carry from the previous lower significant position.

Table 4.4
Full Adder

x	y	z	C	S
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1



(a) $S = x'y'z + x'yz' + xy'z' + xyz$ (b) $C = xy + xz + yz$

FIGURE 4.6
K-Maps for full adder

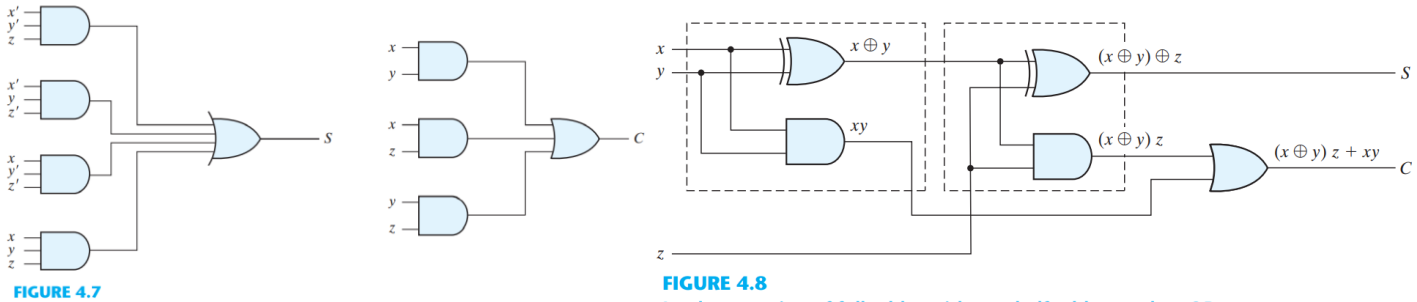


FIGURE 4.7
Implementation of full adder in sum-of-products form

FIGURE 4.8
Implementation of full adder with two half adders and an OR gate

Binary Adder: A binary adder is a digital circuit that produces the arithmetic sum of two binary numbers. It can be constructed with full adders connected in cascade, with the output carry from each full adder connected to the input carry of the next full adder in the chain.

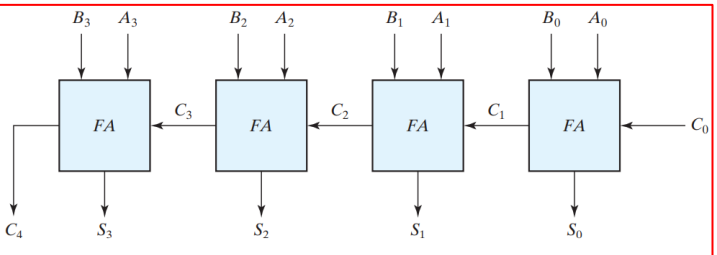


FIGURE 4.9
Four-bit adder

also known as *4 bit ripple carry adder.*

Carry Propagation : The total propagation time is equal to the propagation delay of a typical gate, times the number of gate levels in the circuit. The longest propagation delay time in an adder is the time it takes the carry to propagate through the full adders.

The signal from the input carry Ci to the output carry Ci+1 propagates through an AND gate and an OR gate (see 4.8) , which constitute two gate levels. If there are four full adders in the adder, the output carry C4 would have 2 * 4 = 8 gate levels from C0 to C4. For an n -bit adder, there are 2n gate levels for the carry to propagate from input to output.

The most widely used technique employs the principle of carry lookahead logic. Consider the circuit of the full adder shown in Fig. 4.10 . If we define two new binary variables

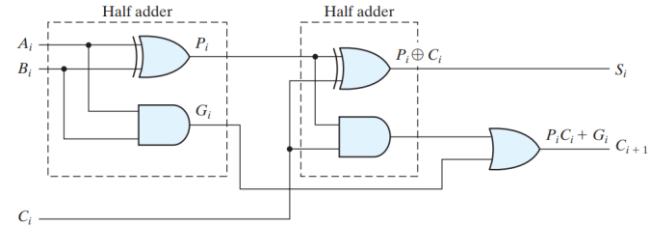


FIGURE 4.10
Full adder with *P* and *G* shown

$$P_i = A_i \oplus B_i$$
$$G_i = A_iB_i$$

the output sum and carry can respectively be expressed as

$$S_i = P_i \oplus C_i$$
$$C_{i+1} = G_i + P_iC_i$$

G_i is called a *carry generate*, and it produces a carry of 1 when both A_i and B_i are 1, regardless of the input carry C_i. P_i is called a *carry propagate*, because it determines whether a carry into stage i will propagate into stage $i + 1$ (i.e., whether an assertion of C_i will propagate to an assertion of C_{i+1}).

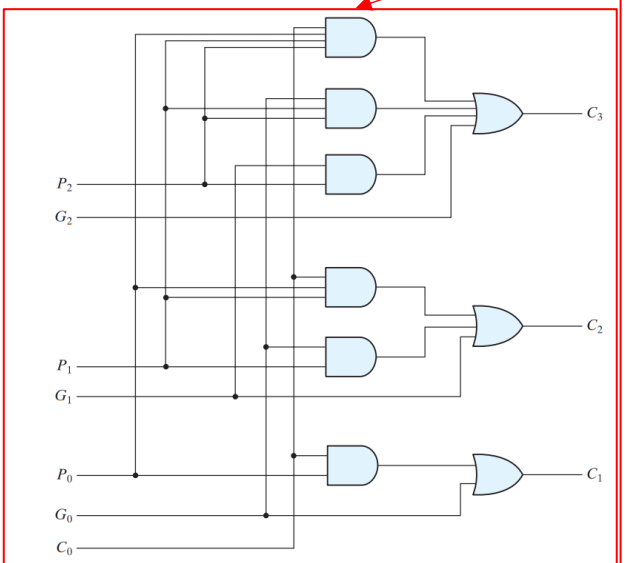


FIGURE 4.11
Logic diagram of carry lookahead generator

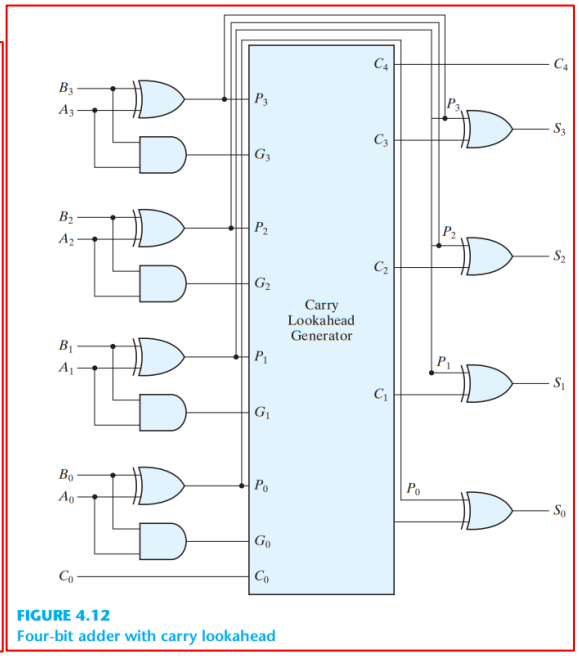


FIGURE 4.12
Four-bit adder with carry lookahead

Binary Subtractor : Remember that the subtraction A - B can be done by taking the 2's complement of B and adding it to A . The 2's complement can be obtained by taking the 1's complement and adding 1 to the least significant pair of bits. The 1's complement can be implemented with inverters, and a 1 can be added to the sum through the input carry.

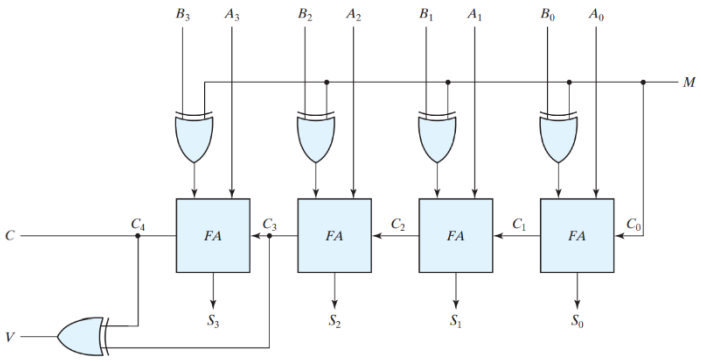
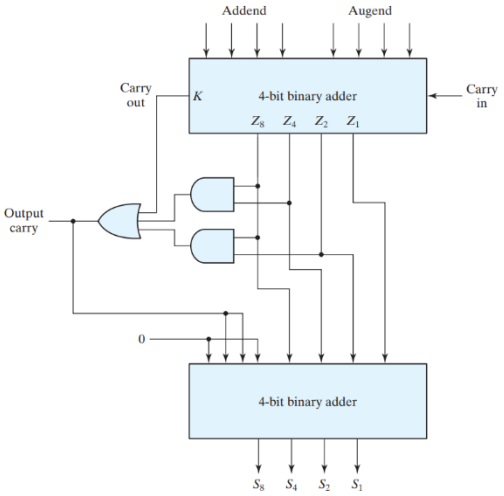


FIGURE 4.13
Four-bit adder–subtractor (with overflow detection)

The mode input M controls the operation. When M = 0, the circuit is an adder, and when M = 1, the circuit becomes a subtractor. Each exclusive-OR gate receives input M and one of the inputs of B . When M = 0, we have B Θ 0 = B. The full adders receive the value of B , the input carry is 0, and the circuit performs A plus B . When M = 1, we have B Θ 1 = B' and C0 = 1. The B inputs are all complemented and a 1 is added through the input carry.

Overflow : When two numbers with n digits each are added and the sum is a number occupying n + 1 digits, we say that an overflow occurred. The detection of an overflow after the addition of two binary numbers depends on whether the numbers are considered to be signed or unsigned. When two unsigned numbers are added, an overflow is detected from the end carry out of the most significant position. In the case of signed numbers, two details are important: the leftmost bit always represents the sign, and negative numbers are in 2's-complement form. When two signed numbers are added, the sign bit is treated as part of the number and the end carry does not indicate an overflow.

4.5) Decimal Adder :



BCD Adder: Since each input digit does not exceed 9, the output sum cannot be greater than 9 + 9 + 1 = 19, the 1 in the sum being an input carry.

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. When there is carry from current sum then we also add binary 6 into current sum.

4.6) Binary Multiplier :

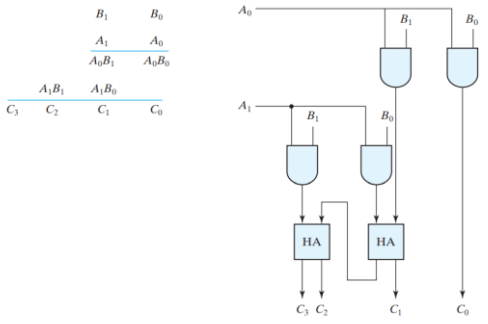


FIGURE 4.15
Two-bit by two-bit binary multiplier

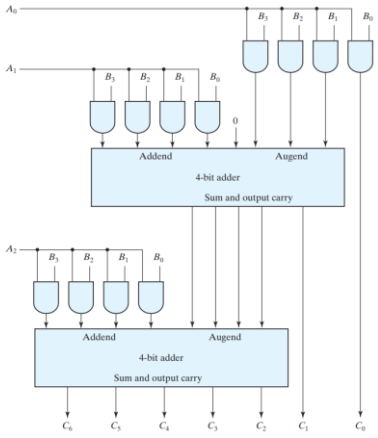


FIGURE 4.16
Four-bit by three-bit binary multiplier

*For J(A) multiplier bits and K multiplicand bits, we need (J * K) AND gates and (J – 1) K -bit adders to produce a product of (J + K) bits. In 4.16, multiplicand is B and multiplier is A.*

4.7) Magnitude Comparator :

A magnitude comparator is a combinational circuit that compares two numbers A and B and determines their relative magnitudes. The outcome of the comparison is specified by three binary variables that indicate whether A>B, A = B, or A<B.

Consider two numbers, A and B , with four digits each. Write the coefficients of the numbers in descending order of significance:

A = A3 A2 A1 A0

B = B3 B2 B1 B0

When the numbers are binary, the digits are either 1 or 0, and the equality of each pair of bits can be expressed logically with an exclusive-NOR function as xi = Ai Bi + A'i B'i for i = 0, 1, 2, 3 where xi = 1 only if the pair of bits in position i are equal (i.e., if both are 1 or both are 0). (A = B) = x3x2x1x0 the binary variable (A = B) is equal to 1 only if all pairs of digits of the two numbers are equal.

If the corresponding digit of A is 1 and that of B is 0, we conclude that A > B. If the corresponding digit of A is 0 and that of B is 1, we have A < B. The sequential comparison can be expressed logically by the two Boolean functions.

(A > B) = A3B'3 + x3 A2B'2 + x3x2 A1B'1 + x3x2x1 A0B'0 ,

(A < B) = A'3B3 + x3 A'2B2 + x3x2 A'1B1 + x3x2x1 A'0B0

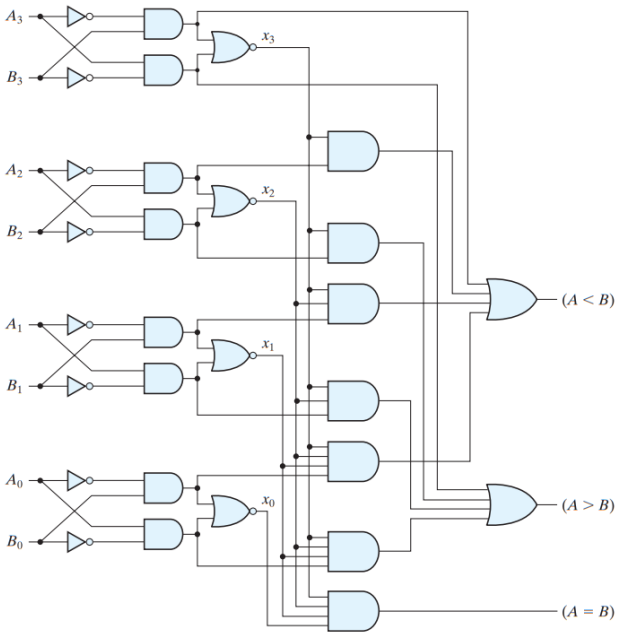


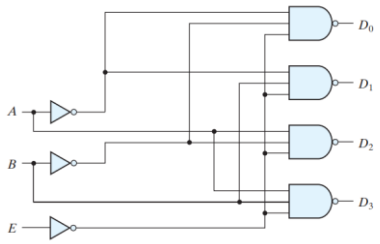
FIGURE 4.17
Four-bit magnitude comparator

4.8) Decoder :

A *decoder* is a combinational circuit that converts binary information from n input lines to a maximum of 2^n unique output lines. If the n -bit coded information has unused combinations, the *decoder* may have fewer than 2^n outputs.

A decoder with enable input can function as a demultiplexer— a circuit that receives information from a single line and directs it to one of 2^n possible output lines. The selection of a specific output is controlled by the bit combination of n selection

lines. Because decoder and demultiplexer operations are obtained from the same circuit, a decoder with an enable input is referred to as a *decoder – demultiplexer*.



(a) Logic diagram

E	A	B	D_0	D_1	D_2	D_3
1	X	X	1	1	1	1
0	0	0	0	1	1	1
0	0	1	1	0	1	1
0	1	0	1	1	0	1
0	1	1	1	1	1	1

(b) Truth table

FIGURE 4.19
Two-to-four-line decoder with enable input

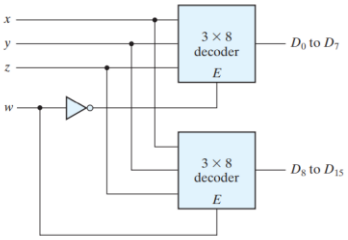


FIGURE 4.20
4×16 decoder constructed with two 3×8 decoders

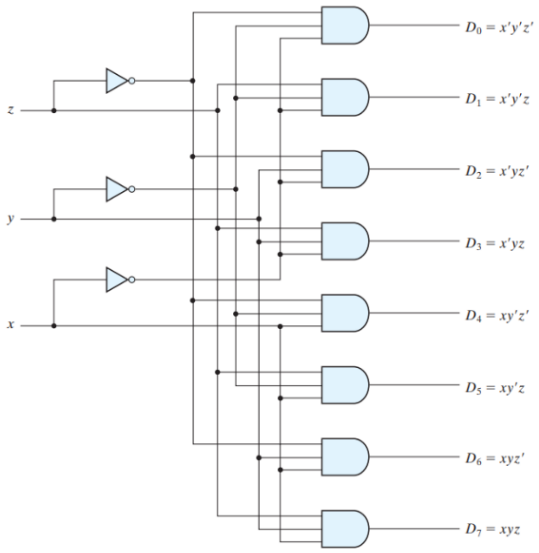


FIGURE 4.18
Three-to-eight-line decoder

Figure 4.20 shows two 3-to-8-line decoders with enable inputs connected to form a 4-to-16-line decoder. When w = 0, the top decoder is enabled and the other is disabled. The bottom decoder outputs are all 0's, and the top eight outputs generate minterms 0000 to 0111. When w = 1, the enable conditions are reversed: The bottom decoder outputs generate minterms 1000 to 1111, while the outputs of the top decoder are all 0's.

Combinational Logic Implementation :

From the truth table of the full adder (see Table 4.4), we obtain the functions for the combinational circuit in sum-of-minterms form:

$S(x, y, z) = \sum(1, 2, 4, 7)$, $C(x, y, z) = \sum (3, 5, 6, 7)$

If NAND gates are used for the decoder, as in Fig. 4.19, then the external gates must be NAND gates instead of OR gates. This is because a two-level NAND gate circuit implements a sum-of-minterms function and is equivalent to a two-level AND–OR circuit.

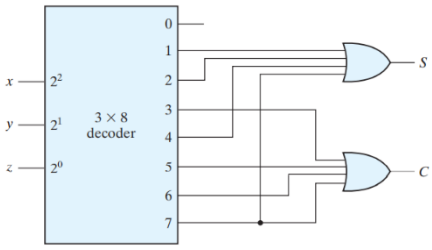


FIGURE 4.21
Implementation of a full adder with a decoder

4.9) Encoder :

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 encoder can be implemented with OR gates whose inputs are determined directly from the truth table. Output z is equal to 1 when the input octal digit is 1, 3, 5, or 7. Output y is 1 for octal digits 2, 3, 6, or 7, and output x is 1 for digits 4, 5, 6, or 7. These conditions can be expressed by the following Boolean output functions: z = D1 + D3 + D5 + D7, y = D2 + D3 + D6 + D7, x = D4 + D5 + D6 + D7, 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

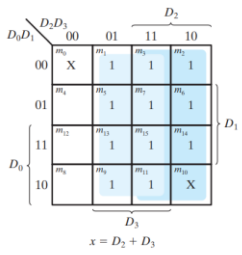
Problem: 1) if D3 and D6 are 1 simultaneously, the output of the encoder will be 111 because all three outputs are equal to 1. The output 111 does not represent either binary 3 or binary 6.

2) Another ambiguity in the octal-to-binary encoder is that an output with all 0's is generated when all the inputs are 0; but this output is the same as when D0 is equal to 1. The discrepancy can be resolved by providing one more output to indicate whether at least one input is equal to 1.

Priority Encoder : A priority encoder is an encoder circuit that includes the priority function. The operation of 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. In table 4.8, the priority of D3 is high so if its value is 1 we would not care the value of other inputs. And we will put its binary value in output. Similarly, for others.

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



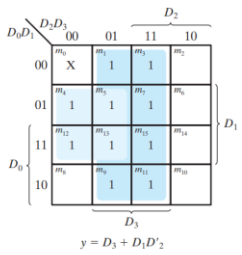


FIGURE 4.22
Maps for a priority encoder

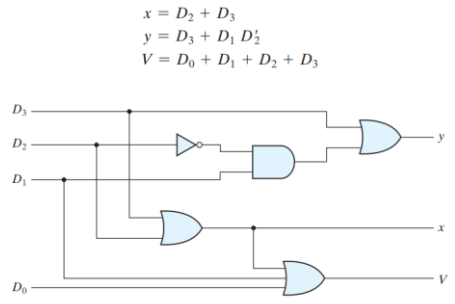


FIGURE 4.23
Four-input priority encoder

4.10) Multiplexer :

A multiplexer is a combinational circuit that selects binary information from one of many input lines and directs it to a single output line. The selection of a particular input line is controlled by a set of selection lines. A two-to-one-line multiplexer connects one of two 1-bit sources to a common destination, as shown in Fig. 4.24.

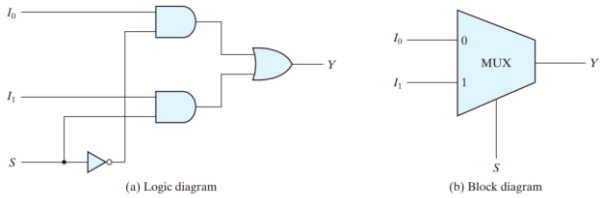


FIGURE 4.24
Two-to-one-line multiplexer

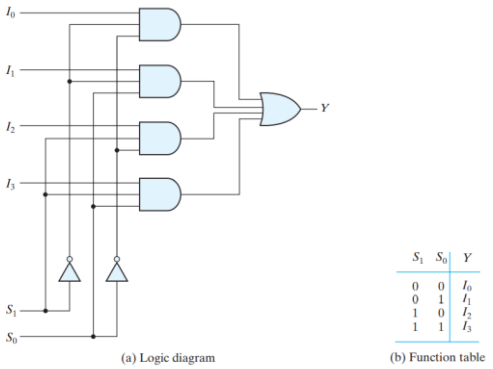


FIGURE 4.25
Four-to-one-line multiplexer

Multiplexer circuits can be combined with common selection inputs to provide multiple-bit selection logic. As an illustration, a quadruple 2-to-1-line multiplexer is shown in Fig. 4.26 . The circuit has four multiplexers, each capable of selecting one of two input lines. Output Y0 can be selected to come from either input A0 or input B0. Similarly, output Y1 may have the value of A1 or B1, and so on.

Input selection line S selects one of the lines in each of the four multiplexers. The enable input E must be active (i.e., asserted) for normal operation. Although the circuit contains four 2-to-1-line multiplexers, we are more likely to view it as a circuit that selects one of two 4-bit sets of data lines. As shown in the function table, the unit is enabled when E = 0. Then, if S = 0, the four A inputs have a path to the four outputs. If, by contrast, S = 1, the four B inputs are applied to the outputs. The outputs have all 0's when E = 1, regardless of the value of S.

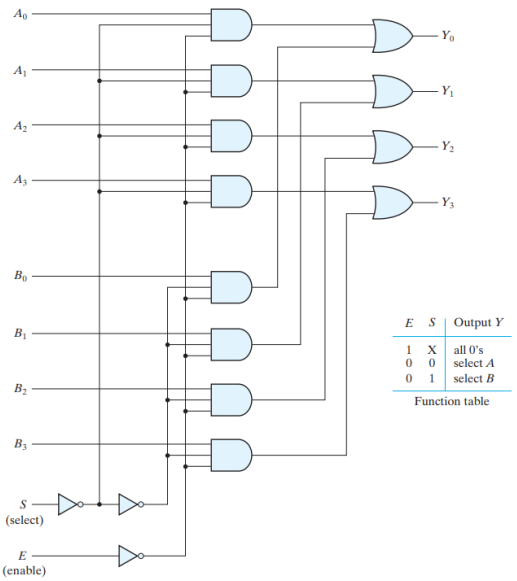


FIGURE 4.26
Quadruple two-to-one-line multiplexer

Boolean Function Implementation :

The general procedure for implementing any Boolean function of n variables with a multiplexer with n - 1 selection inputs (S0,S1) and 2^{n-1} data inputs (0,1,2,3) follows from the previous example.

F (x, y, z) = ∑(1, 2, 6, 7)

F (A, B, C, D) = ∑(1, 3, 4, 11, 12, 13, 14, 15)

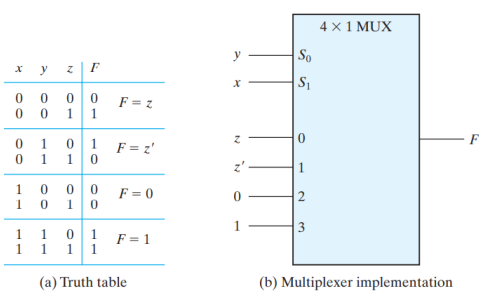


FIGURE 4.27
Implementing a Boolean function with a multiplexer

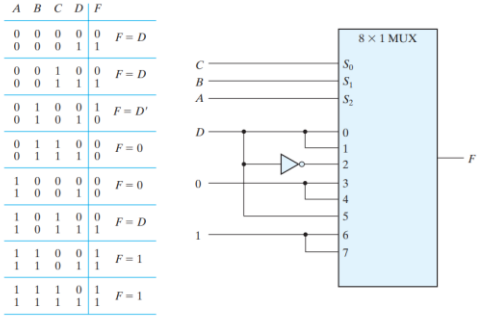


FIGURE 4.28
Implementing a four-input function with a multiplexer

Three-State Gates : Two of the states are signals equivalent to logic 1 and logic 0 as in a conventional gate. The third state is a high-impedance state in which (1) the logic behaves like an open circuit, which means that the output appears to be disconnected, (2) the circuit has no logic significance, and (3) the circuit connected to the output of the three-state gate is not affected by the inputs to the gate.

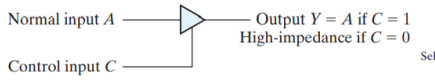


FIGURE 4.29
Graphic symbol for a three-state buffer

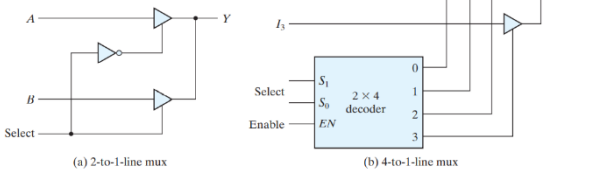
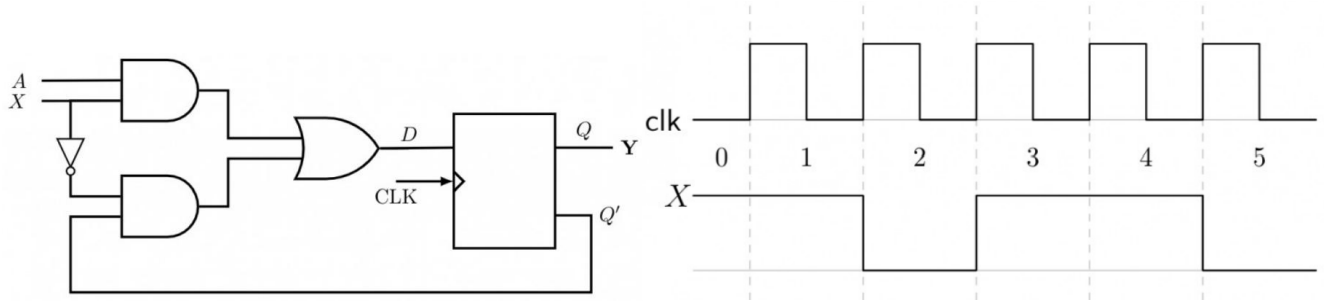


FIGURE 4.30
Multiplexers with three-state gates

<u>Questions on Combinational Circuits</u> :

1) Consider the following circuit involving a positive edge triggered DFF.



Consider the following timing diagram. Let Ai represents the logic level on the line a in the i-th clock period. Let represent the compliment of . The correct output sequence on over the clock periods through is

A0 A1 A'1 A3 A4 *A1 A2 A'2 A3 A4*

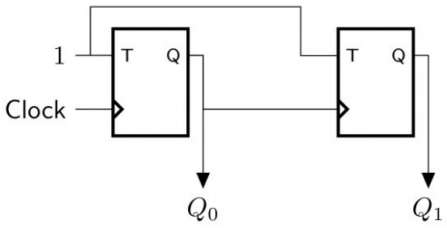
A0 A1 A'2 A3 A4 *A1 A'2 A3 A4 A'5*

Answer :

 this is one cycle.

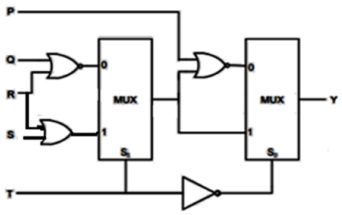
For clk1, is so, A0
For clk2, is so A1
For clk3, is so A'1
For clk4, is so A3
For clk5, is so A4

2) In the sequential circuit shown below, if the initial value of the output Q1Q0 is 00. What are the next four values of Q1Q0 ?



Answer : 11, 10, 01, 00. When Q0 goes from 0 to 1 Q1 gets its value.

For the circuit shown in the figure, the delays of NOR gates, OR, multiplexers and inverters are 4 ns, 3 ns, 2 ns and 1 ns, respectively.



If all the inputs P, Q, R, S and T are applied at the same time instant, the maximum propagation delay (in ns) of the circuit is

(a) 7 (b) 8 (c) 10 (d) 11

3)

Answer: (d)

Solution:

For T= 0

From MUX 1 line 0 will be selected and from MUX 2 line 1 will be selected.

∴Delay for (T= 0) = delay of NOR gate + Delay of MUX 1+Delay of MUX 2 = 4+2+2 = 8ns

For T= 1 From MUX 1 line 1 will be selected and from MUX 2 line 0 will be selected.

∴Total delay for (T=1)

= Delay of OR gate +Delay of MUX1+ Delay of NOR +Delay of MUX 2= 3+2+4+2 = 11nsec

5. Sequential Circuits

5.1) Sequential Circuit :

It consists of a combinational circuit to which storage elements are connected to form a feedback path. 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 at that time.

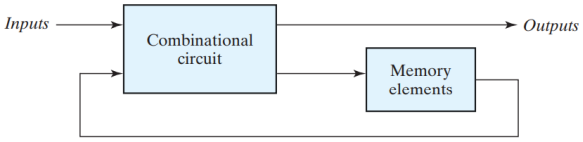


FIGURE 5.1
Block diagram of sequential circuit

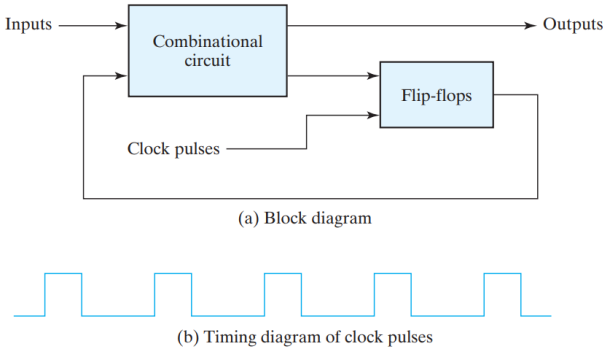


FIGURE 5.2
Synchronous clocked sequential circuit

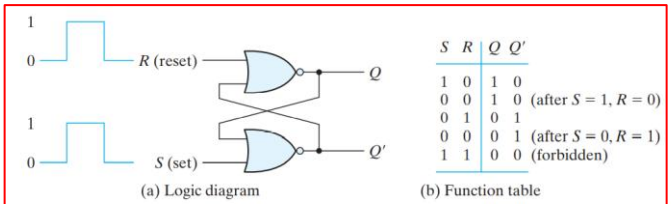
Thus, a *sequential circuit* is specified by a time sequence of inputs, outputs, and internal states. In contrast, the outputs of *combinational logic* depend only on the present values of the inputs.

A *synchronous sequential circuit* is a system whose behaviour can be defined from the knowledge of its signals at discrete instants of time. The behaviour 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.

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 . . In practice, the clock pulses determine **when** computational activity will occur within the circuit, and other signals (external inputs and otherwise) determine **what** changes will take place affecting the storage elements and the outputs.

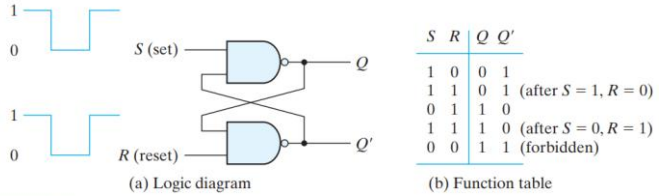
- Synchronous sequential circuits that use *clock pulses* to control storage elements are called clocked sequential circuits. The storage elements (memory) used in clocked sequential circuits are called flip-flops. A flip-flop is a binary storage device capable of storing one bit of information. In a stable state, the output of a flip-flop is either 0 or 1.

SR Latch :



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

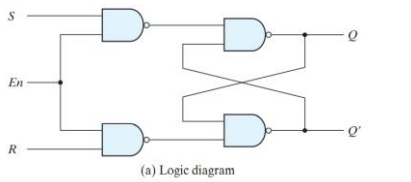
FIGURE 5.3
SR latch with NOR gates



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

FIGURE 5.4
SR latch with NAND gates

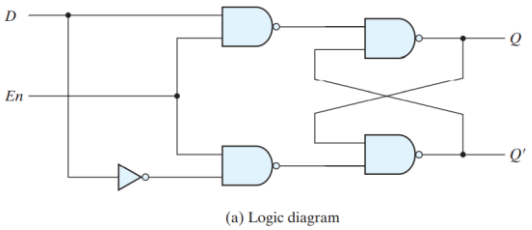
$Q_{n+1} = S + R Q_n$ (for above image)



En	S	R	Next state of Q
0	X	X	No change
1	0	0	No change
1	0	1	$Q = 0$; reset state
1	1	0	$Q = 1$; set state
1	1	1	Indeterminate

(b) Function table

FIGURE 5.5
SR latch with control input



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

(b) Function table

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, shown in Fig. 5.6.

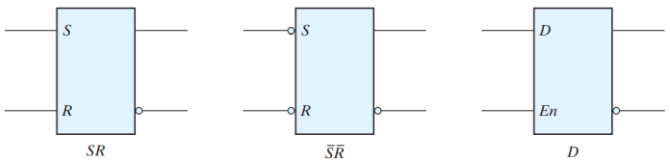
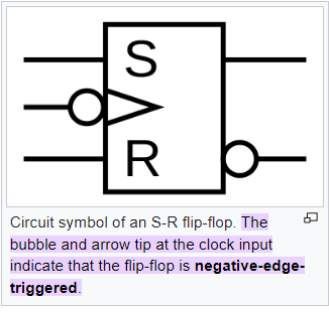
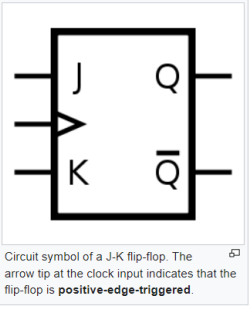


FIGURE 5.7
Graphic symbols for latches



Circuit symbol of an S-R flip-flop. The bubble and arrow tip at the clock input indicate that the flip-flop is **negative-edge-triggered**.



Circuit symbol of a J-K flip-flop. The arrow tip at the clock input indicates that the flip-flop is **positive-edge-triggered**.

5.2) Storage Element : Flip-Flop

The state of a latch or flip-flop is switched by a change in the control input. This momentary change is called a trigger, and the transition it causes is said to trigger the flip-flop. The D latch with pulses in its control input is essentially a flip-flop that is triggered every time the pulse goes to the logic-1 level. A *flip-flop* triggers only during a signal transition (from 0 to 1 or from 1 to 0) of the synchronizing signal (clock) and is disabled during the rest of the *clock pulse.*

Edge-Triggered D Flip-Flop :

The circuit samples the D input and changes its output Q only at the negative edge of the synchronizing or controlling clock (designated as Clk). When the clock is 0, the output of the inverter is 1. The slave latch is enabled, and its output Q is equal to the master output Y . The master latch is disabled because Clk = 0. When the input pulse changes to the logic-1 level, the data from the external D input are transferred to the master. The slave, however, is disabled as long as the clock remains at the 1 level, because its enable input is equal to 0. Any change in the input changes the master output at Y, but cannot affect the slave output. When the clock pulse returns to 0, the master is disabled and is isolated from the D input. At the same time, the slave is enabled and the value of Y is transferred to the output of the flip-flop at Q . Thus, a change in the output of the flip-flop can be triggered only by and during the transition of the clock from 1 to 0.

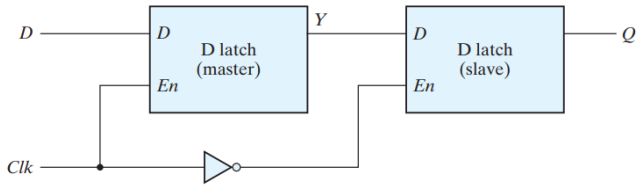


FIGURE 5.9
Master–slave *D* flip-flop

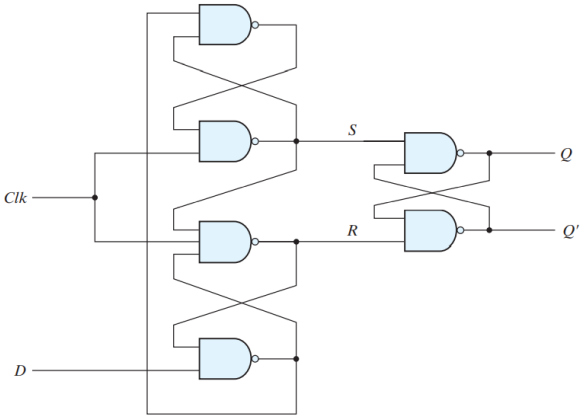
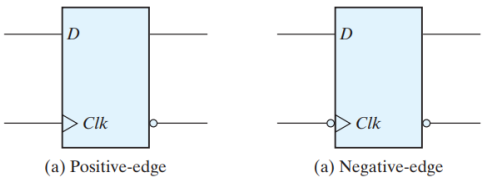


FIGURE 5.10
D-type positive-edge-triggered flip-flop



(a) Positive-edge (a) Negative-edge

FIGURE 5.11
Graphic symbol for edge-triggered *D* flip-flop

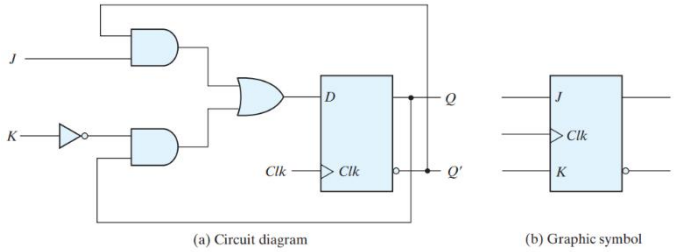
Setup Time : There is a minimum time called the setup time during which the D input must be maintained at a constant value prior to the occurrence of the clock transition.

Hold Time : hold time during which the D input must not change after the application of the positive transition of the clock.

Propagation delay : The propagation delay time of the flip-flop is defined as the interval between the trigger edge and the stabilization of the output to a new state.

Other Flip-flop : The most economical and efficient flip-flop constructed in this manner is the edge-triggered D flipflop, because it requires the smallest number of gates. Other types of flip-flops can be constructed by using the D flip-flop and external logic. Two flip-flops less widely used in the design of digital systems are the JK and T flip-flops.

There are three operations that can be performed with a flip-flop: *Set it to 1, reset it to 0 and complement its output.*

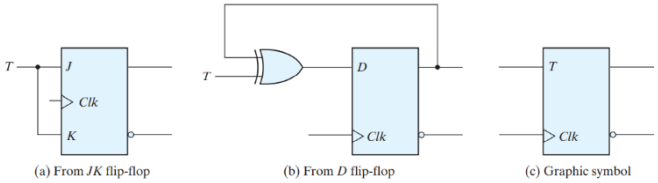


(a) Circuit diagram (b) Graphic symbol

FIGURE 5.12
JK flip-flop

D = JQ' + K'Q When J = 1 and K = 0, D = Q' + Q = 1, so the next clock edge sets the output to 1. When J = 0 and K = 1, D = 0, so the next clock edge resets the output to 0. When both J = K = 1 and D = Q', the next clock edge complements the output. When both J = K = 0 and D = Q, the clock edge leaves the output unchanged.

T = 0 (J = K = 0), a clock edge does not change the output. When T = 1 (J = K = 1), a clock edge complements the output. The complementing flip-flop is useful for designing binary counters. The T flip-flop can be constructed with a D flip-flop



(a) From *JK* flip-flop (b) From *D* flip-flop (c) Graphic symbol

FIGURE 5.13
T flip-flop

and an exclusive-OR gate as shown in Fig. 5.13 (b). The expression for the D input is D = T ⊕ Q = TQ' + T'Q.

Table 5.1
Flip-Flop Characteristic Tables

JK Flip-Flop

J	K	Q(t + 1)	
0	0	Q(t)	No change
0	1	0	Reset
1	0	1	Set
1	1	Q'(t)	Complement

D Flip-Flop

D	Q(t + 1)	
0	0	Reset
1	1	Set

T Flip-Flop

T	Q(t + 1)	
0	Q(t)	No change
1	Q'(t)	Complement

T flip-flop [edit]

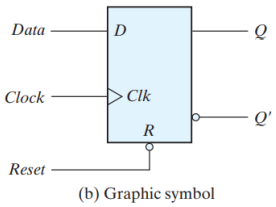
States		Input
Present	Next	T
0	0	0
0	1	1
1	0	1
1	1	0

This is excitation table

Characteristic Equations : For the D flip-flop, we have the characteristic equation Q(t + 1) = D.
The characteristic equation for the JK flip-flop is Q(t + 1) = JQ' + K'Q
The characteristic equation for the T flip-flop is Q(t + 1) = T ⊕ Q = TQ' + T'Q, by using these equations you can make excitation table which shows the minimum inputs that are necessary to generate a particular next state.

NOTE : Positive edge cycle = 0 to 1, Negative edge cycle = 1 to 0, if clear = 1 sets all state to 0

Direct Inputs : Some flip-flops have asynchronous inputs that are used to force the flip-flop to a particular state independently of the clock. The input that sets the flip-flop to 1 is called **preset** or **direct set** . The input that clears the flip-flop to 0 is called **clear** or **direct reset** . When power is turned on in a digital system, the state of the flip-flops is unknown. The direct inputs are useful for bringing all flip-flops in the system to a known starting state prior to the clocked operation. The clock at Clk is shown with an upward arrow to indicate that the flip-flop triggers on the positive edge of the clock. Diagram b is positive edge.



(b) Graphic symbol

R	Clk	D	Q	Q'
0	X	X	0	1
0	↑	0	0	1
0	↑	1	1	0

(b) Function table

FIGURE 5.14
D flip-flop with asynchronous reset

5.3) Analysis of Clocked Sequential Circuits :

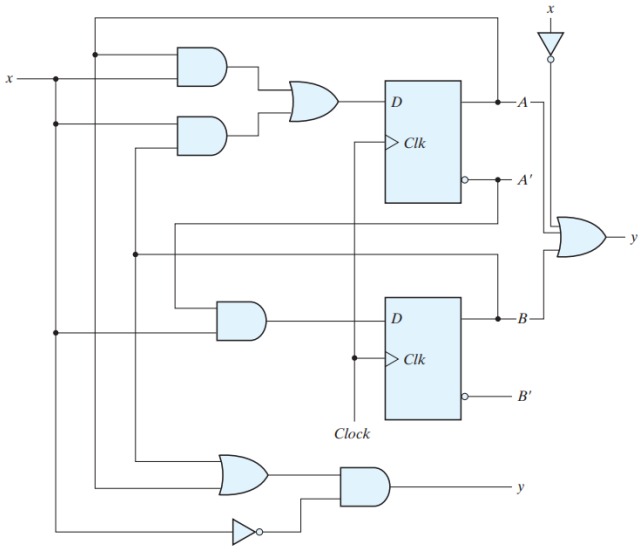


FIGURE 5.15
Example of sequential circuit

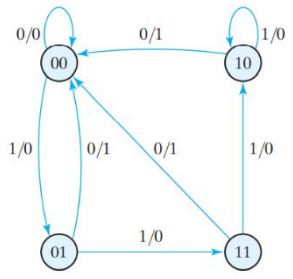
State Equations : A state equation (also called a transition equation) specifies the next state as a function of the present state and inputs.
A(t + 1) = A(t)x(t) + B(t)x(t)
B(t + 1) = A'(t)x(t)
we can omit the designation (t) after each variable for convenience and can express the state equations in the more compact form
A(t + 1) = Ax + Bx, B(t + 1) = A'x
y = (A + B)x'

Table 5.2
State Table for the Circuit of Fig. 5.15

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	0	0	1
1	0	1	1	0	0
1	1	0	0	0	1
1	1	1	1	0	0

Table 5.3
Second Form of the State Table

Present State		Next State				Output	
		x = 0		x = 1		x = 0	x = 1
A	B	A	B	A	B	y	y
0	0	0	0	0	1	0	0
0	1	0	0	1	1	1	0
1	0	0	0	1	0	1	0
1	1	0	0	1	0	1	0



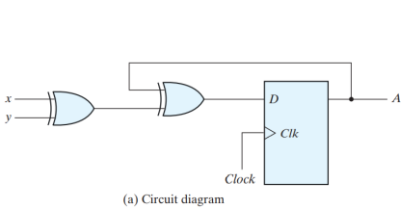
State Diagram : The steps presented in this example are summarized below :
Circuit diagram → *Equations* → *State table* → *State diagram*

In general, a sequential circuit with m flipflops and n inputs needs 2^{m+n} rows in the state table. The binary numbers from 0 to 2^{m+n} - 1 are listed under the present-state and input columns

Flip-Flop Input Equations : For example, the following input equation specifies an OR gate with inputs x and y connected to the D input of a flip-flop whose output is labelled with the symbol Q : $D_Q = x + y$
The logic diagram of the circuit can be expressed algebraically with two flip-flop input equations and an output equation:
$D_A = Ax + Bx$
$D_B = A'x$

y = (A + B)x'

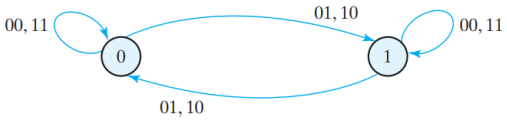
1) ***Analysis with D Flip-Flops*** : $D_A = A \oplus x \oplus y$



(a) Circuit diagram

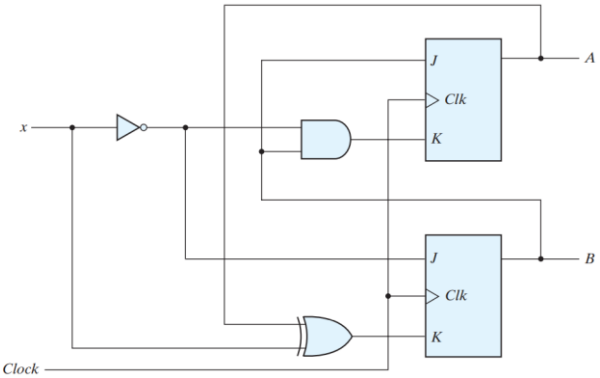
Present state	Inputs		Next state
A	x	y	A
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	0
1	0	0	1
1	0	1	0
1	1	0	0
1	1	1	1

(b) State table



(c) State diagram

2) ***Analysis with JK Flip-Flops*** :



$J_A = B \quad K_A = Bx'$

$J_B = x' \quad K_B = A'x + Ax' = A \oplus x$

Table 5.4
State Table for Sequential Circuit with JK Flip-Flops

Present State		Input	Next State		Flip-Flop Inputs			
A	B	x	A	B	J_A	K_A	J_B	K_B
0	0	0	0	1	0	0	1	0
0	0	1	0	0	0	0	0	1
0	1	0	1	1	1	1	1	0
0	1	1	1	0	1	0	0	1
1	0	0	1	1	0	0	1	1
1	0	1	1	0	0	0	0	0
1	1	0	0	0	1	1	1	1
1	1	1	1	1	1	0	0	0

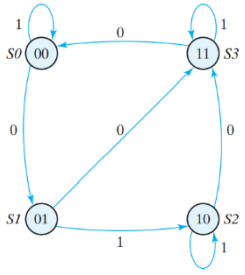
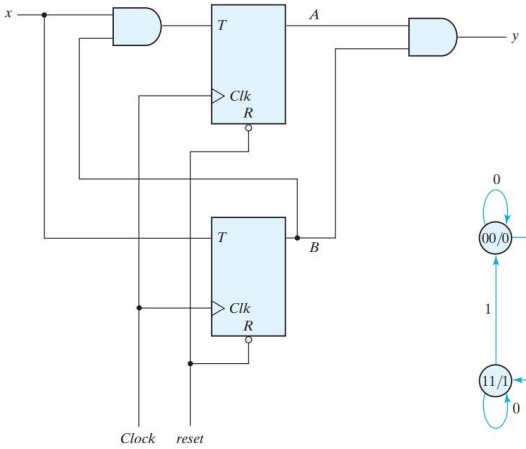


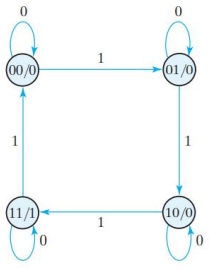
FIGURE 5.18
Sequential circuit with *JK* flip-flop

The characteristic equations for the flip-flops are obtained by substituting A or B for the name of the flip-flop, instead of Q :
A(t + 1) = JA' + K'A
B(t + 1) = JB' + K'B
Substituting the values of J_A and K_A from the input equations, we obtain the state equation for A and B:
A(t + 1) = BA' + (Bx')' A = A'B + AB' + Ax
B(t + 1) = x'B' + (A ⊕ x)'B = B'x' + ABx + A'Bx'

3) ***Analysis with T Flip-Flops*** :



(a) Circuit diagram



(b) State diagram

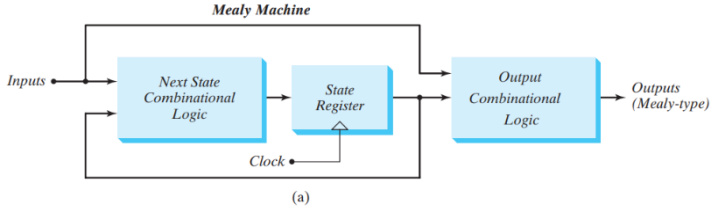
Table 5.5
State Table for Sequential Circuit with T Flip-Flops

Present State		Input	Next State		Output
A	B	x	A	B	y
0	0	0	0	0	0
0	0	1	0	1	0
0	1	0	0	1	0
0	1	1	1	0	0
1	0	0	1	0	0
1	0	1	1	1	0
1	1	0	1	1	1
1	1	1	0	0	1

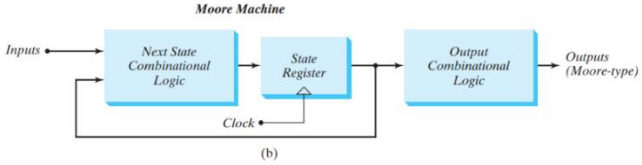
FIGURE 5.20
Sequential circuit with *T* flip-flops (Binary Counter)

The characteristic equation $Q(t + 1) = T \oplus Q = T'Q + TQ'$
Input equations and an output equation: $T_A = Bx$, $T_B = x$, y = AB
Yielding A(t + 1) = (Bx)'A + (Bx)A' = AB' + Ax' + A'Bx, B(t + 1) = x ⊕ B

Mealy and Moore Models of Finite State Machines :



(a)



(b)

FIGURE 5.21
Block diagrams of Mealy and Moore state machines

In a ***Moore model****, the outputs of the sequential circuit are synchronized with the clock, because they depend only on flip-flop outputs that are synchronized with the clock.*

*The output of the **Mealy machine** is the value that is present immediately before the active edge of the clock.*

5.4) State reduction and Assignment :

The analysis of sequential circuits starts from a circuit diagram and culminates in a state table or diagram. The design (synthesis) of a sequential circuit starts from a set of specifications and culminates in a logic diagram.

State Reduction : The reduction in the number of flip-flops in a sequential circuit is referred to as the state-reduction problem. State-reduction algorithms are concerned with procedures for reducing the number of states in a state table, while keeping the external input–output requirements unchanged.

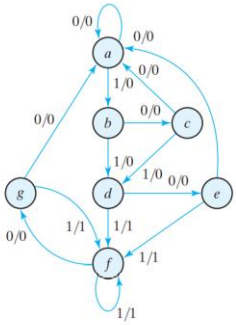


FIGURE 5.25
State diagram

Table 5.6
State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	g	f	0	1
g	a	f	0	1

Table 5.7
Reducing the State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	f	0	1
e	a	f	0	1
f	e	f	0	1

Table 5.8
Reduced State Table

Present State	Next State $x=0$	Next State $x=1$	Output $x=0$	Output $x=1$
a	a	b	0	0
b	c	d	0	0
c	a	d	0	0
d	e	d	0	1
e	a	d	0	1

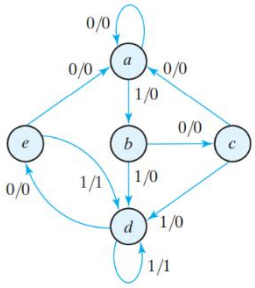


FIGURE 5.26
Reduced state diagram

State Assignment :

Table 5.9
Three Possible Binary State Assignments

State	Assignment 1, Binary	Assignment 2, Gray Code	Assignment 3, One-Hot
a	000	000	00001
b	001	001	00010
c	010	011	00100
d	011	010	01000
e	100	110	10000

One-hot encoding *usually leads to simpler decoding logic for the next state and output.* One-hot machines can be faster than machines with sequential binary encoding, and the silicon area required by the extra flip-flops can be offset by the area saved by using simpler decoding logic. This trade-off is not guaranteed, so it must be evaluated for a given design.

6. Registers and Counters

6.1) Registers :

A *register* is a **group of flip-flops**, each one of which shares a common clock and is capable of storing one bit of information. An n -bit register consists of a group of n flip-flops capable of storing n bits of binary information. In addition to the flip-flops, a register may have combinational gates that perform certain data-processing tasks.

A *counter* is essentially a register that goes through a predetermined sequence of binary states. The gates in the counter are connected in such a way as to produce the prescribed sequence of states. Although counters are a special type of register, it is common to differentiate them by giving them a different name.

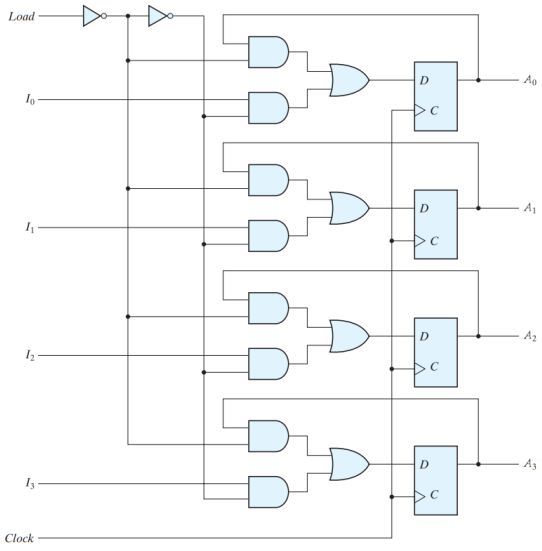


FIGURE 6.2
Four-bit register with parallel load

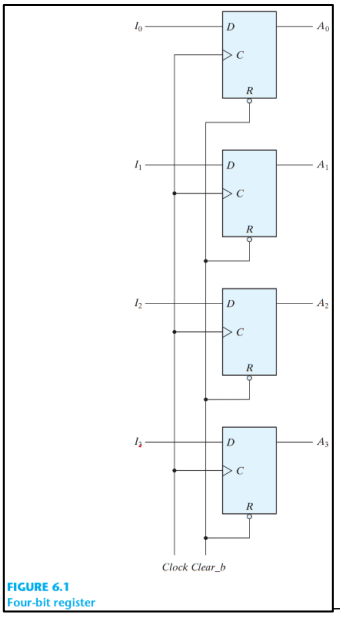
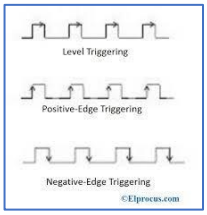


FIGURE 6.1
Four-bit register

Registers are made of edge-triggered FFs whereas latches are made from level-triggered FFs.



The value of (I3 , I2 , I1 , I0) immediately before the clock edge determines the value of (A3 , A2 , A1 , A0) after the clock edge. The four outputs can be sampled at any time to obtain the binary information stored in the register. The input Clear_b goes to the active-low R (reset) input of all four flip-flops. When this input goes to 0, all flip-flops are reset asynchronously. The Clear_b input is useful for clearing the register to all 0's prior to its clocked operation.

Register with Parallel Load :

The transfer of new information into a register is referred to as loading or updating the register. If all the bits of the register are loaded simultaneously with a common clock pulse, we say that the loading is done in parallel.

A four-bit data-storage register with a load control input that is directed through gates and into the D inputs of the flip-flops is shown in Fig. 6.2 . The additional gates implement a two-channel mux whose output drives the input to the register with either the data bus or the output of the register. The transfer of information from the data inputs or the outputs of the register is done simultaneously with all four bits in response to a clock edge.

6.2) Shift Registers :

A register capable of shifting the binary information held in each cell to its neighbouring 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. 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.

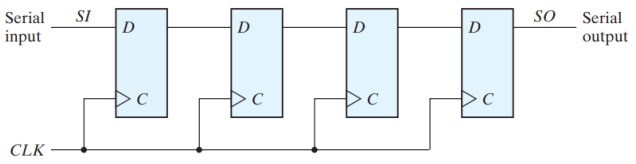


FIGURE 6.3
Four-bit shift register

Serial Transfer : The Datapath 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.

The serial transfer of information from register A to register B is done with shift registers, as shown in the block diagram of Fig. 6.4 (a). The serial output (SO) of register A is connected to the serial input (SI) of register B. To prevent the

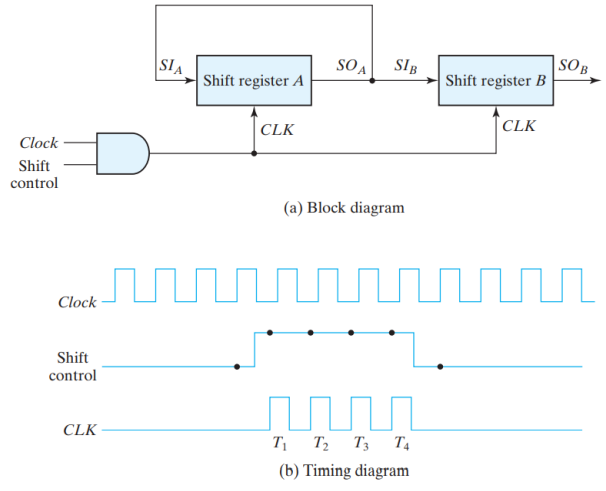


FIGURE 6.4
Serial transfer from register *A* to register *B*

loss of information stored in the source register, the information in register A is made to circulate by connecting the serial output to its serial input. The initial content of register B is shifted out through its serial output and is lost unless it is transferred to a third shift register. The shift control input determines when and how many times the registers are shifted.

Assume that the binary content of A before the shift is 1011 and that of B is 0010.

Table 6.1
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 difference between the serial and the parallel mode of operation should be apparent from this example. In the parallel mode, information is available from all bits of a register and all bits can be transferred simultaneously during one clock pulse. In the serial mode, 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 :

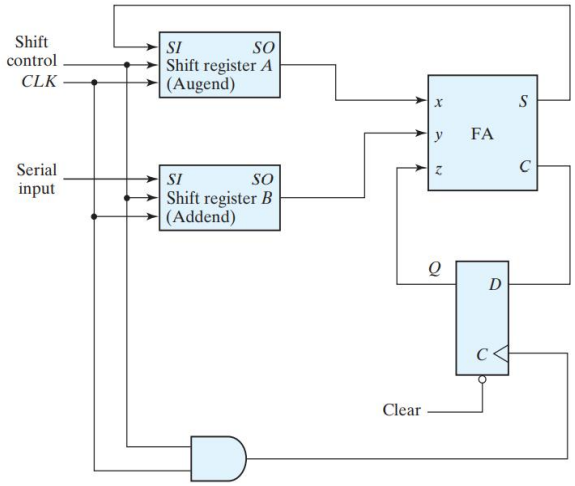


FIGURE 6.5
Serial adder

Initially, register A holds the augend, register B holds the addend, and the carry flip-flop is cleared to 0. The outputs (SO) of A and B provide a pair of significant bits for the full adder at x and y. Output Q of the flip-flop provides the input carry at z. The shift control enables both registers and the carry flip-flop, so at the next clock pulse, both registers are shifted once to the right, the sum bit from S enters the leftmost flip-flop of A, and the output carry is transferred into flip-flop Q. The shift control enables the registers for a number of clock pulses equal to the number of bits in the registers. For each succeeding clock pulse, a new sum bit is transferred to A, a new carry is transferred to Q, and both registers are shifted once to the right. This process continues until the shift control is disabled. Thus, the addition is accomplished by passing each pair of bits together with the previous carry through a single full-adder circuit and transferring the sum, one bit at a time, into register A.

If a JK flip- flop is used for Q, it is necessary to determine the values of inputs J and K by referring to the excitation table (Table 5.12). Two flip-flop input equations and the output equation can be simplified by means of maps to

$J_Q = xy$

$K_Q = x'y' = (x + y)'$

$S = x \oplus y \oplus Q$

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 has both shifts and parallel-load capabilities, it is referred to as a *universal shift register*.

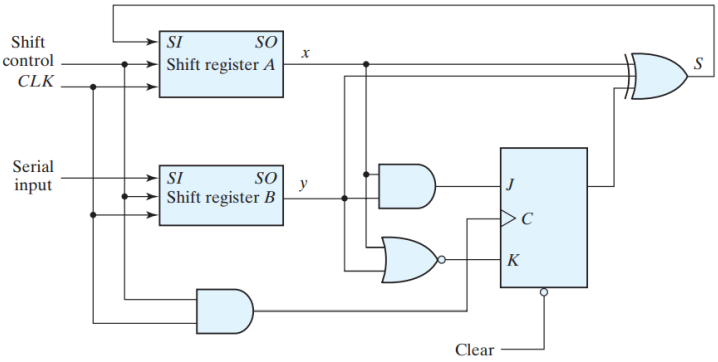


FIGURE 6.6
Second form of serial adder

6.3) Ripple Counter :

A register that goes through a prescribed sequence of states upon the application of input pulses is called a *counter*. The input pulses may be clock pulses, or they may originate from some external source and may occur at a fixed interval of time or at random. The sequence of states may follow the binary number sequence or any other sequence of states. A counter that follows the binary number sequence is called a *binary counter* . An n -bit binary counter consists of n flip-flops and can count in binary from 0 through 2^n - 1.

Counters are available in two categories: *ripple counters* and *synchronous counters*. 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. In a synchronous counter, the C inputs of all flip-flops receive the common clock.

Binary Ripple Counter : 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.

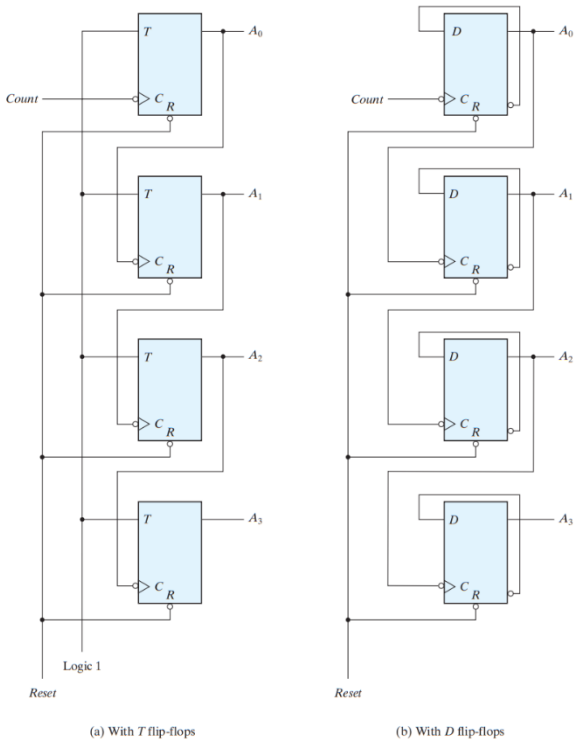


FIGURE 6.8
Four-bit binary ripple counter

The count starts with binary 0 and increments by 1 with each count pulse input. After the count of 15, the counter goes back to 0 to repeat the count. The least significant bit, A0, is complemented with each count pulse input. Every time that A0 goes from 1 to 0, it complements A1. Every time that A1 goes from 1 to 0, it complements A2. Every time that A2 goes from 1 to 0, it complements A3, and so on for any other higher order bits of a ripple counter.

For example, consider the transition from count 0011 to 0100. A0 is complemented with the count pulse. Since A0 goes from 1 to 0, it triggers A1 and complements it. As a result, A1 goes from 1 to 0, which in turn complements A2, changing it from 0 to 1. A2 does not trigger A3, because A2 produces a positive transition and the flip-flop responds only to negative transitions. Thus, the count from 0011 to 0100 is achieved by changing the bits one at a time, so the count goes from 0011 to 0010, then to 0000, and finally to 0100.

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. Any other bit in the sequence is complemented if its previous least significant bit goes from 0 to 1. Therefore, the diagram of a binary countdown counter looks the same as the binary ripple counter in Fig. 6.8 , provided that all flip-flops trigger on the positive edge of the clock. (The bubble in the C inputs must be absent.) If negative-edge-triggered flip-flops are used, then the C input of each flip-flop must be connected to the complemented output of the previous flip-flop. Then, when the true output goes from 0 to 1, the complement will go from 1 to 0 and complement the next flip-flop as required.

BCD Ripple Counter :

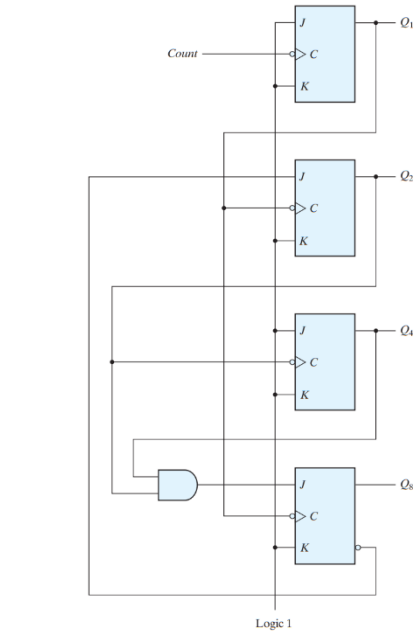


FIGURE 6.10
BCD ripple counter

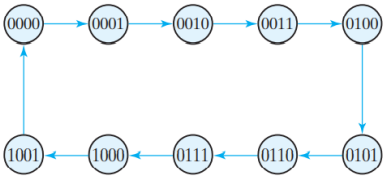


FIGURE 6.9
State diagram of a decimal BCD counter

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.

Q1 changes state after each clock pulse. Q2 complements every time Q1 goes from 1 to 0, as long as Q8 = 0. When Q8 becomes 1, Q2 remains at 0. Q4 complements every time Q2 goes from 1 to 0. Q8 remains at 0 as long as Q2 or Q4 is 0. When both Q2 and Q4 become 1, Q8 complements when Q1 goes from 1 to 0. Q8 is cleared on the next transition of Q1.

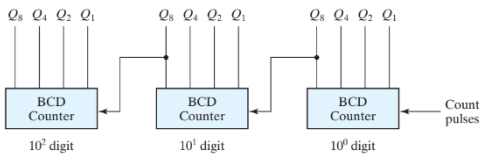


FIGURE 6.11
Block diagram of a three-decade decimal BCD counter

A decade counter, since it counts from 0 to 9. To count in decimal from 0 to 99, we need a two-decade counter. To count from 0 to 999, we need a three-decade counter.

6.4) Synchronous Counters :

Synchronous counters are different from ripple counters in that clock pulses are applied to the inputs of all flip-flops.

Binary Counter :

A flip-flop in any other position is complemented when all the bits in the lower significant positions are equal to 1 . For example, if the present state of a four-bit counter is A3A2A1A0 = 0011, the next count is 0100. A0 is always complemented. A1 is complemented because the present state of A0 = 1. A2 is complemented because the present state of A1A0 = 11. However, A3 is not complemented, because the present state of A2A1A0 = 011, which does not give an all-1's condition.

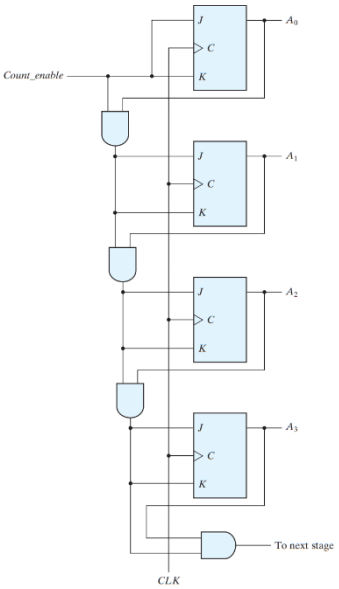


FIGURE 6.12
Four-bit synchronous binary counter

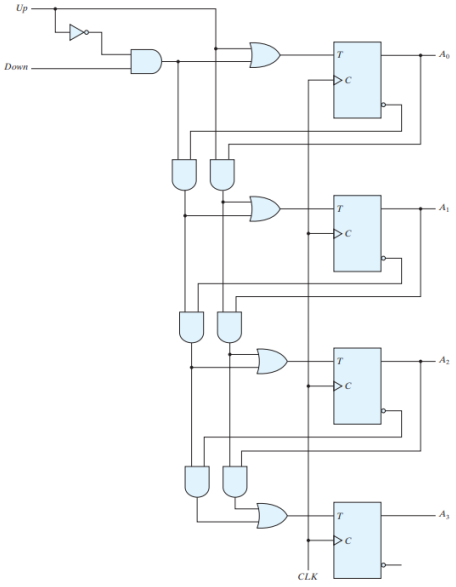


FIGURE 6.13
Four-bit up–down binary counter

A synchronous countdown binary counter goes through the binary states in reverse order, from 1111 down to 0000 and back to 1111 to repeat the count. A bit in any other position is complemented if all lower significant bits are equal to 0. For example, the next state after the present state of 0100 is 0011. The least significant bit is always complemented. The second significant bit is complemented because the first bit is 0. The third significant bit is complemented because the first two bits are equal to 0. But the fourth bit does not change, because not all lower significant bits are equal to 0. When the up input is 1, the circuit counts up, since the T inputs receive their signals from the values of the previous normal outputs of the flip-flops. When the down input is 1 and the up input is 0, the circuit counts down, since the complemented outputs of the previous flip-flops are applied to the T inputs. When the up and down inputs are both 0, the circuit does not change state and remains in the same count. When the up and down inputs are both 1, the circuit counts up. This set of conditions ensures that only one operation is performed at any given time. Note that the up input has priority over the down input.

BCD Counter : A BCD counter counts in binary-coded decimal from 0000 to 1001 and back to 0000. The flip-flop input equations can be simplified by means of maps. The unused states for minterms 10 to 15 are taken as don't-care terms. The simplified functions are $T_{Q1} = 1$, $T_{Q2} = Q'_8 Q_1$, $T_{Q4} = Q_2 Q_1$, $T_{Q8} = Q_8 Q_1 + Q_4 Q_2 Q_1$, $y = Q_8 Q_1$

Table 6.5
State Table for BCD Counter

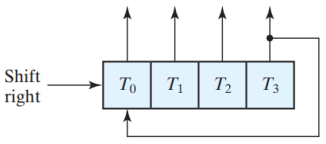
Present State				Next State				Output	Flip-Flop Inputs			
Q_8	Q_4	Q_2	Q_1	Q_8	Q_4	Q_2	Q_1	y	TQ_8	TQ_4	TQ_2	TQ_1
0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	1	0	0	1	0	0	0	0	1	1
0	0	1	0	0	0	1	1	0	0	0	0	1
0	0	1	1	0	1	0	0	0	0	1	1	1
0	1	0	0	0	1	0	1	0	0	0	0	1
0	1	0	1	0	1	1	0	0	0	0	1	1
0	1	1	0	0	1	1	1	0	0	0	0	1
0	1	1	1	1	0	0	0	0	1	1	1	1
1	0	0	0	1	0	0	1	0	0	0	0	1
1	0	0	1	0	0	0	0	1	1	0	0	1

The circuit can easily be drawn with four T flip-flops, five AND gates, and one OR gate. Synchronous BCD counters can be cascaded to form a counter for decimal numbers of any length.

Ring Counter : Timing signals that control the sequence of operations in a digital system can be generated by a shift register or by a counter with a decoder. A ring counter is a circular shift register with only one flip-flop being set at any particular time; all others are cleared(set to 0). The single bit is shifted from one flip-flop to the next to produce the sequence of timing signals. Each flip-flop is in the 1 state once every four clock cycles and produces one of the four timing signals.

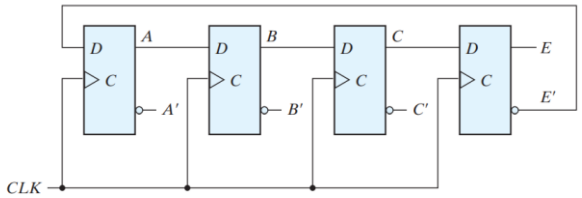
To generate 2^n timing signals, we need either a shift register with 2^n flip-flops or an n -bit binary counter together with an n -to-2^n -line decoder. ←

It is also possible to generate the timing signals with a combination of a shift register and a decoder. That way, the number of flip-flops is less than that in a ring counter, and the decoder requires only two-input gates. This combination is called a *Johnson counter* .



(a) Ring-counter (initial value = 1000)

Johnson Counter : A k -bit ring counter circulates a single bit among the flip-flops to provide k distinguishable states. The number of states can be doubled if the shift register is connected as a switch-tail ring counter. A switch-tail ring counter is a circular shift register with the complemented output of the last flip-flop connected to the input of the first flip-flop.



(a) Four-stage switch-tail ring counter

Sequence number	Flip-flop outputs				AND gate required for output
	A	B	C	E	
1	0	0	0	0	$A'E'$
2	1	0	0	0	AB'
3	1	1	0	0	BC'
4	1	1	1	0	CE'
5	1	1	1	1	AE
6	0	1	1	1	$A'B$
7	0	0	1	1	$B'C$
8	0	0	0	1	$C'E$

(b) Count sequence and required decoding

A Johnson counter is a k -bit switch-tail ring counter with 2 k decoding gates to provide outputs for 2 k timing signals. The eight AND gates listed in the table, when connected to the circuit, will complete the construction of the Johnson counter. Since each gate is enabled during one particular state sequence, the outputs of the gates generate eight timing signals in succession.

For example, sequence 7 has an adjacent 0, 1 pattern in flip-flops B and C . The decoded output is then obtained by taking the complement of B and the normal output of C, or B'C.

Johnson counters can be constructed for any number of timing sequences. The number of flip-flops needed is one-half the number of timing signals. The number of decoding gates is equal to the number of timing signals, and only two-input gates are needed.

Note : Total number of states in any counter is 2^4, now if sequence is given in which k states are used so number of unused states becomes $2^4 - k$.

From table b you can see No. of unused state in Johnson counter is 2^n-2n.

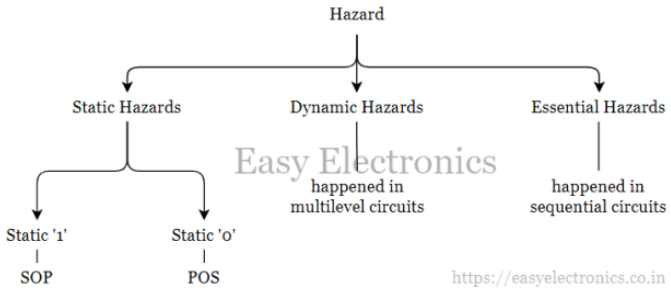
Output frequency of a mod N counters is $f = \frac{f_{in}}{N}$

The output frequency of a decode counter (i.e. mod-10 counters)

$$= \frac{50\ \text{kHz}}{10} = 5\ \text{KHz}$$

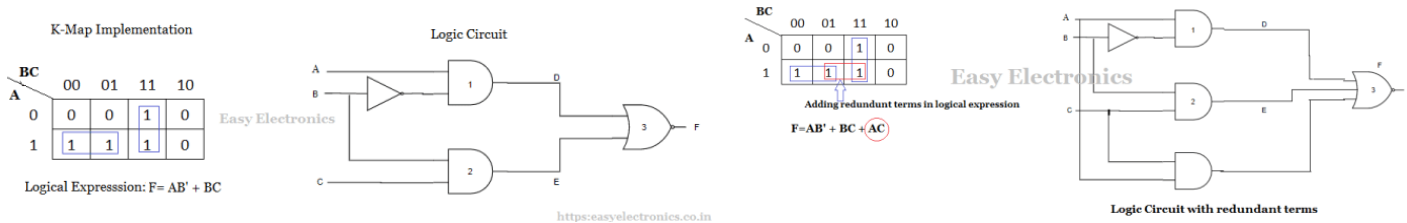
6.5) Hazards in Digital electronics :

Whenever undesirable or unwanted transition in the output signal of digital circuits then we can call them Hazard. This is due to different propagation delays of different paths. They will be very short in duration, like a glitch that will be removed after some time.



6.5.1) Static Hazard **:** Static hazard is those where the signal level should have been constant but it changes for a small amount of time. If any static hazard comes in digital circuits then both static 1 and static 0 hazard will come in the circuit simultaneously. Only one of them will not generate in a digital circuit.

Example :



If A = C = 1 and B = 1 so B appear at first AND gate after some time as it passes through NOT gate so there will be wrong reading at the output. Now to eliminate static hazard from the digital circuit then you have to add the redundant terms in the logical expression. Static hazard can be eliminated by adding redundant terms which increases the hardware but removes the glitch.

6.5.2) Dynamic hazard : This occurs during a multilevel circuit where the output must make a transition from 0 to 1 or from 1 to 0 but the output makes multiple transitions then settles to a final value. If all static hazards are eliminated from a circuit, then dynamic hazards cannot occur.

6.5.3) Essential Hazard : This occur in asynchronous sequential circuits. This type of hazard is caused by unequal delays along two or more paths that originate from an equivalent input.

NOTE :

1) In synchronization, there is a less chance of hazards but it can increase the delay. Then the advantage is ease of avoiding problems due to hazards.
2) While solving flip flop questions make input output table.
3) Base 10 adder means you have to add according to base 10 it means when you have (10)6+(34)4 and if you add in base 10 adder you will get 44 so you need not to convert 10 and 34 to base 10 and then add, remember that.

4) Number of m bit MUX/DeMUX/Decoder/Encoder to construct n bit MUX/DeMUX/Decoder/Encoder is: ceil (n-1)/(m-1). Here n and m represent outputs not inputs.
5) NOR gate, NAND gate, Multiplexers and Half adders can also be used to realise all digital circuits.
6) For ripple counter frequency = 1/(n*delay per flipflop) … Where n is number of flipflop.

Questions on Synchronous circuits, Register and Counter :

1) A modulus -12 ring counter requires a minimum of

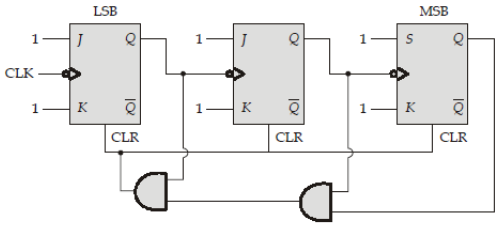
10 flip-flops
12 flip-flops
8 flip-flops
6 flip-flops

Answer : B

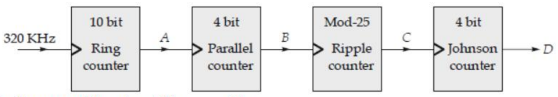
2) A 0 to 6 ripple counter is build using negative edge triggered J-K flip-flops with active high clear inputs and a combination of 2-input gates. All flip-flops J-K inputs are connected to 1. The combinational circuit consists of

One AND gate
One OR gate
One AND gate and One OR gate
Two AND gate

Answer :



As 0 to 6 ripple counter is given we need 3 bits to represent 0 to 6 numbers. We need to switch 111 to 000 directly so we need to connect three input to the and gate but here two input gate is allowed so, 2 and gate is correct answer.



The frequency of the pulse at *D* is _____ Hz.

10

Solution :
10

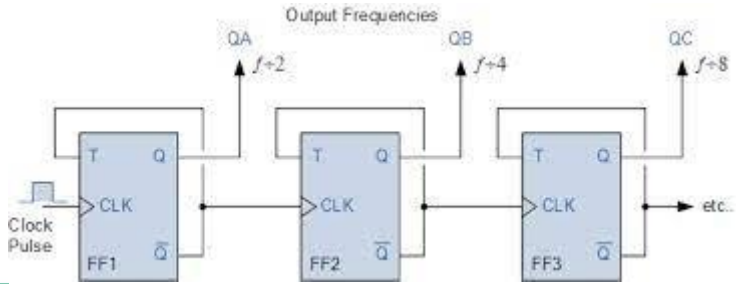
$$\text{Frequency of output} = \frac{\text{Frequency at input}}{\text{Mod value}}$$

$$\text{Frequency of } A = \frac{320\text{ KHz}}{10} = 32\text{ KHz} \quad \text{[10 bit ring counter is MOD 10]}$$

$$\text{Frequency of } B = \frac{32\text{ KHz}}{16} = 2\text{ KHz} \quad \text{[4 bit parallel counter is MOD 16]}$$

$$\text{Frequency of } C = \frac{2\text{ KHz}}{25} = 80\text{ Hz}$$

$$\text{Frequency of } D = \frac{80\text{ Hz}}{8} = 10\text{ Hz} \quad \text{[4 bit Johnson counter is MOD 8]}$$



same as this question If flip flop is given then output frequency will be half of the input.