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.

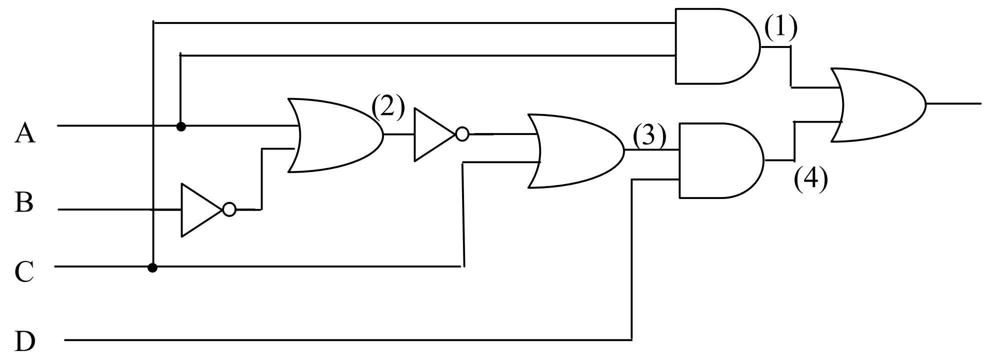

Figure 3.5 Logic circuit.

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•C + D•((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 • A = A
  (b) A + A = A

(3) Laws of 0 and 1
  (a) A • 1 = A
  (b) A + 0 = A
  (a') A • 0 = 0
  (b') A + 1 = 1

(4) Complementary law
  (a) A • A' = 0
  (b) A + A' = 1

(5) Commutative law
  (a) A • B = B • A
  (b) A + B = B + A

(6) Associative law
  (a) (A • B) • C = A • (B • 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).

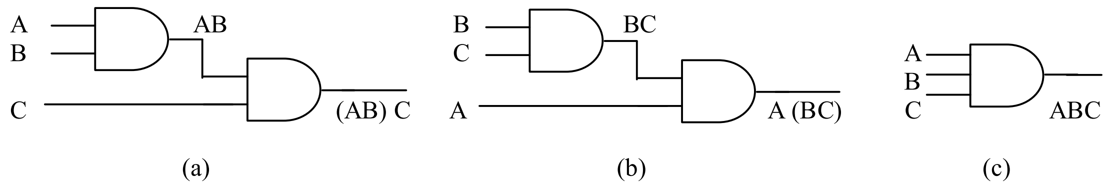

Figure 3.6 (a) Logic circuit for (AB) C. (b) Logic circuit for A (BC). (c) 3-input AND gate.

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'