CHAPTER 3

# BOOLEAN ALGEBRA

Boolean algebra is the fundamental mathematics applied to the analysis and synthesis of digital systems. Because of its application to two-value systems, it is also called switching algebra. The development of switching algebra in this chapter will begin with the introduction of three basic logical operations: NOT, AND, and OR.

## 3.1 Basic Logical Operations

NOT is a logical operation to convert a signal from one value to the other value. For a binary digital system, NOT will change a signal from one state to the other state The truth table for NOT is given in Table 3.1, where x is the signal to which the logical operation is applied, and F is the result of the operation. The device used to perform the NOT operation is called an INVERTER. The logic symbol for an inverter is shown in Figure 3.1. The mathematical representation of NOT is denoted by x', which is called the complement or inversion of x, or NOT x. The prime can also be replaced with a bar over x.

$$F = x' = \overline{x}$$

Table 3.1 Truth table for NOT.

| x | F |
|---|---|
| 0 | 1 |
| 1 | 0 |

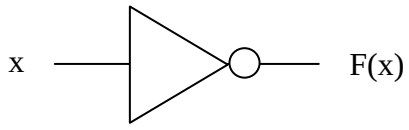



Figure 3.1 Logic symbol for inverter.

Since the value of F depends on that of x, F can also be called a logic function, Boolean function, switching function, or in short a function of x. x is called a Boolean variable, switching variable, or in short, a variable. The function of x can be written as

$$F(x) = x'$$

If the circle at the output of the inverter is missing, as shown in Figure 3.2, the symbol is called a buffer. The output of a buffer is the same as the input.

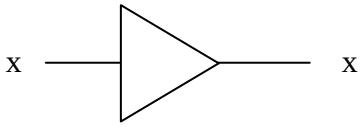



Figure 3.2 Logic symbol for buffer.

The AND operation is described by the truth table in Table 3.2. It defines a relationship between two signals x and y. The result of the operation is F(x, y), which is called "x AND y". F is a function of x and y. The algebraic form is

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

The dot between x and y may be omitted. x and y are the variables of the function F. The device that carries out the AND operation is called an AND gate. Its logic symbol is shown in Figure 3.3. An AND gate has at least two inputs. An AND gate with n inputs is called an n-input AND gate.

Table 3.2 Truth table for AND.

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

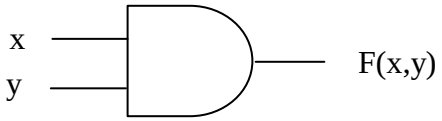



Figure 3.3 Logic symbol for AND gate.

The truth table for the OR operation is given in Table 3.3. The algebraic expression for OR is

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

The operation is carried out by a device called an OR gate as shown in Figure 3.4. An OR gate with n inputs is called an n-input OR gate.

Table 3.3 Truth table for OR.

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

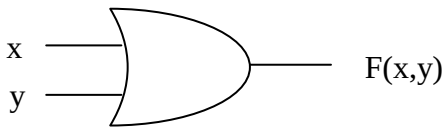



Figure 3.4 Logic symbol for OR gate.

The properties of AND gates and OR gates with more than two inputs are discussed in the next section.

Boolean Expressions and Digital Circuits

Input signals to a digital circuit are represented by Boolean or switching variables such as A, B, C, etc. The output is a function of the inputs. When there is more than one logical operation in a digital circuit, parentheses are used to specify the order of operations in the Boolean expression for the circuit.

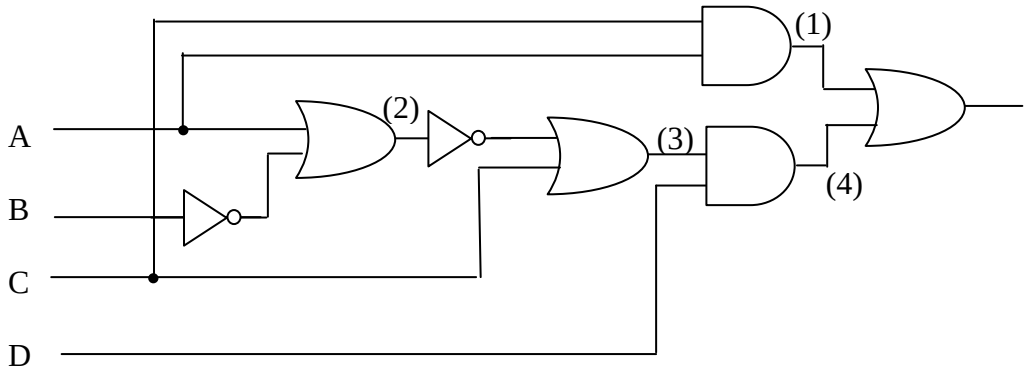



Figure 3.5 Logic circuit.

For the circuit in Figure 3.5, the Boolean expression is written as follows:

| | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | ( | A | • | C | ) | + | ( | D | • | ( | ( | A | + | B ' | ) ' | + | C | ) | ) |
| Parenthesis pair | | 1 | | 1 | | | 4 | | 3 | 2 | | | 2 | | | | 3 | 4 | |

Four pairs of parentheses are used to determine the order of seven logical operations: two NOTs, three ORs, and two ANDs. The operation(s) within a pair of parentheses are performed before those outside this pair of parentheses. Precedence is always applied to the inversion of a variable. If an inversion is performed on the Boolean expression within a pair of parentheses, such as the prime attached to parentheses pair 2, it has the precedence over all operations outside this parenthesis pair. The Boolean function(s) performed within each of the four pairs of parentheses are also shown in Figure 3.5 as (1), (2), (3), and (4)

If the precedence rule that ANDs are executed before ORs in the absence of parentheses, parentheses pairs 1 and 4 can be removed without ambiguity. The expression can be simplified to

$$A \cdot C + D \cdot ( ( A + B' )' + C )$$

## 3.2 Basic Laws

The properties of Boolean algebra are described by the basic laws introduced in this section. Students should try to show the validity of basic laws (1) through (5) using truth tables. This method of proving the equality of two expressions is known as the

perfect induction method. Basic law (6a) will be used as an example to show how to prove the validity of a Boolean/switching identity using the perfect induction method.

(1) Involution law
$(A')' = A$

(2) Idempotency law
(a) $A \cdot A = A$
(b) $A + A = A$

(3) Laws of 0 and 1
(a) $A \cdot 1 = A$
(b) $A + 0 = A$
(a') $A \cdot 0 = 0$
(b') $A + 1 = 1$

(4) Complementary law
(a) $A \cdot A' = 0$
(b) $A + A' = 1$

(5) Commutative law
(a) $A \cdot B = B \cdot A$
(b) $A + B = B + A$

(6) Associative law
(a) $(A \cdot B) \cdot C = A \cdot (B \cdot C)$
(b) $(A + B) + C = A + (B + C)$

Table 3.4 Proof of associative law (6a)

| A B C | A • B | Left-hand-side of (6a) (A • B ) • C | B • C | Left-hand-side of (6b) A • (B • C) |
|---|---|---|---|---|
| 0 0 0 | 0 | 0 | 0 | 0 |
| 0 0 1 | 0 | 0 | 0 | 0 |
| 0 1 0 | 0 | 0 | 0 | 0 |
| 0 1 1 | 0 | 0 | 1 | 0 |
| 1 0 0 | 0 | 0 | 0 | 0 |
| 1 0 1 | 0 | 0 | 0 | 0 |
| 1 1 0 | 1 | 0 | 0 | 0 |
| 1 1 1 | 1 | 1 | 1 | 1 |

Table 3.4 is the proof for the associative law (6a). All the possible combinations of A, B, and C are given in the first column from the left. In the next two columns, the

values of the left-hand-side of (6a) are derived. Those values for the right-hand-side of (6a) are given in the two right columns. For each and every possible combination of A, B, C, the values listed for the left-hand-side and the right-hand-side of (6a) are equal. Thus it can be concluded that (A • B ) • C is equal to A • (B • C).

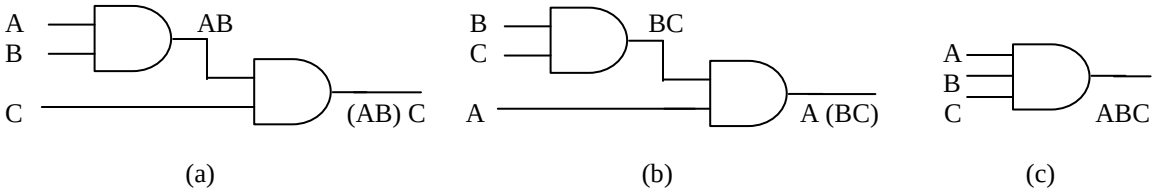



Figure 3.6 (a) Logic circuit for (AB) C. (b) Logic circuit for A (BC). (c) 3-input AND gate.

The left-hand-side and the right-hand-side of (6a) are implemented using 2-input AND gates in Figures 3.6(a) and 3.6(b) respectively. The associative law indicates that the order of the two AND operations is immaterial. The parentheses can be omitted without ambiguity. By applying the commutative property, the expression ABC can also be written in any other permutations such as ACB, BCA, etc. In fact, the expression ABC can be implemented by an AND gate with three inputs, or a 3-input AND gate as shown in Figure 3.6(c). The output of a 3-input AND gate is 1 if and only if all inputs are 1. This can be extended to any number of inputs.

Similarly, the associative law (6b) can be written as A + B + C, A + C + B, B + A + C, etc. A 3-input OR gate can be used to implement the expression A + B + C. The output of an n-input OR gate is 1 when one or more than one input is equal to 1, where n is an integer equal to or greater than 2.

(7) Distributive law
  (a) A (B + C) = A B + A C
  (b) A + B C = (A + B) (A + C)

The validity of the distributive law (7a) is proved in Table 3.5 using the perfect induction method. The value obtained for the left-hand-side (LHS) of the distributive law and that for the right-hand-side (RHS) are equal for each and every one of the eight different input states for A, B, C.

The validity of the distributive law (7b) is proved by a more compact approach. In the compact method, perfect induction is applied to only some of the variables. These variables are expressed as 0 and 1 in a truth table. Algebraic operations such as the basic laws are used for other variables. Such approach will reduce the number of rows in a truth table and is called "compact truth table" method.

In Table 3.6, the perfect induction method is applied to A only. Therefore there are two rows in the table, one row for A = 0 and a second row for A = 1. Algebraic approach is used for B and C. The values of B and C are not explicitly listed in the table.

The variable A on the right-hand-side and the left-hand-side of the distributive law (7b) is then substituted by 0 and 1. When A = 0, both sides are equal to BC. When A = 1, both sides are equal to 1. Thus it can be concluded that the distributive law (7b) is valid.

Table 3.5 Proof of distributive law (7a) by perfect induction.

| A B C | B + C | Left-hand-side of (7a) A (B + C) | A B | A C | Right-hand-side of (7a) AB + AC |
|---|---|---|---|---|---|
| 0 0 0 | 0 | 0 | 0 | 0 | 0 |
| 0 0 1 | 1 | 0 | 0 | 0 | 0 |
| 0 1 0 | 1 | 0 | 0 | 0 | 0 |
| 0 1 1 | 1 | 0 | 0 | 0 | 0 |
| 1 0 0 | 0 | 0 | 0 | 0 | 0 |
| 1 0 1 | 1 | 1 | 0 | 1 | 1 |
| 1 1 0 | 1 | 1 | 1 | 0 | 1 |
| 1 1 1 | 1 | 1 | 1 | 1 | 1 |

Table 3.6 Proof of distributive law (7b) by the compact truth table method.

| A | Left-hand-side of (7b) A + B C | Right-hand-side of (7b) (A + B)( A + C) |
|---|---|---|
| 0 | 0 + B C = B C | (0 + B ) (0 + C) = B C |
| 1 | 1 + B C = 1 | (1 + B ) (1 + C) = 1•1 = 1 |

## 3.3 Sum-of-Products and Product-of-Sums Expressions

When a variable appears unprimed or primed in a switching expression, it is called a literal. An unprimed or a primed variable is also said to be, respectively, in true form or complemented form. If several literals are ANDed together, the result is called a product. Similarly, the ORing of several literals produces a sum. When several products are ORed together, the expression is called a sum-of-products (SOP) expression. In a sum-of-products expression, a product can have only one literal. Two examples for sum-of-products expression are given below. The first expression is a sum of three products. The second expression has four products, one of which is a single literal.

AB’ + BC + A’BD’

B’ + CD + A’C’D’ + AE’

A product-of-sums (POS) expression is the AND of several sums. In a product-of-sums expression, a sum may have just one literal. Two examples for product-of-sums expression are shown below. The first product-of-sums expression has three sums. The second expression also has three sums. But one of them is a single literal.

(A' + C') (A + C + D') (B + D')

C' (B' + D') (A + B + D)

Simplest (Minimal) Sum-of-Products and Product-of-Sums Expressions

Sum-of-products and product-of-sums are two fundamental expressions for Boolean functions. When a literal or a product is deleted from a sum-of-products expression for a switching function, the expression with deleted literal/product is no longer correct for the function. Then the sum-of-products expression is said to be simplest or minimal. In other words, a sum-of-products expression is simplest if and only if no literal or product can be deleted from the expression. Thus a simplest sum-of-products expression for a function consists of a minimum number of product terms and the total number of literals from all the product terms is also minimum. Two examples to illustrate what is a simplest sum-of-products expression are given in Section A.1 of Appendix.

Sum-of-products and product-of-sums expressions can be implemented as standard 2-level AND-OR and 2-level OR-AND circuits respectively as shown in Figure 6.1.

## 3.4 Theorems

The theorems given in this section are useful in simplifying switching expressions or in changing an expression to different forms. In proving a theorem by algebraic approach, only the basic laws and the theorems that have been proved will be used. When a basic law (L) or a theorem (T) is used, the basic law or the theorem number will be quoted inside a pair of square brackets at the end of a line.

(1) Combination theorem

(a) A B + A B' = A

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

Proof: (a) LHS = A B + A B'

= A (B + B') [L7a]

= A • 1 [L4b]

= A = RHS [L3a]

(b) LHS = (A + B) (A + B')

= A + B B' [L7b]

= A + 0 [L2a']
= A = RHS [L3b]

The combination theorem allows two product (sum) terms to be combined to one product (sum) term. The number of literals in the resulting product (sum) is also reduced by one.

(2) Absorption theorem
(a) A + A B = A
(b) A ( A + B) = A

Proof: (a) LHS = A + A B
= A • 1 + A B [L3a]
= A (1 + B) [L7a]
= A • 1 [L3b']
= A = RHS [L3a]

(b) LHS = A (A + B)
= A A + A B [L7a]
= A + A B [L2a]
= A = RHS [T2a]

The following is an alternate proof of (b) using only basic laws.

(b) LHS = A (A + B)
= (A + 0) (A + B) [L3b]
= A + 0 • B [L7b]
= A + 0 [L3a']
= A = RHS [L3b]

When applying the absorption theorem, a product (sum) term will be absorbed by another term and disappears completely. The absorbed term is in fact part of the absorbing term. This is illustrated by Venn diagrams in Section A.2 of Appendix.

❖ Example 3.1

The absorption theorem is used to simplify two expressions in this example.

(a) AC + AB'CDE = (AC) + (AC) (B'DE) = AC

(b) B' ( A + B') (B' + CD') = B' (B' + CD') = B'

Example 3.1 (b) can also be simplified using basic laws.

B' ( A + B') (B' + CD') = (B'+ 0) ( A + B') (B' + CD') =B' + (0)(A)(CD') = B'

(3) Elimination theorem

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

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

Proof: (a) LHS $= A + A' B$

$= (A + A') (A + B)$ [L7b]

$= 1 \bullet (A + B)$ [L4b]

$= A + B = RHS$ [L3a]

(b) LHS $= A (A' + B)$

$= A A' + A B$ [L7a]

$= 0 + A B$ [L4a]

$= A B = RHS$ [L3b]

In the elimination theorem, a literal is eliminated from a product (sum) term that has more literals.

❖ Example 3.2

This example illustrates the simplification of Boolean expression using the elimination theorem.

(a) $AC' + AB'CDE' = A (C' + B'CDE') = A [C' + C(B'DE')]$
$= A (C' + B'DE') = AC' + AB'DE'$

(b) $(B + C') (A + B + C' + D + E)$
$= (B + C') [ (B + C') + (A + D + E)]$
$= B + C'$

❖ Example 3.3

Simplify the sum-of-products expression $(AB' + BCD + A'B'D')$.

By applying the elimination theorem

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

Since $(AB' + BCD + A'B'D')$ can be simplified by removing a literal, it is not a simplest sum-of-products expression.

Some of the basic laws, the combination theorem, absorption theorem, elimination theorem and DeMorgan's theorem are illustrated by Venn diagrams of Set Theory in Section A.2 of Appendix.

(4) Consensus theorem

(a) A B + A' C + B C = A B + A' C

(b) (A + B) (A' + C) (B + C) = (A + B) (A' + C)

Proof: (a) LHS = A B + A' C + B C

= A B + A' C + 1•B•C [L3a]

= A B + A' C + (A + A') B C [L4b]

= A B + A' C + A B C + A' B C [L7a]

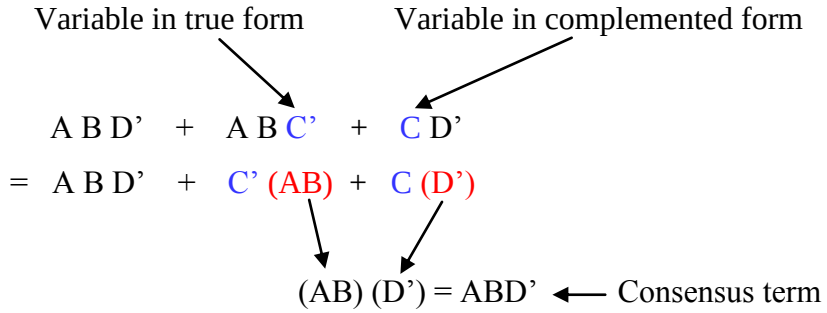
= (A B) + (A B) C + (A' C) + (A' C) B [L6a]

= A B + A' C = RHS [T2a]

The proof of the second form of the consensus theorem is left as an exercise. The consensus theorem will remove a redundant term known as the "consensus term". To find the consensus term, first look for a variable that appears in true form in one product and in complemented form in another product. This variable is A in the first form of the consensus theorem. A and A' are ANDed with B and C respectively. The consensus term is BC. From the proof, it is seen that the consensus term BC is divided into two terms, one "absorbed" by AB and the other by A'C.

❖ Example 3.4

Simplify the sum-of-products expression (A B D' + A B C' + C D') by eliminating a consensus term.



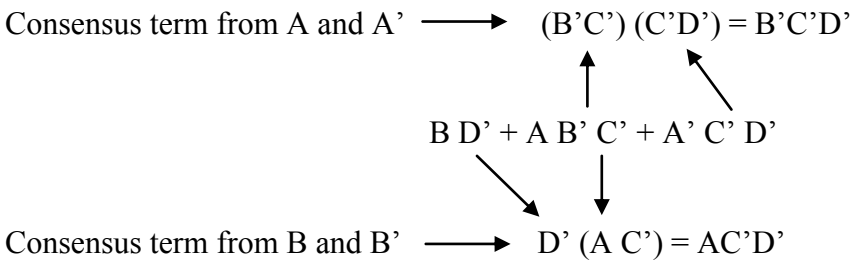

= ABC' + CD'

❖ Example 3.5

B D' + A B' C' + A' C' D'

The above expression is used to show how the consensus theorem can be used to simplify a Boolean expression in a manner different from that in Example 3.6. In

Example 3.6, a consensus term exists in a Boolean expression and is removed. In this example, the expression cannot be simplified before a consensus term is added to the expression.




Two consensus terms, B’C’D’ and AC’D’, can be generated from the variables A and B respectively. None of them exists in the expression. However, AC’D’ is added to the expression so that

BD’ + AB’C’ + A’C’D’ = BD’ + AB’C’ + A’C’D’ + AC’D’

By using the combination theorem,

A’C’D’ + AC’D’ = (A’ + A) C’D’ = C’D’

The expression is simplified to

BD’ + AB’C’ + C’D’

❖ Example 3.6

In this example, the application of consensus theorem is illustrated by adding a consensus term to a Boolean expression in order to eliminate two other terms in the original expression.

A’C + BCD + AC’D + AB’C’

In the above expression, a consensus term ABD can be generated from BCD and AC’D and are added to the expression.

A’C + BCD + AC’D + AB’C’ = A’C + BCD + AC’D + AB’C’ + ABD

BCD is a consensus term from A’C and ABD and can be eliminated from the expression.

A’C + BCD + AC’D + AB’C’ + ABD = A’C + AC’D + AB’C’ + ABD

AC’D is a consensus term derived from AB’C’ and ABD. Thus

A’C + AC’D + AB’C’ + ABD = A’C + AB’C’ + ABD

(5) Interchange Theorem

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

Proof:

| | | |
|---|---|---|
| RHS | = (A + C) (A' + B) | |
| | = A A' + A B + A' C + B C | [L7a] |
| | = 0 + A B + A' C + B C | [L4a] |
| | = A B + A' C + B C | [L3b] |
| | = A B + A' C = LHS | [T4a] |

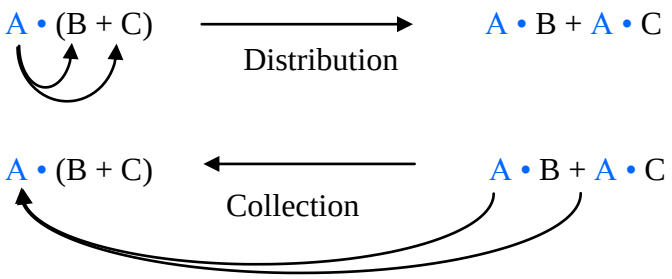
The interchange theorem is not for simplification of Boolean expressions. It is used for the conversion of an expression from a sum-of-products to a product-of-sums, or vice versa. In applying this theorem from the conversion between a SOP expression and a POS expression, a variable should appear in true form in one product and in complemented form in the other product. Or the true form of a variable in one sum and its completed form in another sum. If such a variable does not exist, the theorem is not applicable. The conversion involves a process of interchanging literals. If the variable appears in true and complemented forms is A. The literals associated with A and A’ in the sum-of-products (LHS of Theorem 5) are B and C respectively. To change the expression to a product-of-sums (RHS of Theorem 5), B will associate with A’ and C becomes a partner of A. The process of interchanging associating literals is also applicable to the conversion from product-of-sums expression to sum-of-products expression.

<u>Conversions between Sum-of-Products and Product-of-Sums Expressions</u>

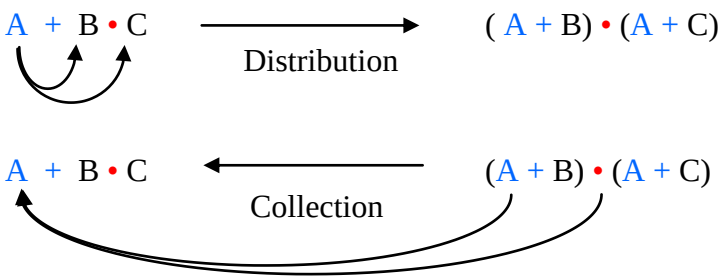
When a product-of-sums expression is converted to a sum-of-product expression, the process is called “multiplying-out”. On the contrary, the change of a sum-of-products expression to a product-of-sums expression is called “factoring”.

The distributive laws are always used in conversions between sum-of-products and product-of-sums. In fact, there are two opposite processes in the distributive laws. These two processes are shown below. Distribution is to distribute a literal to each and every literal in another term. Collection is to collect a common literal from each and every term. One process is the reverse of the other process.

Distributive law (7a)



Distributive law (7b)




When converting a product-of-sums expression to a sum-of-products expression, applying the distributive law (7a) is correct but may not be the best approach. For instance,

(A + B + C)(A + B + D)
= AA + AB + AD + AB + BB + BD + AC + BC + CD

Not only that it is tedious in multiplying out the two sums, the result is not simple and needs to be simplified using the idempotency law and the absorption theorem.

| | |
|---|---|
| AA + AB + AD + AB + BB + BD + AC + BC + CD | |
| = A + AB + AD + B + BD + AC + BC + CD | [L2] |
| = A + B + BD + BC + CD | [T2a] |
| = A + B + CD | [T2a] |

If the second form of the distributive law (7b) is used, it requires only one step in getting the simplest sum-of-products expression.

(A + B + C)(A + B + D)
= [ (A + B) + C] [ (A + B) + D]
= (A + B) + CD

❖ Example 3.7

The collection process of the distributive law can be applied to more than two terms. It is illustrated in this example by converting the following expression to a simplest sum-of-products expression.

(A + B + C)(A + B + D)(A + B + E)(B' + D')

(A + B + C)(A + B + D)(A + B + E)(B' + D')
= ( A + B + CDE ) (B' + D')

= [ B + ( A + CDE )] ( B’ + D’ )
= BD’ + B’ (A + CDE) [T5]
= BD’ + AB’ + B’CDE

❖ Example 3.8

The interchange theorem is employed in the conversion of (BCD’ + B’D + AB) to a product-of-sums expression. B or B’ appears in every product. Therefore the expression can be formulated into two parts, one with B and another with B’.

BCD’ + B’D + AB = B(A + CD’) + B’D

By applying the interchange theorem, the expression becomes

(B + D) (B’ + A + CD’)

By distributing A + B’ to C and D’

A + B’ + CD’ = (A + B’ + C) (A + B’ + D’)
Thus BCD’ + B’D + A’B = (B + D) (A + B’ + C) (A + B’ + D’)

❖ Example 3.9

This example illustrates the conversion of a product-of-sums expression to a sum-of-products expression in its simplest form. For a simplest or minimal sum-of-products expression, the number of product terms should be a minimum. The total number of literals should also be a minimum. There should not be any expression in which the number of product terms or/and the total number of literal are smaller.

(A + B) (A’ + C) (C’ + D)
= (AC + A’B) (C’ + D) [T5]
= ACC’ + ACD + A’BC’ + A’BD [L7a]
= ACD + A’BC’ + A’BD

If the distributive law (7a) is used to multiply out (A + B)(A’ + C), the result is (AC + A’B + BC) instead of (AC + A’B). The consensus theorem has to be used in the elimination of BC. Thus in the interchange theorem, the consensus theorem has already been built in.

❖ Example 3.10

Convert the sum-of-products expression (A’B + CD) to a product-of-sums. The distributive law (7b) is used to distribute A’B to C and D.

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

For each of the two expressions within the parentheses, C as well as D is again distributed to A'B using the same distributive law.

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

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

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

From the four examples given above for conversions between sum-of-products and product-of-sums expression, it is suggested that the collection process of the distributive law be used first and followed by the interchange theorem, and then the distribution process of the distributive law.

Examples 3.7 to 3.10 and the example before Example 3.7 provide a 3-step guideline that can be used for conversions between sum-of-products and product-of-sums expressions. Note that this guideline may not be the best approach in all cases. The procedure is as follows:

(i) Apply the collective process of the distributive law.
(ii) Apply the Interchange theorem.
(iii) Apply the distributive process of the distributive law.

Ignore any step that is not applicable. Basic laws and theorems that can be used for simplification must be used when they are applicable in any step.

(6) DeMorgan's theorem

(a) $(A \cdot B)' = A' + B'$
(b) $(A + B)' = A' \cdot B'$

Table 3.7 Proof of DeMorgan's theorem (6a).

| A B | A B | Left-hand-side of (6a) (A B)' | A' B' | Right-hand-side of (6a) A' + B' | A'•B' |
|---|---|---|---|---|---|
| 0 0 | 0 | 1 | 1 1 | 1 | 1 |
| 0 1 | 0 | 1 | 1 0 | 1 | 0 |
| 1 0 | 0 | 1 | 0 1 | 1 | 0 |
| 1 1 | 1 | 0 | 0 0 | 0 | 0 |

DeMorgan's theorem is used to manipulate the inversion or complement of a Boolean expression. The first form of the theorem (6a) is proved in Table 3.7 using the perfect

induction method. The proof for the second form is left as an exercise. By comparing the two rightmost columns of Table 3.7, it shows that

$$(A\ B)' \neq A'\ B'$$

Similarly, $(A + B)' \neq A' + B'$

DeMorgan's theorem is not limited to just two variables and can be extended to any number of variables. The general forms are as follows.

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

The general DeMorgan's theorem can be proved using the induction method. To prove the first form (a), the starting step is to show that the theorem is true for two variables, which has already been established. The second step is to show that it is also true for three variables.

$$(x_1 \bullet x_2 \bullet x_3)' = ((x_1 \bullet x_2) \bullet x_3)' = (x_1 \bullet x_2)' + x_3'$$
$$= (x_1' + x_2') + x_3' = x_1' + x_2' + x_3'$$

The first form of the 2-variable DeMorgan's theorem is applied to the second equality in the above proof. Now assume the general DeMorgan's theorem is true for (n-1) variables.

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

It is necessary to prove that the theorem is also true for n variables.

$$\begin{aligned} & (x_1 \bullet x_2 \bullet x_3 \bullet \ldots\ldots\ldots \bullet x_{n-1} \bullet x_n)' \\ = \ & ((x_1 \bullet x_2 \bullet x_3 \bullet \ldots\ldots\ldots \bullet x_{n-1}) \bullet x_n)' \\ = \ & (x_1 \bullet x_2 \bullet x_3 \bullet \ldots\ldots\ldots \bullet x_{n-1})' + x_n' \\ = \ & (x_1' + x_2' + x_3' + \ldots\ldots\ldots + x_{n-1}') + x_n' \\ = \ & x_1' + x_2' + x_3' + \ldots\ldots\ldots + x_{n-1}' + x_n' \end{aligned}$$

The 2-variable DeMorgan's theorem is used for the second equality. DeMorgan's theorem for (n-1) variables is applied to the third equality. The second general form of DeMorgan's theorem for n variables can be proved in a similar manner.

❖ Example 3.11

The application of DeMorgan's theorem is illustrated by converting the expression $[A' + B(C + D') + E]'$ to a sum-of-products form. In eliminating a prime outside a pair of parentheses or brackets, each term within the parentheses or brackets is complemented, and the logical operation AND is changed to OR, or vice versa.

$$[A' + B(C + D') + E]'$$

$$= A \bullet [B (C + D')]' \bullet E'$$

$$= A \bullet [B' + (C + D')'] \bullet E'$$

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

$$= AB'E' + AC'DE'$$

## 3.5 Minimization of Literals

Under certain circumstances, the number of literals in a Boolean expression needs to be minimized. The expression with a minimum number of literals may not necessarily be a sum-of products or product-of-sums expression. Inversions can be applied only to variables if an expression is defined as one with a minimum number of literals. The collection process of the distributive law is useful in reducing the number of literals. The process may also be applied to part of an expression and the common factor is not limited to just a single literal.

❖ Example 3.12

$$F(A,B,C,D) = BD + CD + A'BC + ABC'$$

Given above is the simplest sum-of-products form for a 4-variable function F(A,B,C,D). By factoring D from the first two products and B from the last two products, the expression becomes

$$D(B + C) + B(A'C + AC')$$

The number of literal for the given expression can also be minimized by factoring different literals, which results in the following two expressions.

$$CD + B(D + A'C + AC')$$

and $$B(D + AC') + C(D + A'B)$$

Each of the three expressions has eight literals.

❖ Example 3.13

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

D is the only literal that appears more than once in the above expression. By collecting D from the last two sums, F becomes

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

The number of literals is reduced from seven to six. The following is another form of F using the interchange theorem.

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

The expression still consists of seven literals. The interchange theorem only changes the form of an expression. It does not reduce the number of literals.

## 3.6 Duality

For all the laws and theorems of Boolean algebra introduced in this chapter, it is seen that each of them always occur in pairs, except the involution law and the interchange theorem. The relationship between the two different forms of a law or a theorem is called duality. One is said to be the dual of the other. The dual of a Boolean expression, identity, or equation can be obtained by the following transformations.

| | | |
|---|---|---|
| AND | ⟶ | OR |
| OR | ⟶ | AND |
| 0 | ⟶ | 1 |
| 1 | ⟶ | 0 |

All the variables and their complements are left intact and the order of logical operations is not altered by the transformations. To ensure that the order is not changed, a Boolean expression can be fully parenthesized. In other words, the convention that AND operations are performed before OR operations in the absence of parentheses cannot be applied. The order of operations is specified by parentheses and brackets. Mathematically, the dual of a Boolean expression, $F^D$, can be specified as follows:

$$F^D(x_{n-1}, x_{n-2}, \ldots\ldots, x_2, x_1, x_0, 0, 1, \bullet, +)$$
$$= F(x_{n-1}, x_{n-2}, \ldots\ldots, x_2, x_1, x_0, 1, 0, +, \bullet)$$

Two basic laws are used to illustrate the principle of duality.

Laws of 0 and 1: (L3a') $\quad A \bullet 0 = 0$

Laws of 0 and 1: (L3b') $\quad A + 1 = 1$

Distributive law: (7a) $A \bullet ( B + C ) = ( A \bullet B ) + ( A \bullet C )$

Distributive law: (7b) $A + ( B \bullet C ) = ( A + B ) \bullet ( A + C )$

❖ Example 3.14

The dual of a Boolean expression F is obtained by fully parenthesizing the expression before transformation and by removing unnecessary parentheses and brackets using the AND/OR convention after transformation.

$$F = [ A' + B ( C + D' ) + E \bullet 0 ]' \bullet B'$$

F fully parenthesized: $F = \{ A' + [ B \bullet ( C + D' ) ] + ( E \bullet 0 ) \}' \bullet B'$

Transformation: $F^D = \{ A' \bullet [ B + ( C \bullet D' ) ] \bullet ( E + 1 ) \}' + B'$

$$= [ A' ( B + C D' ) ( E + 1 ) ]' + B'$$

Positive Logic and negative Logic

The logic values of a signal in a binary digital system are defined by two different voltage levels called HIGH (H) and LOW (L). As described in Section 1.1, the two levels denote two different states. The two states can also be represented by 0 and 1. Because 0 and 1 may not have any numerical significance, it is not required that 0 is assigned to L and 1 to H. When 0 is assigned to L and 1 to H, it is referred to as "positive logic". On the other hand, “negative logic” will assign 0 to H and 1 to L.

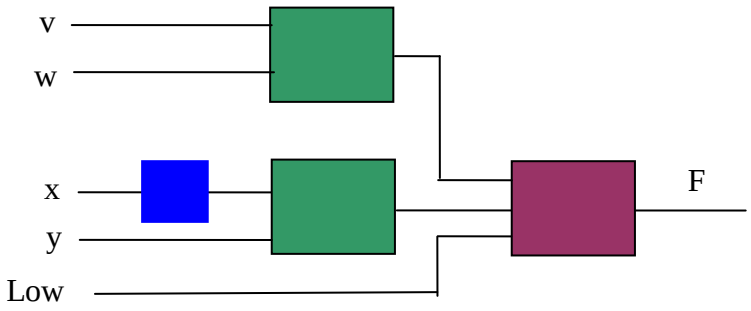



Figure 3.7 A digital circuit with three different components and un specified logic.

Figure 3.7 is a circuit built from three different types of components specified by colors. Table 3.8 is the truth tables for these three different components. Regardless of the selected logic, a component that satisfies the truth table in Table 3.8(a) is always an inverter. If positive logic is adopted, the truth table in Table 3.8(b) is for AND gates. It becomes a truth table for OR gates if the adopted logic is negative. Similarly, the truth table in Table 3.8(c) is for OR gates in positive logic and for AND gates in negative logic. The change of one logic to the other logic is equivalent to converting 0 to 1, 1 to 0, AND to OR, and OR to AND. Thus, when a circuit output or Boolean expression F is given for one type of logic, the output or the Boolean function F for the other type of logic is the dual of F.

Table 3.8 Truth tables for three types of gates.

(a)

| Input | Output |
|---|---|
| L | H |
| H | L |

(b)

| Inputs | Output |
|---|---|
| L L | L |
| L H | L |
| H L | L |
| H H | H |

(c)

| Inputs | Output |
|---|---|
| L L L | L |
| L L H | H |
| L H L | H |
| L H H | H |
| H L L | H |
| H L H | H |
| H H L | H |
| H H H | H |

If positive logic is applied to the circuit in Figure 3.7 and as shown in Figure 3.8(a), the output Z is

$$F = vw + x'y + 0$$

A "LOW" input to the circuit is purposely included in the circuit for illustrating the relationship between positive logic and negative logic. If negative logic is adopted for the circuit, the OR operation is replaced with an AND operation and the two OR operations will replace the two AND operations. The circuit for negative logic in Figure 3.7(b) is shown in Figure 3.8(b).

$$F = (v + w) \bullet (x' + y) \bullet 1$$

The two different expressions of Z show that one is the dual of the other.

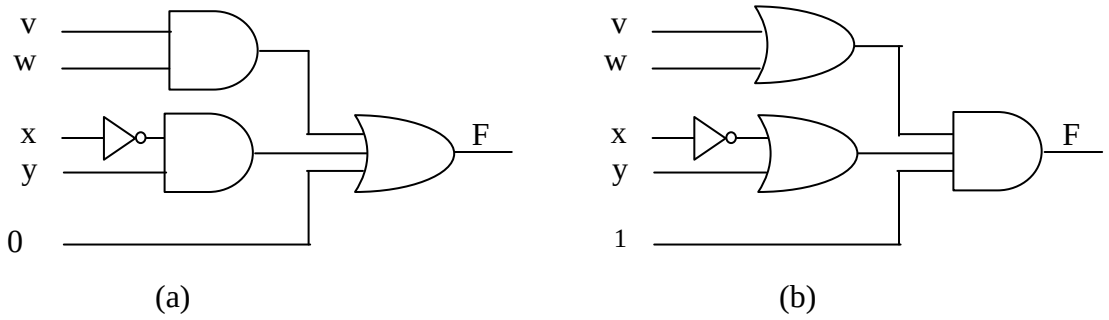



Figure 3.8 A digital circuit with different logic. (a) Positive logic. (b) Negative logic.

PROBLEMS

1. Find the truth table for each of the following expressions.

   (a) f(A,B,C,D) = AC’ + CD’ + B(AC + BD)
   (b) f(A,B,C,D) = A(B + C’) + A’B’C’D’ + A’(BD + C)
   (c) f(A,B,C,D) = (AB + C’D’)(C + A) + A’CD’

2. Simplify each of the following expressions to a single literal or constant value using Boolean algebra.

   (a) ABC + B’CD + C
   (b) B’ + B’(A’ + CD + CE’)
   (c) (A’ + A + BCD)(A + A’ + B’C’D’)
   (d) ABC + AB’ + AC’

3. Simplify each of the following expressions to a sum-of-products or product-of-sums expression using Boolean algebra.

   (a) ABC’+ C
   (b) AB + B’CD + BCD
   (c) A(B’ + C’) + A(B + C) + B + C
   (d) A’B’ + AB’ + BC’D’

4. Minimize the number of literals in each of the following expressions using Boolean algebra.

   (a) x’y’ + xy + xy’
   (b) x’y’ + (xy)’ + xy’
   (c) x’y’ + (x’y’)’ + xy’
   (d) x’y’ + xy + x’y

5. Minimize the number of literals in each of the following expressions using Boolean algebra.

   (a) f(a,b,c,d) = [(a + bc’)(a + c’d’) + ab]’
   (b) f(a,b,c,d) = [ a + b’ + (ab’ + cd’)(c + ab)]’
   (c) f(a,b,c,d) = (abc’ + d)’ (a’d + b)’
   (d) f(a,b,c,d) = [(a’b’ + ab)’ + (a + b’ + c)’]’

6. Minimize the number of literals in each of the following expressions using Boolean algebra.

   (a) (w’ + y’)(y’ + xz)(y’ + x)
   (b) a’bc + ab + bcd’
   (c) (ad’ +c)(a +d)(d’ + c) + bc

7. Simplify each of the following switching expressions to a simplest sum-of-products expression. Use consensus theorem if necessary.

   (a) ab + ab'd + a'cd
   (b) ab' + a'b'd' + bc'd' + abcd'
   (c) a'bd + a'bc + bcd'
   (d) abc + a'b'c' + a'c'd + b'c

8. Prove the validity of the following equation using
   (a) Perfect induction method.
   (b) Compact truth table method.
   (c) Consensus theorem.

   A'B + AC + B'C' = A'C' + BC + AB'

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

   (a) (a + b) (a + c) (a' + c' + bd')
   (b) d (c' + abd) (ac' + bc)
   (c) (abc' + d)(bd + a')(a' + b + c)

10. Simplify each of the following expressions to a product-of-sums expression.

    (a) (a + c) (a + c' + b) (a + c' + d')
    (b) d (c' + abd) (ac' + b)
    (c) (a + bc'd)(ad + d')( b + c + d')

11. Find the dual for each of the following expressions. (Do not simplify the expression and the result).

    (a) a b c + [ e (d' + c') + a']
    (b) a b d + c' (a' b' + 0 + d')
    (c) (w' x + 1) (y z + u) + u' w' (y + x)

12. Each of the following expressions is the output of a circuit using negative logic. What should the output be if positive logic is adopted?

    (a) abc' + bcd +a'c'd'
    (b) a (b' + d) + a'c' (b + e)
    (c) d' [ (a' + c)(b' + c') ] + a d

13. Convert each of the following expressions to a simplest sum-of-products expression.

    (a) (a + b + c)(a' + b' + c)(c + d')
    (b) (a + b + c' ) (a + b' + d) (b' + d' )
    (c) (w + x + y' ) ( w' + x' + y' ) (y' + z) (u' + x + y' )

14. Convert each of the following expressions to a simplest product-of-sums expression.

(a) a' c' + a b' d
(b) a' b' c' + b' d + b d'
(c) ab’c’ + a’bc’ + b’cd’ + bd