CHAPTER 4

BOOLEAN FUNCTIONS AND DIGITAL CIRCUITS

4.1 Canonical Forms

4.1.1 Canonical Sum-of-Products

In Chapter 3, truth tables and Boolean functions are used to describe the functions of digital circuits. Truth tables can be constructed easily from Boolean functions. In this section, the conversion of a truth table to a Boolean function in standard or canonical forms is introduced.

A function F of two variables is described by a truth table in Table 4.1(a). When the function is implemented as a digital circuit, the variables are the inputs and the function value is the output. Table 4.1(a) is similar to that of an AND operation. Among the four combinations of values for A, B, only one produces a value of 1 for F. But this combination is not 11. To transform this table into another table so that F is equal to 1 when the combination of values is 11, the value for B is complemented such that X = B', which is shown in Table 4.2(b). Thus

$$F = AX = AB'$$

Table 4.1 (a) A truth table for two variables. (b) Conversion of (a) to a table for AND.

(a)

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

X = B' →

(b)

A	X	F
0	1	0
0	0	0
1	1	1
1	0	0

Table 4.2 is an example of three variables. To convert Table 4.2(a) to a truth table for a 3-variable AND operation, both A and B are complemented and replaced with X and Y in Table 4.2(b).

$$F = XYC = A'B'C$$

From the two examples, it is seen that the product can actually be written from the values of the variables that produce a value of 1 for F without any transformation. Each variable appears once in the product either in true form or complemented form. The

variable is in true form if its value is 1. It is in complemented form if the value is 0. A product has n literals if F is a function of n variables. Such a product is called a canonical product or standard product. When more than one input combination in a truth table produces a function value of 1, each combination produces one canonical product and the corresponding Boolean function for the truth table is a sum of all such products.

Table 4.2 (a) A truth table for three variables. (b) Conversion of (a) to a table in comparison with AND.

(a)

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

$X = A', \; Y = B'$ ⇒

(b)

X	Y	C	F
1	1	0	0
1	1	1	1
1	0	0	0
1	0	1	0
0	1	0	0
0	1	1	0
0	0	0	0
0	0	1	0

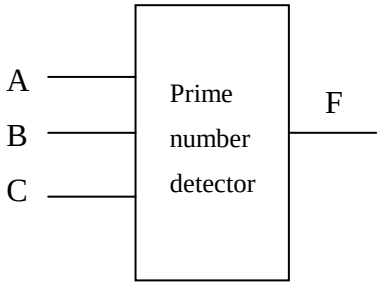


Figure 4.1 Block diagram for a prime number detector.

Table 4.3 Truth table for prime number detector.

A	B	C	F(A,B,C)
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Figure 4.1 is the block diagram of a circuit called a prime number detector. The input combination is a binary number ABC. If ABC is a prime number or decimal 1, the output F is equal to 1; otherwise F = 0. The truth table for the prime number detector is given in Table 4.3. There are five input states that produce a function value of 1. These

input states are 001, 010, 011, 101, and 111. Their respective canonical products are A'B'C, A'BC', A'BC, AB'C, and ABC.

Thus $F(A,B,C) = A'B'C + A'BC' + A'BC + AB'C + ABC$

The sum-of-products expression in which all products are canonical is called a canonical or standard sum-of-products expression.

Table 4.4 List of canonical products for the prime number detector.

A B C		Canonical product	Minterm
Decimal	Binary		
1	0 0 1	A' B' C	m_1
2	0 1 0	A' B C'	m_2
3	0 1 1	A' B C	m_3
5	1 0 1	A B' C	m_5
7	1 1 1	A B C	m_7

Since each canonical product is derived from a combination of input values, it can be easily represented by these input values using their decimal equivalents. If the decimal equivalent of this combination of input values is i, its corresponding canonical product is called minterm i, or m_i. "i" is a minterm number. For instance, AB'C is obtained from ABC = 101, which is equal to decimal 5, AB'C can be represented as m_5. The canonical products and their corresponding minterms and input values in both binary and decimal are listed in Table 4.4. The canonical sum-of-products expression can be re-written as

$$F(A,B,C) = m_1 + m_2 + m_3 + m_5 + m_7$$

The expression is called a sum-of-minterms. It can further be simplified to

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

Such a representation of a Boolean function is called a minterm list representation.

4.1.2 Canonical Product-of-Sums Expressions

This section shows how a product-of-sums expression can be derived from the combinations of input values that produce a function value of 0. Table 4.5 is a truth table for F', the complement of the function in Table 4.3. There are three minterms for F'. The minterms for F' come from those combinations of input values that produce a function

value of 1. A sum-of-minterms expression is readily written from the table for F'. The purpose is to derive an expression for F, not F'. Therefore, the sum-of-minterms has to be complemented.

Table 4.5 Truth table for prime number detector.

A B C	F'(A,B,C)	Minterm for F'
0 0 0	1	m_0
0 0 1	0	
0 1 0	0	
0 1 1	0	
1 0 0	1	m_4
1 0 1	0	
1 1 0	1	m_6
1 1 1	0	

As shown in the following derivation, taking the complement of F' results in a product of m_0', m_4', and m_6' for F. Each of the minterms is replaced with a canonical product. The complement of a canonical product becomes a sum. It is called a canonical or standard sum because each variable, either in true form or complemented form, appears once. A product-of-sums expression is said to be standard or canonical if all the sums are standard or canonical.

$$F'(A,B,C) = m_0 + m_4 + m_6$$

$$F(A,B,C) = (m_0 + m_4 + m_6)'$$

$$= m_0' \bullet m_4' \bullet m_6'$$

$$= (A'B'C')' \bullet (AB'C')' \bullet (ABC')'$$

$$= (A + B + C) \bullet (A' + B + C) \bullet (A' + B' + C)$$

$$= M_0 \bullet M_4 \bullet M_6$$

$$= \pi M(0, 4, 6)$$

From the derivation, it is seen that a canonical sum can be obtained from a combination of input values that produces a function value of 0. A variable is in true form if its input value is equal to 0. On the other hand, the complemented form of a variable should be adopted if its value is equal to 1. Similar to minterms, a simpler notation can be used for a canonical sum. If a canonical sum is found from a combination of input values with a decimal equivalent of i, the canonical sum is called "maxterm i", or M_i. Therefore, a canonical product-of-sums can be expressed as a product-of-maxterms. In the last step of the derivation, a product-of-maxterms is shortened to a list of maxterm numbers. It is called a maxterm list representation.

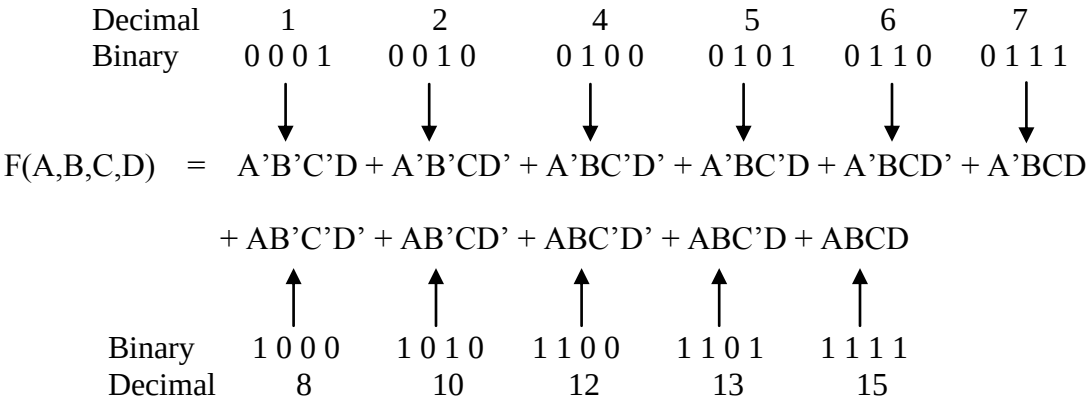
❖ Example 4.1

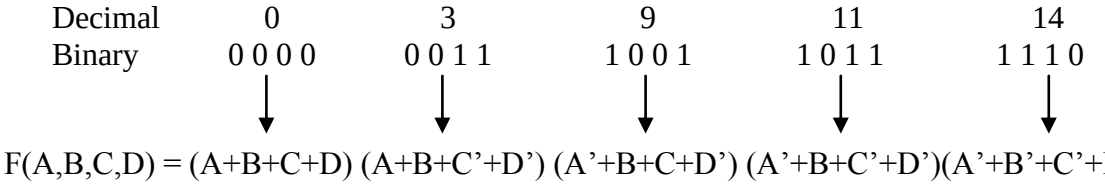
$$F(A,B,C,D) = \Sigma\, m(1, 2, 4, 5, 6, 7, 8, 10, 12, 13, 15)$$

The function value for an input combination can only be either 1 or 0, not both. Therefore, a decimal number for a binary input combination is either a minterm number or maxterm number, not both. Also the range of minterm and maxterm numbers for an n-variable function is from 0 to $2^n - 1$. From the minterm list representation given for F, the maxterm list representation is

$$F(A,B,C,D) = \pi\, M(0, 3, 9, 11, 14)$$

The canonical sum-of-products and canonical product-of-sums expressions are derived below.





Note that if a decimal minterm/maxterm number is the equivalent of a binary input combination of ABCD if F is expressed as a function of A, B, C, D. The decimal minterm/maxterm number may change if the order of the variables is changed. For example, if 7 is a minterm number for the input combination ABCD = 0111. By changing the order of the variables to D, C, B, A but without changing the values of the variables, it becomes m_{14} because DCBA = 1110. Thus the minterm list representation for the function in Example 4.1 is changed to the following if the function F is a function of A, C, B, D.

$$F(A,C,B,D) = \Sigma\, m(1, 4, 2, 3, 6, 7, 8, 12, 10, 11, 15)$$

4.1.3 Conversion to Canonical Forms

To convert a Boolean expression to a canonical sum-of-products expression, it is first changed to a sum-of-product expression and then expanded to the canonical form. An example is used to demonstrate the conversion.

❖ Example 4.2

Find the minterm list representation for

$$F(A,B,C,D) = A'B'C + BC' + AC'D + ABCD'$$

Since every variable appears once in a canonical product, a product with a missing variable x_0 may be ANDed with $(x_0' + x_0)$ and then multiplied out to two canonical products. For a product with m missing variables $x_0, x_1, \ldots, x_{m-1}$, the product is ANDed with $(x_0' + x_0)(x_1' + x_1) \ldots\ldots (x_{m-1}' + x_{m-1})$ and multiplied out to 2^m canonical products.

$A'B'C = A'B'C(D' + D) = A'B'CD' + A'B'CD = m_2 + m_3$

$AC'D = AC'D(B' + B) = AB'C'D + ABC'D = m_9 + m_{13}$

$BC' = BC'(A' + A)(D' + D) = A'BC'D' + A'BC'D + ABC'D' + ABC'D$

$= m_4 + m_5 + m_{12} + m_{13}$

$ABCD' = m_{14}$

The minterm numbers can also be found directly from a method shown in Table 4.6 without using Boolean algebra. For a product with (n-m) literals, assign a value of 0 to a literal in complemented form and a value of 1 to a literal in true form. As for the m missing variables, fill in all the possible combinations of values. There are 2^m combinations for m missing variables. Values for the missing variables in a product are placed within a rectangle in Table 4.6.

$$F(A,B,C,D) = (m_2 + m_3) + (m_9 + m_{13}) + (m_4 + m_5 + m_{12} + m_{13}) + m_{14}$$
$$= \Sigma\, m(2, 3, 4, 5, 9, 12, 13, 14)$$

Similar approach can be used to obtain the canonical product-of-sums or product-of-maxterms for a Boolean expression. The expression should start with a product-of-sums expression.

Table 4.6 Conversion of products to minterms.

Product	Variable A B C D	Minterm number
A B C D'	1 1 1 0	14
A' B' C	0 0 1 0	2
	0 0 1 1	3
A C' D	1 0 0 1	9
	1 1 0 1	13
B C'	0 1 0 0	4
	0 1 0 1	5
	1 1 0 0	12
	1 1 0 1	13

❖ Example 4.3

$$F(A,B,C,D) = (B' + C + D)\,(A + B')\,(A' + D')$$

To find the maxterm list representation of the above expression, the approach in Example 4.2 can be used to find the minterm numbers by first multiplying out the product-of-sums to a sum-of products expression.

$$\begin{aligned} F(A,B,C,D) &= (B' + C + D)\,(A + B')\,(A' + D') \\ &= (B' + C + D)\,(AD' + A'B') \\ &= AB'D' + ACD' + A'B' + A'B'C + A'B'D \\ &= AB'D' + ACD' + A'B' \end{aligned}$$

The respective minterms for the three products are $(m_8 + m_{10})$, $(m_{10} + m_{14})$, and $(m_0 + m_1 + m_2 + m_3)$. The minterm list representation is

$$F(A,B,C,D) = \Sigma\, m(0, 1, 2, 3, 8, 10, 14)$$

or
$$F(A,B,C,D) = \pi\, M(4, 5, 6, 7, 9, 11, 12, 13, 15)$$

The maxterm list representation can also be obtained directly from the product-of-sums expression without getting first the minterms. A sum with m missing variables is equivalent to the product of 2^m canonical sums or maxterms.

$$B' + C + D = A'A + B' + C + D = (A' + B' + C + D)\,(A + B' + C + D) = M_{12}\,M_4$$

$$\begin{aligned} A + B' &= A + B' + C'C + D'D = (A + B' + C' + D'D)\,(A + B' + C + D'D) \\ &= (A + B' + C' + D')(A + B' + C' + D)(A + B' + C + D')(A + B' + C + D) \\ &= M_7\,M_6\,M_5\,M_4 \end{aligned}$$

$$\begin{aligned} A' + D' &= A + B'B + C'C + D' = (A' + B' + C'C + D')\,(A' + B + C'C + D') \\ &= (A' + B' + C' + D')(A' + B' + C + D')(A' + B + C' + D')(A' + B + C + D') \\ &= M_{15}\,M_{13}\,M_{11}\,M_9 \end{aligned}$$

$$\begin{aligned} F(A,B,C,D) &= (M_{12}\,M_4)\,(M_7\,M_6\,M_5\,M_4)\,(M_{15}\,M_{13}\,M_{11}\,M_9) \\ &= \pi\, M(4, 5, 6, 7, 9, 11, 12, 13, 15) \end{aligned}$$

Table 4.7 Conversion of sums to maxterm numbers.

Sum	Variable A	B	C	D	Maxterm number
B'+C+D	0	1	0	0	4
	1	1	0	0	12
A+B'	0	1	0	0	4
	0	1	0	1	5
	0	1	1	0	6
	0	1	1	1	7
A'+D'	1	0	0	1	9
	1	0	1	1	11
	1	1	0	1	13
	1	1	1	1	15

Similar to Example 4.2, the maxterm numbers can also be found without using Boolean algebra. To find the maxterms of a sum with (n-m) literals, assign a value of 1 to a literal in complemented form and a value of 0 to a literal in true form. The values for the missing variables consist of all the 2^m combinations. Therefore each sum is equivalent to the product of 2^m maxterms. Table 4.7 shows the maxterm numbers obtained from the three sums in F.

In general, the minterm list representation of an n-variable Boolean function can be expressed as follows:

$$F(x_{n-1}, x_{n-2}, \ldots, x_2, x_1, x_0) = \sum_{0}^{2^n - 1} a_i m_i \qquad (4.1)$$

where $a_i \varepsilon (0, 1)$ and is called a minterm coefficient. $a_i = 1$ if i is a minterm number. For the function in Example 4.3, the values of the coefficients are as follows:

$$a_0 = a_1 = a_2 = a_3 = a_8 = a_{10} = a_{14} = 1$$

$$a_4 = a_5 = a_6 = a_7 = a_9 = a_{11} = a_{12} = a_{13} = a_{15} = 0$$

4.2 Incompletely Specified Functions

Certain combinations of input values to a digital circuit sometimes may never occur. Under such circumstances, the values at the outputs are immaterial. It can be either 0 or 1. An "x" or “d” instead of 0 or 1 is written in the truth table for the function values of such input combinations. Since the outputs are not specified for those input combinations, the function for such a circuit is called an incompletely specified function. An input state with an unspecified function value is called a don't-care term. A don’t-care term has the option of becoming either a minterm or a maxterm.

As shown in Figure 4.1, the prime number detector has four inputs w, x, y, and z. wxyz is a binary-coded-decimal (BCD) code. Therefore the last six combinations of wxyz, which are 1010, 1011, 1100, 1101, 1110, and 1111, are not valid BCD codes and should never become inputs to the circuit. The truth table is given in Table 4.8. The minterm list representation is

$$F(w,x,y,z) = \Sigma\, m(1, 2, 3, 5, 7) + d(10, 11, 12, 13, 14, 15)$$

The second list on the right-hand-side of the above equation is for don’t-care term numbers. They should be listed separately from the minterm numbers. Neither can they be omitted. They should also be included in the maxterm list representation.

$$F(w,x,y,z) = \pi\, M(0, 4, 6, 8, 9) \bullet D(10, 11, 12, 13, 14, 15)$$

After a circuit is designed or built for an incompletely specified function, the output will be either 0 or 1 for each input combination, including those combinations of input values that should never occur. A don't-care term becomes either a minterm or maxterm during the design process or minimization process for an incompletely specified function. In other words, each "x" is assigned a value of either 0 or 1, depending on how the design can be minimized.

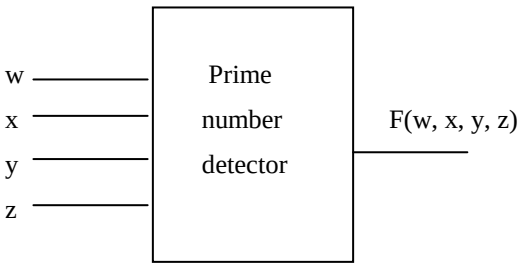


Figure 4.2 Block diagram for prime number detector

Table 4.8 Truth table for the prime number detector in Figure 4.2.

w	x	y	z	F(w,x,y,z)
0	0	0	0	0
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	0
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	0
1	0	0	1	0
1	0	1	0	d
1	0	1	1	d
1	1	0	0	d
1	1	0	1	d
1	1	1	0	d
1	1	1	1	d

To show how don't-care terms can be utilized to minimize Boolean functions, the prime number detector is first designed based on the minterms only. The design will then be compared with one that includes don't-care terms.

$$
\begin{aligned}
F(w,x,y,z) &= \Sigma\, m(1, 2, 3, 5, 7) \\
&= w'x'y'z + w'x'yz' + w'x'yz + w'xy'z + w'xyz \\
&= w'x'y'z + w'x'yz' + w'x'yz + w'x'yz + w'xy'z + w'xyz \\
&= (w'x'y'z + w'x'yz) + (w'x'yz' + w'x'yz) + (w'xy'z + w'xyz) \\
&= w'x'z + w'x'y + w'xz \\
&= w'x'y + w'z
\end{aligned}
$$

If the value of F is given a value of 1 for the input combinations 1010 and 1011, and all the other don't-care values remain 0. Two canonical products, wx'yz' and wx'yz, have

to be added to the sum-of-products expression. By including these two don't-care terms as minterms, F is further simplified by eliminating y from the product w'x'y.

$$\begin{aligned} F(w,x,y,z) &= w'x'y + w'z + wx'yz' + wx'yz \\ &= w'x'y + w'z + wx'y \\ &= x'y + w'z \end{aligned}$$

4.3 Expansion of Boolean Functions

A Boolean function can be expanded to or developed into two terms with respect to a variable using the following theorem known as Shannon's expansion theorem.

$$F(x_{n-1}, x_{n-2}, ..., x_{i+1}, x_i, x_{i-1},....., x_2, x_1, x_0) = x_i' F_{x_i = 0} + x_i F_{x_i = 1} \quad (4.2)$$

where $$F_{x_i = 0} = F(x_{n-1}, x_{n-2}, ..., x_{i+1}, x_i = 0, x_{i-1},....., x_2, x_1, x_0) \quad (4.3a)$$

and $$F_{x_i = 1} = F(x_{n-1}, x_{n-2}, ..., x_{i+1}, x_i = 1, x_{i-1},....., x_2, x_1, x_0) \quad (4.3b)$$

are called sub-functions of F and x_i is called an expansion variable. The complemented form of the expansion variable is associated with the sub-function $F_{x_i = 0}$ and the true form of the expansion variable with $F_{x_i = 1}$.

Table 4.9 Proof of Shannon's expansion theorem.

x_i	Left-hand-side of (4.2)	Right-hand-side of (4.2)
0	$F(x_{n-1}, x_{n-2}, ..., x_{i+1}, x_i = 0, x_{i-1},....., x_2, x_1, x_0) = F_{x_i = 0}$	$(0)'F_{x_i = 0} + (0)F_{x_i = 1} = F_{x_i = 0}$
1	$F(x_{n-1}, x_{n-2}, ..., x_{i+1}, x_i = 1, x_{i-1},....., x_2, x_1, x_0) = F_{x_i = 1}$	$(1)'F_{x_i = 0} + (1)F_{x_i = 1} = F_{x_i = 1}$

The 4-variable function in Example 4.2 is used as an illustration of expanding a function.

$$F(A,B,C,D) = \Sigma\, m(2, 3, 4, 5, 9, 12, 13, 14) = A'B'C + BC' + AC'D + ABCD'$$

The sum-of-products expression can be simplified using the elimination theorem as show below:

$$F(A,B,C,D) = A'B'C + BC' + AC'D + ABCD'$$

$$= A'B'C + AC'D + B(C' + ACD')$$
$$= A'B'C + AC'D + B(C' + AD')$$
$$= A'B'C + BC' + AC'D + ABD' \quad (4.4)$$

By selecting B as the expansion variable, the two sub-functions $F_{B=0}$ and $F_{B=1}$ are

$$F_{B=0} = F(A, B = 0, C, D)$$
$$= A'(0)'C + (0)C' + AC'D + A(0)D'$$
$$= A'C + AC'D \quad (4.5a)$$

and $$F_{B=1} = F(A, B = 1, C, D)$$
$$= A'(1)'C + (1)C' + AC'D + A(1)D'$$
$$= C' + AC'D + AD' = C' + AD' \quad (4.5b)$$

Thus $$F(A,B,C,D) = B'(\underline{A'C + AC'D}) + B(\underline{C' + AD'}) \quad (4.6)$$

where the sub-functions are underlined. The expansion may continue to be applied to the sub-functions. For instance, the two sub-functions $F_{B=0}$ and $F_{B=1}$ are expanded with C as the expansion variable.

The two sub-functions of $F_{B=0} = A'C + AC'D$ are

$$F_{BC=00} = F(A, B = 0, C = 0, D) = A'(0) + A(0)'D = AD \quad (4.7a)$$

$$F_{BC=01} = F(A, B = 0, C = 1, D) = A'(1) + A(1)'D = A' \quad (4.7b)$$

From Equations (4.7a) and (4.7b)

$$F_{B=0} = A'C + AC'D = C' (\underline{AD}) + C (\underline{A'}) \quad (4.8)$$

The two sub-functions of $F_{B=1} = C' + AD'$ are

$$F_{BC=10} = F(A, B = 1, C = 0, D) = (0)' + AD' = 1 \quad (4.9a)$$

$$F_{BC=11} = F(A, B = 1, C = 1, D) = (1)' + AD' = AD' \quad (4.9b)$$

From Equations (4.9a) and (4.9b)

$$F_{B=1} = C' + AD' = C'(\underline{1}) + C(\underline{AD'}) \quad (4.10)$$

By replacing the two sub-functions $F_{B=0}$ and $F_{B=1}$ in Equation (4.6) with Equations (4.8) and (4.10), F becomes

$$F(A,B,C,D) = B'(\underline{A'C + AC'D}) + B(\underline{C' + AD'})$$
$$= B' [C' (\underline{AD}) + C (\underline{A'})] + B [C'(\underline{1}) + C(\underline{AD'})]$$
$$= B'C' (\underline{AD}) + B'C (\underline{A'}) + BC'(\underline{1}) + BC(\underline{AD'})$$
$$= B'C' \underline{F_{BC = 00}} + B'C \underline{F_{BC = 01}} + BC' \underline{F_{BC = 10}} + BC \underline{F_{BC = 11}} \qquad (4.11)$$

Because a function (or sub-function) is expanded to two sub-functions using Shannon's expansion theorem, the expansion of a function with k variables generates 2^k sub-functions. For an n-variable function, each sub-function is a function of (n – k) variables. If an n-variable function is expanded with all the variables, each of the 2^n sub-function is a minterm coefficient, a constant of either 0 or 1. The expansion can be illustrated by a diagram called a binary tree. Equation (4.11) is used as an example for a binary tree. The function and each sub-function is represented by a dot that is called a node. Each line that connects two nodes is called a branch.

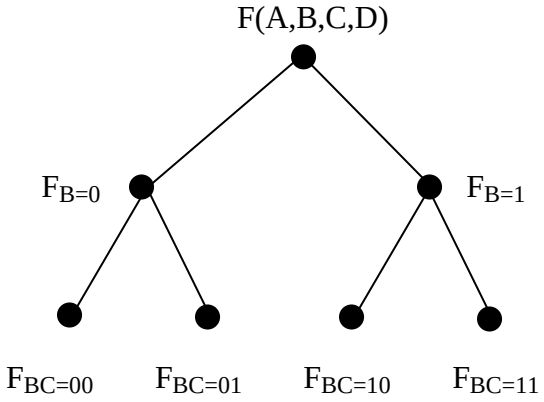


Figure 4.3 A binary tree for the expansion of a Boolean function.

4.4 Functionally Complete Set

Since a Boolean function can always be expressed as a sum-of-products or product-of-sums expression, it is sufficient to use only AND gates, OR gates, and inverters in implementing Boolean functions. Thus AND, OR, and NOT are called a functionally complete set. Two other logical operations, NAND and NOR, are introduced in this section. Each of them is by itself a functionally complete set.

NAND is a logical operation equivalent to the complement of AND. The device used to perform a NAND operation is called a NAND gate. The truth table for a 2-variable NAND is given in Table 4.9 and the logic symbol for a NAND gate is shown in Figure 4.4. The mathematical expression for NAND is

$$F(x,y) = (x\ y)'$$

In Figure 4.5(a), it shows that a NAND gate can be used as an inverter if both inputs are connected together because

$$x' = (x\ x)'$$

A 2-input NAND gate can also generate an output of x' if one input is x and the other input has a logic value of 1.

Table 4.9 Truth table for NAND.

x	y	F(x,y)
0	0	1
0	1	1
1	0	1
1	1	0

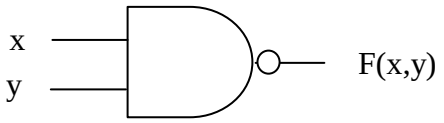


Figure 4.4 Logic symbol for NAND gate.

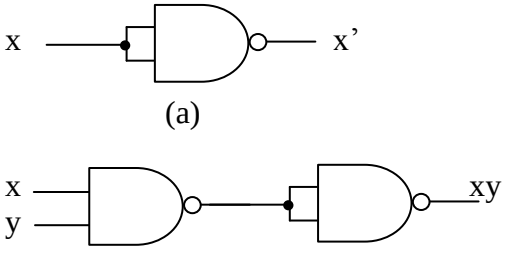
In Figure 4.5(b), two cascaded NAND gates are used to implement an AND operation, with the second NAND gate used as an inverter.

$$x\ y = [\ (\ x\ y\ )'\ ]'$$

By applying DeMorgan's theorem,

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

Therefore, OR can be implemented by a NAND gate with inverted inputs, as shown in Figure 4.5(c).



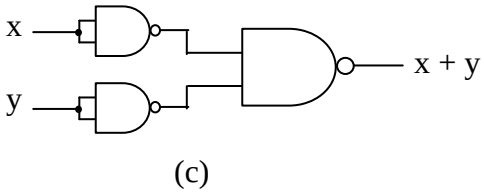


Figure 4.5 Implementation of NOT, AND, OR using NAND gates.

NOR is a logical operation equivalent to the complement of OR. The truth table for a 2-variable NOR function is given in Table 4.10. The algebraic expression for NOR is

$$F(x,y) = (x + y)'$$

NOR gate is a device used to implement the NOR operation. The logic symbol is shown in Figure 4.6. The realization of NOT, OR, AND by NOR gates are shown in Figure 4.7 because

$$x = (x + x)'$$
$$x + y = [(x + y)']'$$
$$x\,y = (x' + y')'$$

A 2-input NOR gate can also generate an output of x' if one input is x and the other input has a logic value of 0.

Table 4.10 Truth table for NOR.

x	y	F(x,y)
0	0	1
0	1	0
1	0	0
1	1	0

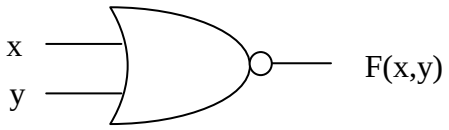


Figure 4.6 Logic symbol for NOR gate.

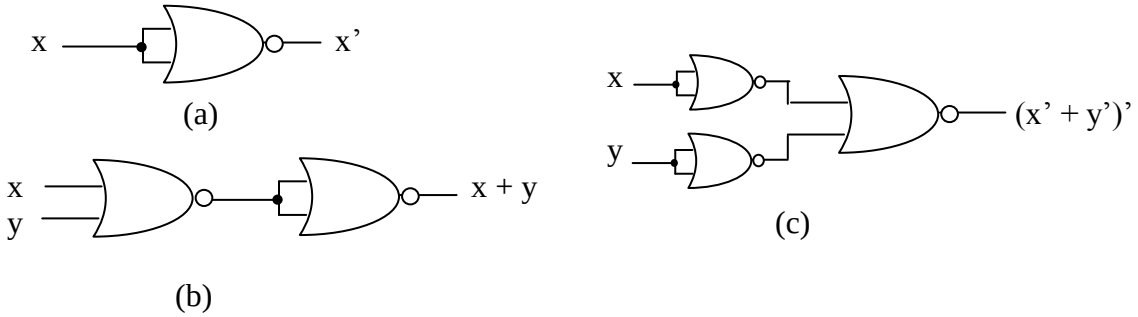


Figure 4.7 Implementation of NOT, OR, AND using NOR gates.

Since AND gates, OR gates, and inverters can be replaced with either NAND gates or NOR gates, NAND and NOR are each a functionally complete set.

4.5 Exclusive-OR and Exclusive-NOR

Two more logical operations are introduced in this section. The truth table for exclusive-OR (XOR) is given in Table 4.11. The function of XOR produces a value of 1 when the two variables have different values, and a value of 0 when the values of the

two variables are equal. The operation is implemented by an XOR gate. The logic symbol is shown in Figure 4.8. Unlike AND, OR, NAND, and NOR which can have more than two inputs, XOR gates have only two inputs.

Mathematically, the XOR function is expressed as follows:

$$F(x, y) = x \oplus y$$

Table 4.11 Truth table for EXCLUSIVE-OR.

x	y	F(x,y)
0	0	0
0	1	1
1	0	1
1	1	0

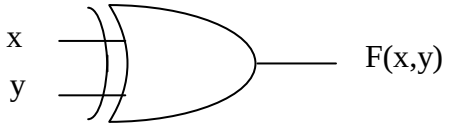


Figure 4.8 Logic symbol for Exclusive-OR gate.

Exclusive-NOR (XNOR) is the complement of exclusive-OR. As shown by the truth table in Table 4.12, the operation generates a value of 1 if the values of the two variables are the same, and a value of 0 if they have different values. Therefore, XNOR is also known as EQUIVALENCE. The logic symbol for an XNOR gate is shown in Figure 4.9. The algebraic expression for XNOR is

$$F(x, y) = (x \oplus y)' = x \odot y$$

Table 4.12 Truth table for Exclusive-NOR.

x	y	F(x,y)
0	0	1
0	1	0
1	0	0
1	1	1

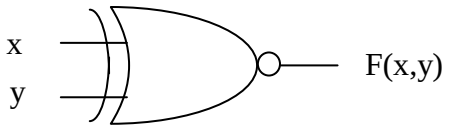


Figure 4.9 Logic symbol for Exclusive-NOR gate.

Some of the basic laws for XOR are given below.

(1) Law of 0 $x \oplus 0 = x$
(2) Law of 1 $x \oplus 1 = x'$
(3) Idempotency law $x \oplus x = 0$
(4) Complementary law $x \oplus x' = 1$
(5) Commutative law $x \oplus y = y \oplus x$

The canonical sum-of-products and canonical product-of-sums for XOR and XNOR can be obtained from Tables 4.11 and 4.12.

$$x \oplus y = x'y + xy' = (x' + y')(x + y)$$

$$x \odot y = (x \oplus y)' = x'y' + xy = (x' + y)(x + y')$$

The following identity can be derived mathematically or by logic reasoning.

$$(x \oplus y)' = x \oplus y' = x' \oplus y$$

If the operation is $x \oplus y$, $x \oplus y'$ and $x' \oplus y$ suggest that the value of one of the two inputs to an XOR gate is changed. If the values of x and y were equal, now they are different. If they were different, now they are equal. The change of one of the two values at the inputs definitely changes the value at the output. This is equivalent to changing the value of $(x \oplus y)$. That is, the output $(x \oplus y)$ has to change to $(x \oplus y)'$.

4.6 Timing Diagram and Propagation Delay

It has been illustrated so far that the function of a digital circuit can be described using words, a truth table, Boolean expressions, or circuit diagrams. This section introduces one more way of describing a digital circuit: timing diagram. A timing diagram shows the response of the circuit output due to temporal changes at the inputs. A 2-input AND gate is used as an example. The two inputs are A, B, and the output is AB.

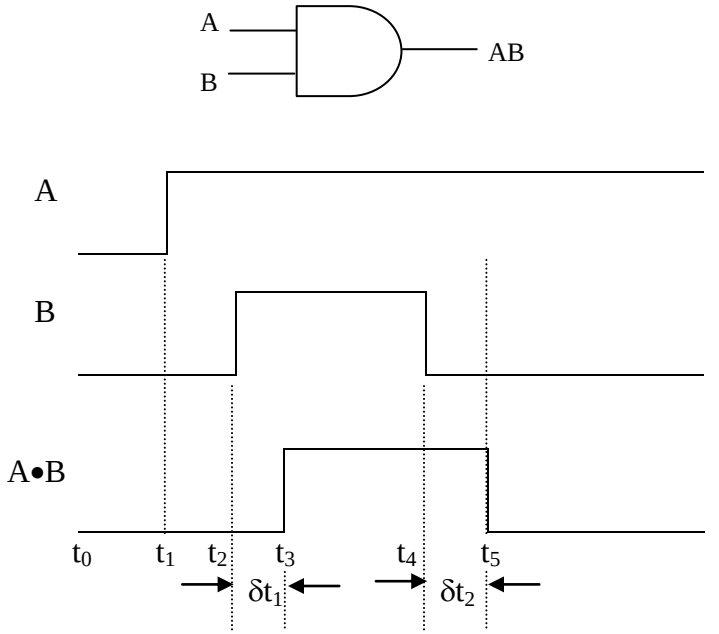


Figure 4.10 Timing diagrams for a 2-input AND gate with propagation delay.

As shown in Figure 4.10, both A and B are 0 at t_0 and the output AB is 0. At t_1, the value of A changes from 0 to 1 and AB is still 0. At t_2, B becomes 1 and AB is expected to become 1. However, the output does not become 1 until t_3, a delay of $t_3 - t_2 = \delta t_1$ seconds. At t_4, B returns to 0. Again, the output does not become 0 until t_5, a delay of $t_5 - t_4 = \delta t_2$ seconds. The time delay between the input change and the actual output change is called propagation delay. Propagation delays for the output transition from high to low and low to high are not the same. Figure 4.11 is the timing diagram for the AND gate without showing propagation delay. This is not a realistic, but rather idealistic, situation. For convenience, all the timing diagrams used in later chapters and sections will not show the delays unless otherwise stated.

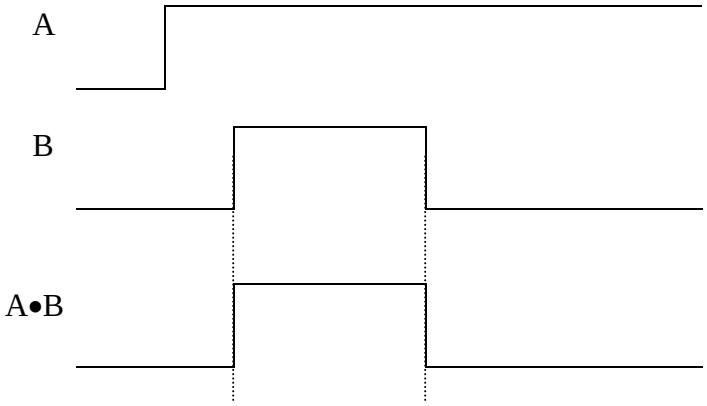


Figure 4.11 Timing diagram for a 2-input AND gate without propagation delay.

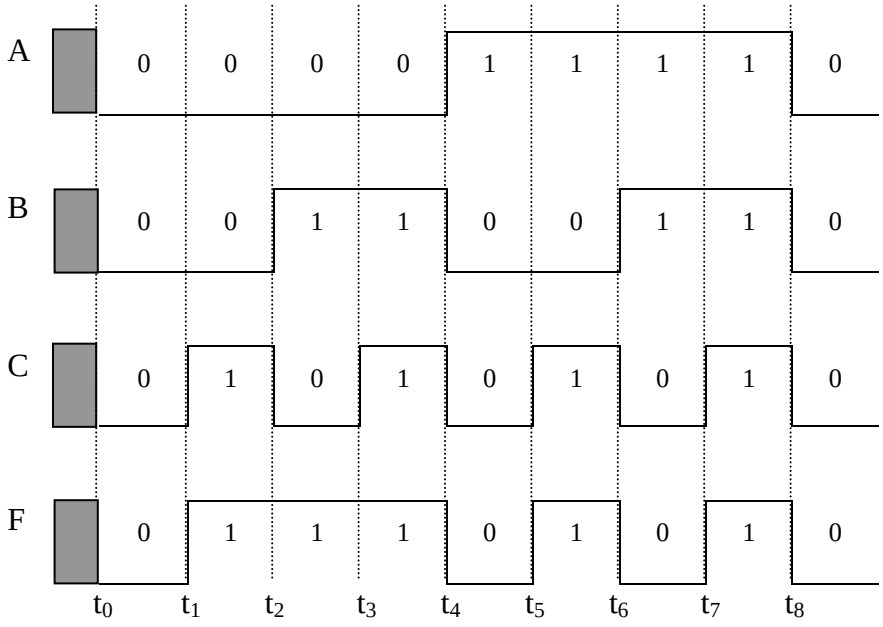


Figure 4.11 Timing diagrams for prime number detector.

Timing diagrams can be used to describe the behavior of a function or circuit. The truth table in Table 4.3 for a prime number detector is used as an example. As shown in Figure 4.12, the input changes to the circuit are in the same order as the values listed in the truth table.

4.7 Analysis of Combinational Circuits

In designing a digital circuit, a statement describing the function of the circuit is usually given. The process will end up with a circuit diagram. Analysis is the reverse process trying to find the function of a given circuit. The function of the circuit can be described by a Boolean expression, truth table, timing diagram, or verbally.

❖ Example 4.4

The circuit in Figure 4.13 is used as an example for analysis. Instead of writing one complex expression for F, some intermediate signals in the circuit, such as P_1, P_2, P_3, and P_4, are derived first.

$P_1 = BC' + D'$
$P_2 = (BD)' = B' + D'$
$P_3 = (A\ P_1)' = [A\ (BC' + D')]'$
$P_4 = (C\ P_2)' = [C\ (B' + D')]'$

$F(A,B,C,D) = (P_3\ P_4)'$
$= \{[A\ (BC' + D')]' \bullet [C\ (B' + D')]'\}'$
$= A\ (BC' + D') + C\ (B' + D')$
$= ABC' + AD' + B'C + CD'$

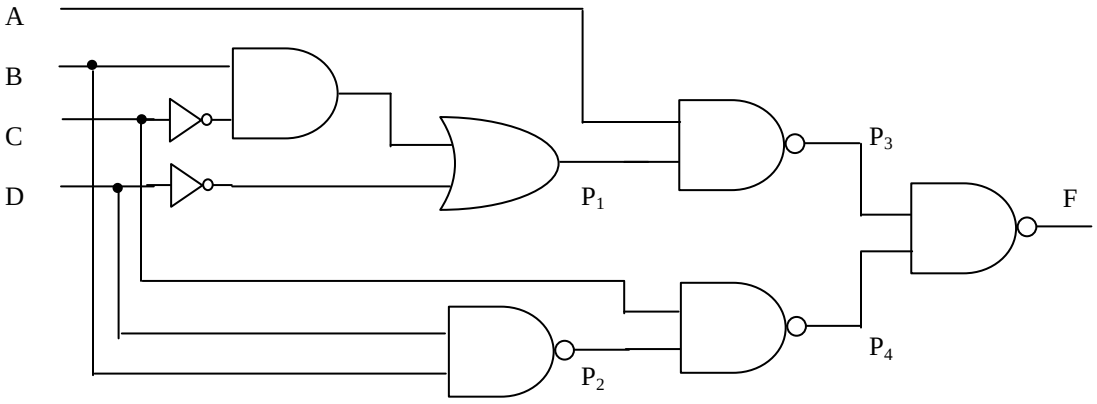


Figure 4.13 Circuit for Example 4.4.

When a circuit have internal inversions and inversions at the outputs, finding a Boolean expression in the form of sum-of-products or product-of-sums requires frequent applications of DeMorgan's theorem, which makes the analysis more tedious. Analysis can be made easier by first eliminating as many internal/output inversions as possible using the gate equivalencies derived from DeMorgan's theorem as shown in Figure 4.14. Elimination of inversions should start from the outputs and proceed towards the inputs.

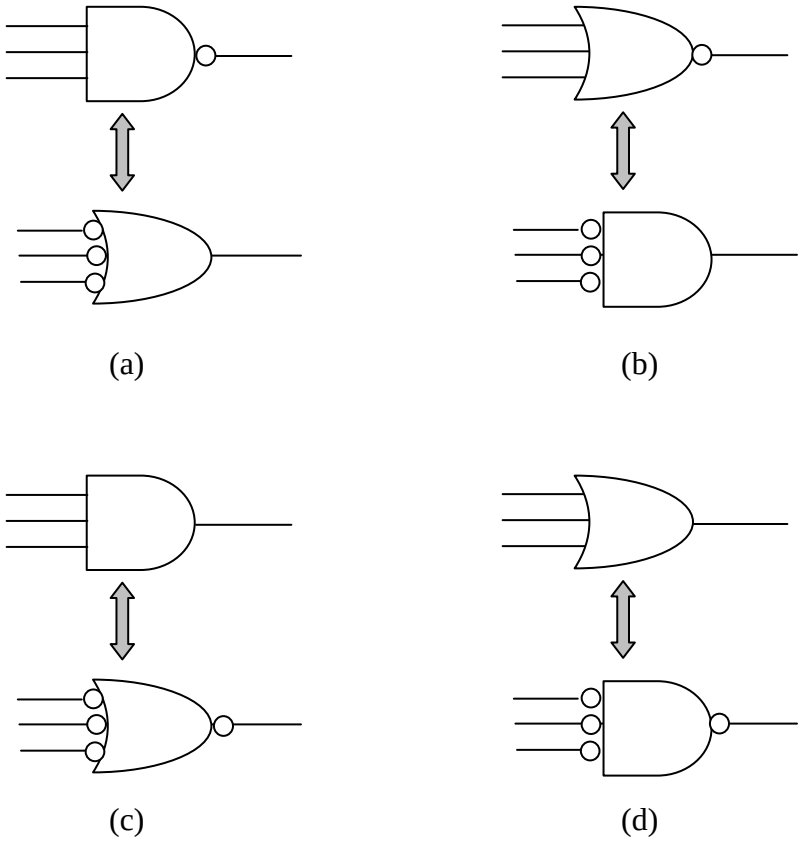


Figure 4.14 Gate equivalencies using DeMorgan's theorem.

Based on DeMorgan's theorem in Chapter 3, which is repeated below.

(a) $(x_1 \bullet x_2 \bullet x_3 \bullet \ldots\ldots\ldots \bullet x_{n-1} \bullet x_n)' = x_1' + x_2' + x_3' + \ldots\ldots\ldots + x_{n-1}' + x_n'$

(b) $(x_1 + x_2 + x_3 + \ldots\ldots\ldots + x_{n-1} + x_n)' = x_1' \bullet x_2' \bullet x_3' + \ldots\ldots\ldots \bullet x_{n-1}' \bullet x_n'$

A NAND gate is equivalent to an OR gate with all the inputs inverted. A NOR gate is the same as an AND gate with inverted inputs. The equivalencies are shown in Figures 4.14 (a) and 4.14(b) for 3-input gates. The applications of DeMorgan's theorem to AND and OR are shown in Figures 4.14(c) and 1.14(d).

❖ Example 4.5

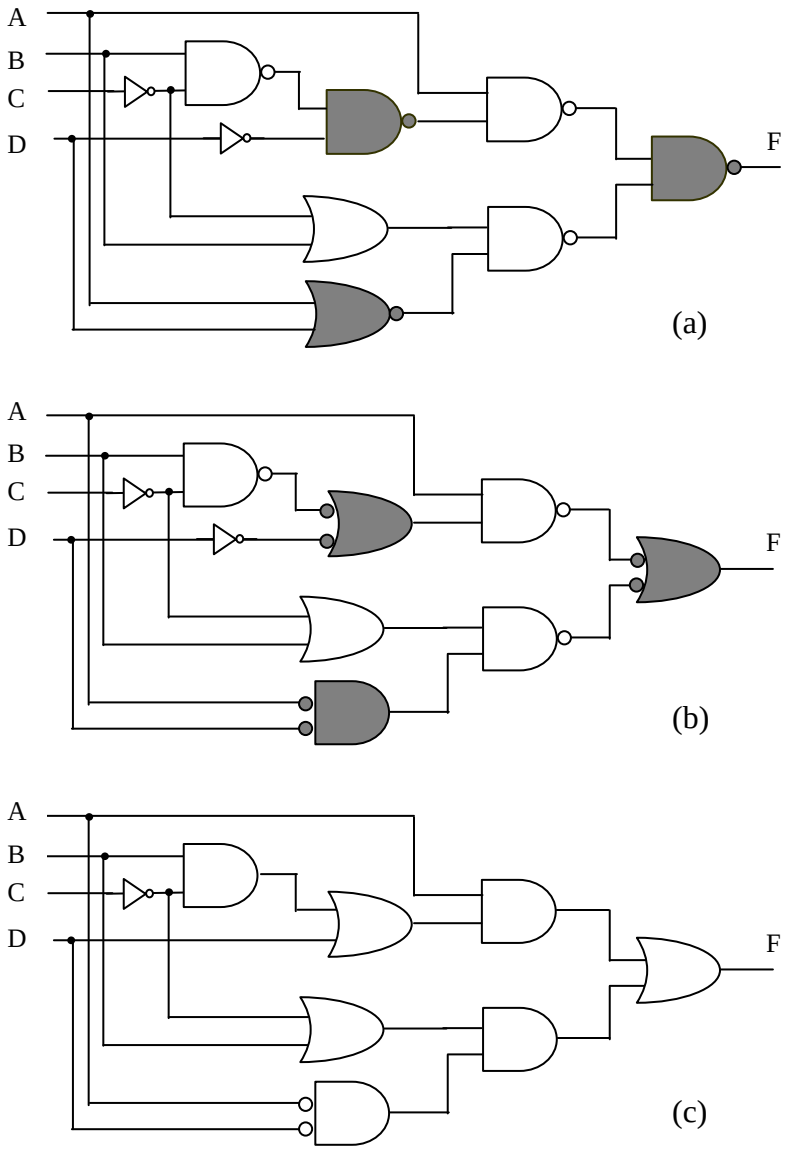


Figure 4.15 Circuit for Example 4.5.

To analyze the circuit in Figure 4.15(a), each of the two highlighted NAND gates are replaced with an OR gate with inverted inputs using the gate equivalency shown in Figure 4.14(a). The highlighted NOR gate is changed to an AND gate with inverted inputs as shown in Figure 4.14(b). The circuit after conversion is shown in Figure 4.15(b). Since two cascaded inversions offset each other, the circuit in Figure 4.15(b) is simplified by removing three pairs of inversion, which is shown in Figure 4.15(c). The circuit in Figure 4.15(c) does not have any internal and output inversion. A Boolean expression can be readily written and multiplied out without mathematical manipulations using DeMorgan's theorem.

F = A(D + BC') + A'D'(B + C')
= AD + ABC' + A'BD' + A'C'D'

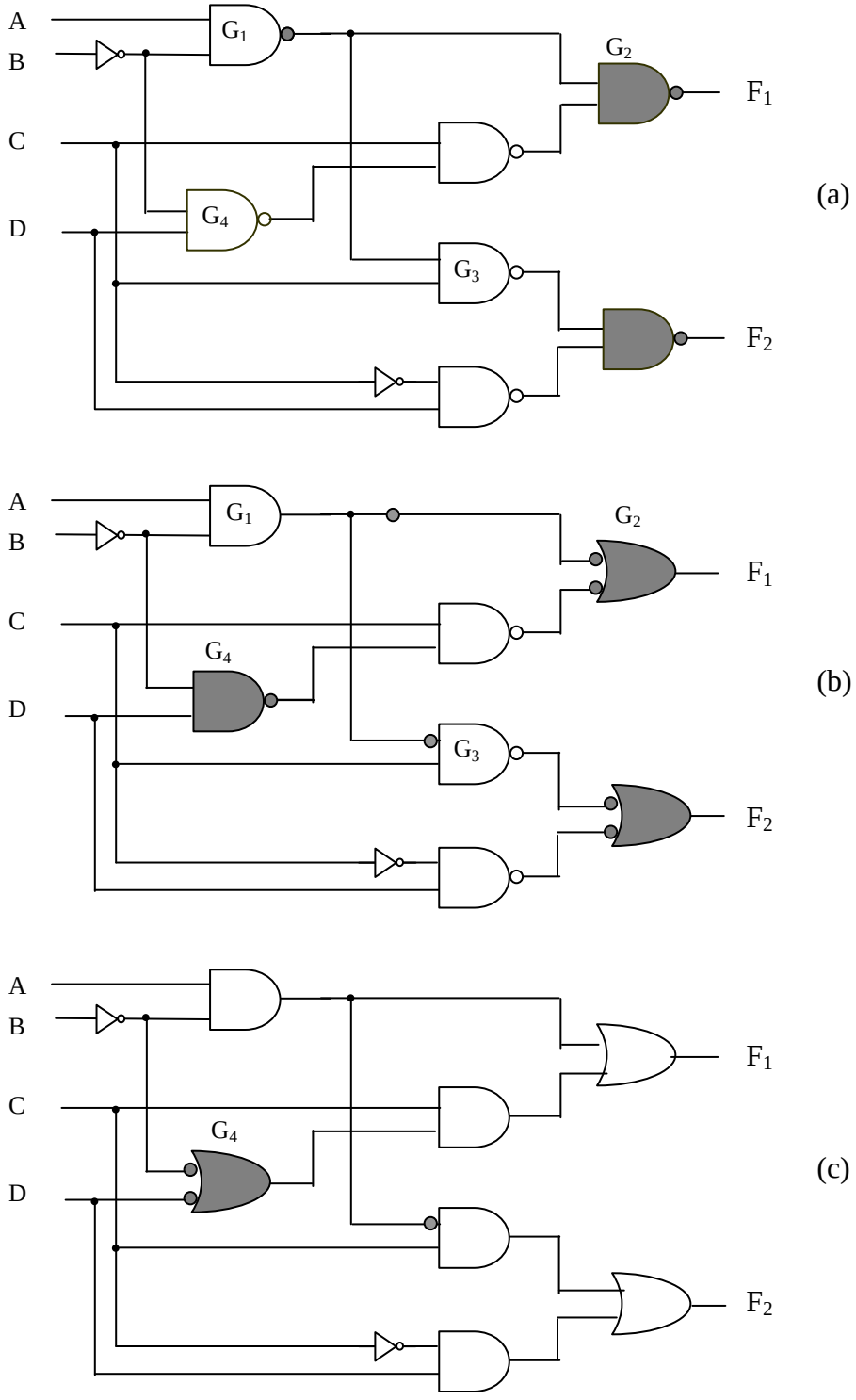


Figure 4.16 Circuit for Example 4.6.

❖ Example 4.6

To analyze the circuit in Figure 4.16(a), each of the two highlighted NAND gates at the outputs are replaced by an OR gate with inverted inputs. It is noted that the bubble of the NAND gate labeled as G_1 is highlighted. Because gate G_1 is connected to gates G_2 and G_3, it is not clear at this point if this bubble can be completely offset by another bubble without affecting the inputs to G_2 and G_3. Therefore the bubble is detached from G_1 and placed in each of the two paths to G_2 and G_3. The circuit after conversion is shown in Figure 4.16(b). It is seen that the bubble detached from G_1 and placed in the path to G_2 can be offset by one of the two input bubbles of G_2. The bubble to G_3 cannot be offset and will remain there. After removing all the other bubble pairs in Figure 4.16(b), only gate G_4 needs to be taken care of. As shown in Figure 4.16(c), G_4 can be replaced by a NOR gate with inverted inputs. The Boolean expressions for F_1 and F_2 can be easily obtained from Figure 4.16(c).

$$F_1 = AB' + (B + D')C = AB' + BC + CD$$

$$F_2 = (AB')'C + C'D = (A' + B)C + C'D = A'C + BC + C'D$$

4.8 Assertion and Signal Level

A descriptive name may be given to a signal in a logic circuit to indicate a certain condition or an operation to be executed. If the condition is true or the operation is to be executed, the signal is said to be asserted. When an asserted signal is 1, the signal is said to have an active-high signal level. If the assertion of a signal requires a value of 0 for the signal, the signal is called an active-low signal. When an active-high signal has a value of 0, it is said to be de-asserted. A de-asserted signal indicates that the condition is false or the operation is not to be executed. An active-low signal is de-asserted if it has a value of 1. The name of an active-low signal is usually preceded by a slash. A bubble is also placed at the input of a gate or the output of a circuit for an active-low signal.

An illustration of assertion and signal levels is given in Table 4.13 and Figure 4.17. Suppose there is a request to provide service from a device, such a printer. The service will be delivered if the device is not busy or is idle. The truth table is given in Table 4.13 (a) for active-high inputs and outputs. The function can be implemented by an AND gate as shown in Figure 4.17(a). Table 4.13(b) is the truth table for active-high inputs and active-low output. A slash is added to the signal name that becomes /Print and the values are inverted. The implementation is shown in Figure 4.17(b). The truth table for active-low inputs and active-high output is given in Table 4.13(c). This table is actually a truth table for NOR. A NOR gate can be used for implementation. The diagram shown in Figure 4.17(c) is an AND gate with inverted inputs, which is equivalent to a NOR gate.

$$(/Idle)' \bullet (/Request)' = (/Idle + /Request)'$$

The bubbles at the inputs /Idle and /Request indicate that the inputs are active-low signals. The absence of a bubble at the output suggests that the output is active-high. Similarly, a truth table for active-low inputs and active-low output is given in Table 4.13(d). The function can be implemented by an OR gate. But an AND gate with inverted inputs and output is drawn in Figure 4.17(d) to indicate that all the inputs and the output are active-low.

By examining the circuits in Figure 4.17, it is seen that an active-high signal can easily be converted to active-low by adding a bubble to the input/output and a slash in front of the name. An active-low signal can be changed to active-high by removing the bubble and the slash.

Table 4.13 Truth table for providing service by a printer.

(a)

Idle	Request	Print
0	0	0
0	1	0
1	0	0
1	1	1

(b)

Idle	Request	/Print
0	0	1
0	1	1
1	0	1
1	1	0

(c)

/Idle	/Request	Print
0	0	1
0	1	0
1	0	0
1	1	0

(d)

/Idle	/Request	/Print
0	0	0
0	1	1
1	0	1
1	1	1

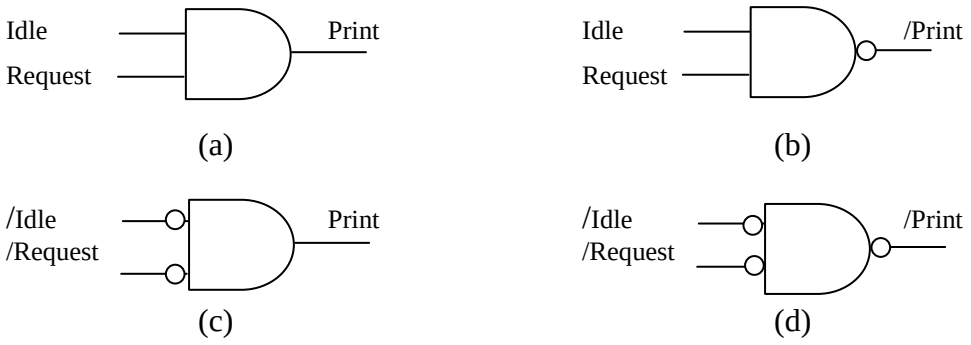


Figure 4.17 Implementation of the truth tables in Table 4.13.

PROBLEMS

1. Expand each of the following functions into a canonical sum-of-products expression.

   (a) F(x, y, z) = xy' + y'z' + x'
   (b) F(w, x, y, z) = x'y' + wxy' + w'yz'
   (c) F(A,B,C,D) = ABC + B'CD + C'D'
   (d) F(A,B,C,D) = A'B + B'CD + D'

2. Expand each of the following functions into a canonical product-of-sums expression.

   (a) F(x, y, z) = (x + y')(x' + z')
   (b) F(w, x, y, z) = (x + y' + z')(w + z)

3. Find the minterm and maxterm list forms for each of the following functions.

   (a) F(A,B,C,D) = A B + C D + A' B D
   (b) F(A,B,C,D) = (A + B + C') (A + C' + D') (A' + BD)
   (c) F(A,B,C,D) = (A + B)(C + B' + D)(C' + D)
   (d) F(A,B,C,D) = (B' + C')(A + B + C')D'
   (e) F(A,B,C,D,E) = (A' + B)(B' + C + D + E)(C' + E')
   (f) F(A,B,C,D,E) = (A ⊕ B)D' + BC'DE + AD'

4. Find the minterm and maxterm list forms for the complement of each of the functions in Problem 3.

5. Given f(w,x,y,z) = Σ m(0, 1, 2, 4, 6, 8, 10, 12, 14), find
   (a) the canonical sum-of-products expression for f.
   (b) the canonical product-of-sums expression for f.
   (c) the simplest sum-of-products expression for f.
   (d) the simplest product-of-sums expression for f.

6. Find the minterm list form for F(A,D,B,C) if

   F(A,B,C,D) = Σ m(0, 1, 3, 5, 7, 9, 10, 13, 14)

7. Find the minterm and maxterm list forms for f(a,b,c,d) if a b = 11 (both a and b are 1) never occur in input states.

   f(a,b,c,d) = a' b + a c d + a' d' + c' d

8. Given f(w,x,y,z) = Σ m (1,2,4,5,7,11,12,15)
   and g(w,x,y,z) = Σ m (0,1,2,7,8,9,12,14,15),

find the minterm list forms for

(a) f ‘	(b) g’	(c) f•g	(d) f + g,
(e) f ‘•g	(f) f •g’	(g) f ‘ + g’	(h) f ⊕ g

9. Simplify each of the following expressions to a sum-of-products expression.

   (a) a' ⊕ b' ⊕ c
   (b) a ⊕ (a b + c')
   (c) (a' ⊕ b ⊕ 1) (a ⊕ b)+ (c ⊕ c' ⊕ 0) (a + d e) (a + b')

10. Find the minterm list form for each of the following functions.

    (a) f(a,b,c,d) = a'b ⊕ b'cd ⊕ cd’
    (b) f(a,b,c,d) = (a' + b) ⊕ (c + d) ⊕ (a + c’ + d’)
    (c) f(a,b,c,d) = 1 ⊕ bcd ⊕ acd’

11. Given F(A,B,C,D) = (AB ⊕ C’D) + BD + A’B’CD’
    find all the sub-function of F with A and D as expansion variables. Express each sub-function as a simplest sum-of-products expression.

12. Find the simplest sum-of-products expression for F(A,B,C,D,E) using the following sub-functions.

    $F_{BD = 00}$ = A’ + E’
    $F_{BD = 01}$ = AE’
    $F_{BD = 10}$ = 0
    $F_{BD = 11}$ = A(C + E’)

13. Find the simplest sum-of-products expression for the sub-function $F_{AC = 00}$ of a 5-variable function F(A,B,C,D,E) if

    $F_{ACD = 000}$ = B’E
    $F_{ACD = 001}$ = B + E

14. Find the minterm list form for F(A,B,C,D,E) using the following sub-functions.

    $F_{AB = 00}$ = DE + C’D’
    $F_{AB = 01}$ = (C + D’)E
    $F_{AB = 10}$ = C + D’E
    $F_{ABD = 110}$ = C
    $F_{ABD = 111}$ = C + E

15. Given in Figure P4.1 is the timing diagrams of a Boolean function F(A,B,C), find the simplest product-of-sums expression for F.

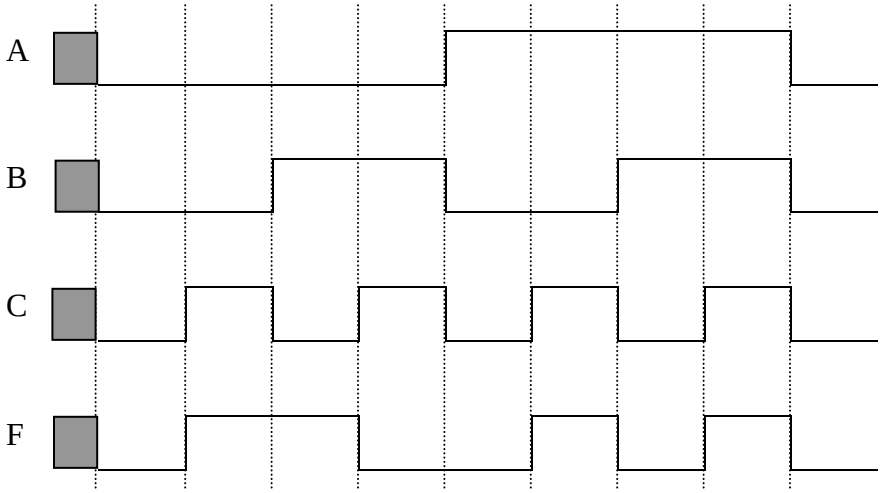
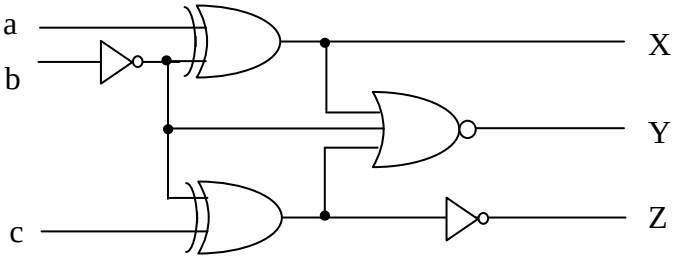


Figure P4.1

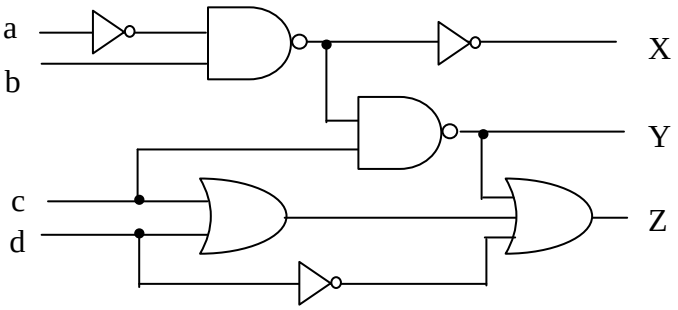
16. Find the simplest sum-of-products expressions for X, Y, and Z in each of the logic circuits in Figure P4.2.

Figure P4.2

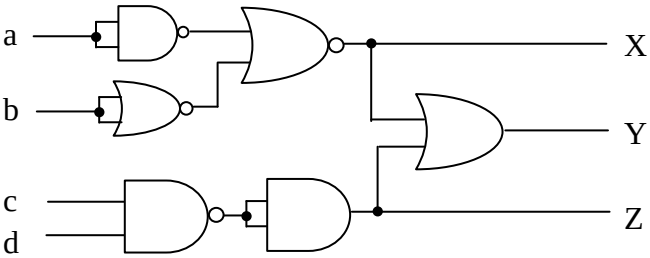
Circuit (a)



Circuit (b)

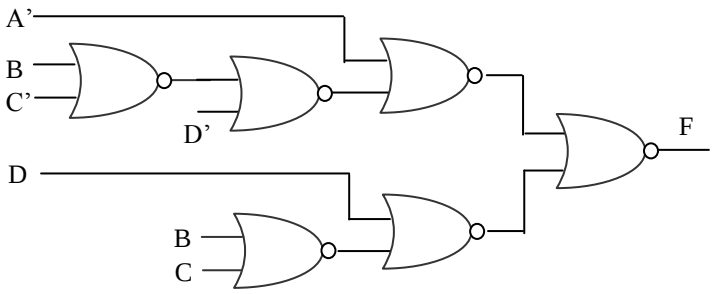


Circuit (c)



17. Find the simplest sum-of-products and product-of-sums expressions for the logic circuits in Figure P4.3 without eliminating the internal and output inversions.

Figure P4.3



18. Minimize the number of internal and output inversions for the circuit in Figure P.4.3 and then determine the simplest sum-of-products expression for F.

19. A switching circuit has four inputs a, b, c, d and an output /V. The input combination abcd is a (6, 3, 1, 1) weighted-code given in the rightmost column of Table 2.3. /V is an active-low output which is asserted if and only if the input combination is a valid (6, 3, 1, 1) weighted code. Construct a truth table for /V and find the simplest sum-of-products and simplest product-of-sums expressions for /V.

20. A switching circuit has four inputs a, b, c, d and an output V. The input combination abcd is the reflected code given in Table 2.4. V is an active-high output which is asserted if and only if the input combination is a valid reflected code. Find the minterm and maxterm list forms for V.

21. A switching circuit has four inputs a, b, c, d and four outputs W, X, Y, Z. The input combination abcd is an excess-3 code and the output combination WXYZ is the reflected code given in Table 2.4. Find the minterm and maxterm list forms for W, X, Y, and Z.