

Chapter 4 Boolean Logic

A digital system, like any other system consists of a group of interrelated parts that form a complex whole. But in the field of engineering, *system* also implies a unique transformation or operation that transforms a given input condition to a specific output. Digital systems have inputs and outputs (that are typically reduced to digital signals as discussed in Chapter 1), and the thing that ties them together and provides the unique transformation is the digital circuitry.

In order to design digital circuits in a concise, methodical way, it is necessary to understand some underlying theory. It is important to understand that this theory is independent of circuit technology or voltage levels. In fact, there is no requirement that signals have voltage levels at all, only that they are constrained, at any point in time, to one of two possible values. The physical details of the signals are deliberately omitted from this chapter in an effort to focus on the *semantics* or meaning of the signals.

Rather than cover this theory in isolation (as most texts do), this chapter will lead us through several design problems and introduce the theory as it is needed and applicable.

4.1 Boolean Algebra

Let's begin with a practical design problem. An appliance manufacturer needs a digital system to control the hot and cold water valves on an automatic washing machine. The user can select hot, warm or cold wash temperature, but the rinse temperature is always cold. Finally, the legal department requires that the hot water valve cannot be on when the lid is open unless the user has selected hot. (If someone selects warm water, which normally opens both valves, and burns themselves because the cold water valve has failed, there might be a liability issue.)

The next step is to determine the inputs and outputs of the system. Only then can we clearly understand what the system is required to do. The buttons that select wash temperature provide three input signals for hot, warm and cold. The main controller provides two more inputs that indicate whether wash or rinse water is needed. The lid also sends a signal to indicate that the lid door is open. There are two outputs; one turns on the hot water valve and the other turns on the cold.

So where do we go from here? We can write a verbal description for each output, for example:

Turn on the cold water valve when the controller is requesting rinse water or when the controller is requesting wash water and the user has not selected hot.

Turn on the hot water valve when the controller is requesting wash water and the user has selected hot or warm water, but not if the user has selected warm water and the lid is open.

The verbal descriptions help humans understand the problem better, but they don't really get us closer to a design. Thankfully, the theoretical work of George Boole gives us an alternative. He realized that the principles of algebra also apply to logical propositions (statements that are classified as either true or false). In the algebra he developed (which is called *Boolean Algebra*), variables and expressions represent logical propositions and are also either true or false.

The next step in the design, then, is to represent all inputs and outputs of the system as Boolean variables. Once that is done, the only real difficulty that remains is finding a mathematical transformation that maps the possible inputs to the desired outputs. For this design, one choice of Boolean input and output variables is shown below.

| <u>Input Variable</u> | <u>Description</u> |
|-----------------------|---|
| <i>HOT</i> | True if user has requested hot wash water, false otherwise. |
| <i>WARM</i> | True if user has requested warm wash water, false otherwise. |
| <i>COLD</i> | True if user has requested cold wash water, false otherwise. |
| <i>WASH</i> | True if the controller is requesting wash water, false otherwise |
| <i>RINSE</i> | True if the controller is requesting rinse water, false otherwise |
| <i>LID</i> | True if the lid is open, false otherwise |

| <u>Output Variable</u> | <u>Description</u> |
|------------------------|--|
| <i>HOT_VALVE</i> | True if the hot water valve is open, false otherwise. |
| <i>COLD_VALVE</i> | True if the cold water valve is open, false otherwise. |

There are three fundamental logic operations that can be applied to Boolean variables: **and**, **or** and **not**. The meaning of these is defined by the *truth tables* in Figure 4-1. (A truth table gives the result of an operation for every possible combination of operands.) Note that the behavior of **or** may not be what you expect when both operands are true.

| A | B | A and B | A | B | A or B | A | not A |
|-------|-------|---------|-------|-------|--------|-------|-------|
| false | false | false | false | false | false | false | true |
| false | true | false | false | true | true | true | false |
| true | false | false | true | false | true | | |
| true | true | true | true | true | true | | |

Figure 4-1 Truth Tables for **and**, **or** and **not**.

And, **or** and **not** are sufficient to describe any logical relationship between Boolean variables, no matter how complex. For example, a logical expression for the cold water valve is:

$$COLD_VALVE = RINSE \text{ or } (WASH \text{ and not } HOT) \quad (4.1)$$

The expression for the hot water valve is a little more complex:

$$HOT_VALVE = WASH \text{ and } (HOT \text{ or } WARM) \text{ and not } (WARM \text{ and } LID) \quad (4.2)$$

The student is encouraged to look back at the verbal descriptions and verify that these logical expressions are derived directly from those descriptions.

We often need to manipulate logical expressions. For example, the expression for HOT_VALVE is more complicated than necessary, and we may want to manipulate the expression to simplify it. The notation above is easy to understand, but it does not easily lend itself to algebraic manipulation. Boolean Algebra solves this problem by adopting mathematical notation for constants, variables and logic expressions. This notation is described below:

In Boolean algebra, a one (1) is equivalent to true, and a zero (0) is equivalent to false.

The **and** operator is represented by the symbol “.” and follows all the algebraic rules for multiplication. The result of the **and** operation is called the *product*. The product of the Boolean variables *A* and *B* is written $A \cdot B$ (or simply AB if the meaning is clear).

The **or** operator is represented by the symbol “+” and follows all the algebraic rules for addition. The result of the **or** operation is called the *sum*. The sum of Boolean variables *A* and *B* is written $A + B$. Note that **and** has precedence over **or**, so in the expression $A + B \cdot C$, $B \cdot C$ is evaluated first.

The result of the **not** operator is called the *complement*. The **not** operator may be represented by several different symbols. This text uses overbar ($\bar{\quad}$) notation, which means that the complement of the Boolean variable *A* is written \bar{A} . Other texts may write it $\neg A$, $\sim A$ or A' . The overbar notation may also be used with expressions, such as $\overline{A + B}$. The expression under the bar is always evaluated before the **not** operator is applied.

Using this notation, the truth tables in Figure 4-1 can be rewritten as shown in Figure 4-2.

| | | | | | | | |
|---|---|-------|---|---|-------|---|-----------|
| A | B | A · B | A | B | A + B | A | \bar{A} |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |
| 0 | 1 | 0 | 0 | 1 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Figure 4-2 Truth Tables for **and**, **or** and **not** in Boolean Algebra.

The logical expressions for the cold and hot water valves can now be rewritten:

$$COLD_VALVE = RINSE + WASH \cdot \overline{HOT} \quad (4.3)$$

$$HOT_VALVE = WASH \cdot (HOT + WARM) \cdot \overline{WARM \cdot LID} \quad (4.4)$$

At first, this notation may not be as easy to understand, but with practice it will become as natural as the notation in Equations 4.1 and 4.2.

Example 4.1. A safe manufacturer wants to design a safe that has two key switches, one for the manager and one for the assistant manager. The safe will also have a clock so that the door can be opened with either key during business hours, but both keys are required to open the safe after hours. You are to design a digital system to drive a solenoid that unlocks the door. (a) list the input and output signals that the system requires, (b) write down a verbal description for what the system should do, (c) define the Boolean variables you would use for input and output, and (d) write an expression for each output using Boolean Algebra.

Solution:

(a) List the input and output signals that the system requires.

Each key switch sends a signal that tells whether or not the key has been turned. The clock generates a signal that indicates whether or not the current time is during business hours. The output of the system is a signal to unlock the door.

(b) write down a verbal description for what the system should do.

Unlock the door when either key is turned during business hours or when both keys are turned after business hours.

(c) Define the Boolean variables you would use for input and output.

| <u>Input Variable</u> | <u>Description</u> |
|-----------------------|--|
| KEY1 | One (1) if the manager's key has been turned, zero (0) otherwise. |
| KEY2 | One (1) if the assistant manager's key has been turned, zero (0) otherwise. |
| BUSINESS_HOURS | One (1) if the clock indicates that it is currently during business hours, zero (0) otherwise. |

| <u>Output Variable</u> | <u>Description</u> |
|------------------------|--|
| UNLOCK | One (1) if the door-unlock solenoid is on, zero (0) otherwise. |

(d) Write an expression for each output using Boolean Algebra.

$$UNLOCK = (KEY1 + KEY2) \cdot BUSINESS_HOURS + KEY1 \cdot KEY2 \cdot \overline{BUSINESS_HOURS}$$

We will see how to design digital circuitry to implement the logical equations in Chapter 5.

One last operator that is sometimes used in Boolean Algebra is **exclusive-or**. This operator is much like **or**, except that if both operands are true, the result is false (See Figure 4-3). The **exclusive-or** operator is represented by the symbol " \oplus " and also follows the algebraic rules for addition. The exclusive or of Boolean variables A and B is written $A \oplus B$. Note that this operator it is not actually necessary because:

$$A \oplus B = A\bar{B} + \bar{A}B \quad (4.4)$$

The **exclusive-or** operator had more utility years ago when digital circuits largely consisted of small-scale integration logic chips, but today its value is debatable.

| A | B | $A \oplus B$ |
|-----|-----|--------------|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 0 |

Figure 4-3 Truth Table for the **exclusive-or** operator.

4.2 Truth Tables, Minterms and Maxterms

Sometimes, the logical expression that relates the inputs of a system to the outputs cannot be easily derived from the problem statement. For example, suppose we are to design a prime number detecting circuit that takes a 3-bit binary number as an input and tells whether or not the number is prime. Even if we write a verbal expression and define the Boolean variables, the logical expression is still illusive:

Verbal expression: *If the input number is 2, 3, 5 or 7, output a 1, otherwise output a 0.*

Boolean Variables:

| <u>Input Variable</u> | <u>Description</u> |
|------------------------|---|
| B_0 | Least significant (1's) bit of the input binary number. |
| B_1 | 2's bit of the input binary number. |
| B_2 | 4's bit of the input binary number. |
| <u>Output Variable</u> | <u>Description</u> |
| $PRIME$ | 1 if the number is prime, 0 otherwise. |

Logical Expression: ?

In cases like these, we can use a truth table to derive a logical expression as long as we know what the output should be for any given input. A truth table lists every possible input combination and the output(s) for each. To create a truth table, make columns for each input and each output of the digital circuit. Put the inputs on the left and the outputs on the right, separated by a vertical line. The ordering is arbitrary, but once it is chosen, you have to stick with it. Next, treat the inputs like bits in a binary number and write down every possible input combination in numerical order. For the prime number detector, above, this step is illustrated in Figure 4-4.

| B_2 | B_1 | B_0 | $PRIME$ |
|-------|-------|-------|---------|
| 0 | 0 | 0 | |
| 0 | 0 | 1 | |
| 0 | 1 | 0 | |
| 0 | 1 | 1 | |
| 1 | 0 | 0 | |
| 1 | 0 | 1 | |
| 1 | 1 | 0 | |
| 1 | 1 | 1 | |

Figure 4-4 Truth Table Inputs in Numerical Order.

Now consider each row, one at a time and answer the question: what *should* the output(s) be? For the first line of the prime number detector, $B_2B_1B_0 = 000$ which is zero in binary. Zero is not prime, so the output should be 0. If this process is repeated for each row, the truth table in Figure 4-5 emerges:

| B_2 | B_1 | B_0 | $PRIME$ |
|-------|-------|-------|---------|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 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 4-5 Truth Table for the Prime Number Detector.

Could we have listed the inputs B_2 , B_1 and B_0 in a different order? Absolutely. The truth table would be different (and perhaps more confusing), but it would still be valid.

But the question still remains: How is a truth table going to help us derive a logical expression? In order to answer that, we must understand minterms and maxterms.

Minterms

A minterm is a logical expression that is true (1) for one and only one row of the truth table. For any row, the associated minterm is the product of every input that is 1 in the row and the complement of every input that is 0. To illustrate, Figure 4-5 shows a truth table for all the minterms with three inputs.

| A | B | C | $\bar{A}\bar{B}\bar{C}$ (m_0) | $\bar{A}\bar{B}C$ (m_1) | $\bar{A}B\bar{C}$ (m_2) | $\bar{A}BC$ (m_3) | $A\bar{B}\bar{C}$ (m_4) | $A\bar{B}C$ (m_5) | $AB\bar{C}$ (m_6) | ABC (m_7) |
|-----|-----|-----|--------------------------------------|--------------------------------|--------------------------------|--------------------------|--------------------------------|--------------------------|--------------------------|--------------------|
| 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 |
| 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 |
| 1 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 |

Figure 4-5 Truth Table of Minterms.

Each minterm is also denoted m_i , where i is the row of the truth table (starting with 0) where the minterm is true (1). The number of minterms depends on the number of inputs. In general, if a truth table has n inputs, there will be 2^n minterms denoted m_0 through m_{2^n-1} .

The key observation is that a logical expression can be formed for *any* output in a truth table simply by taking the sum of the minterm(s) corresponding to rows in the truth table where the output is 1.

Figure 4-6 shows the truth table for the prime number detector, except that it is annotated with the minterms for the rows where *PRIME* is 1. These are the minterms that will contribute the logical expression for *PRIME*.

| B_2 | B_1 | B_0 | <i>PRIME</i> | Minterms |
|-------|-------|-------|--------------|-----------------------------------|
| 0 | 0 | 0 | 0 | |
| 0 | 0 | 1 | 0 | |
| 0 | 1 | 0 | 1 | $\overline{B_2}B_1\overline{B_0}$ |
| 0 | 1 | 1 | 1 | $\overline{B_2}B_1B_0$ |
| 1 | 0 | 0 | 0 | |
| 1 | 0 | 1 | 1 | $B_2\overline{B_1}B_0$ |
| 1 | 1 | 0 | 0 | |
| 1 | 1 | 1 | 1 | $B_2B_1B_0$ |

Figure 4-6 Truth Table for the Prime Number Example, Annotated with Minterms.

All that remains to derive a logical expression is to sum these minterms:

$$PRIME = \overline{B_2}B_1\overline{B_0} + \overline{B_2}B_1B_0 + B_2\overline{B_1}B_0 + B_2B_1B_0 \quad (4.6)$$

This expression can also be written:

$$PRIME = m_2 + m_3 + m_5 + m_7 \quad (4.7)$$

Some authors use a shorthand notation for the sum of minterms shown below:

$$PRIME = \sum m(2,3,5,7) \quad (4.8)$$

Maxterms

A maxterm is a logical expression that is 0 for one and only one row of the truth table. For any row, the associated maxterm is the sum of every input that is 0 in the row and the complement of every input that is 1 in the row (See Figure 4-7).

| A | B | C | $A+B+C$ (M_0) | $A+B+\overline{C}$ (M_1) | $A+\overline{B}+C$ (M_2) | $A+\overline{B}+\overline{C}$ (M_3) | $\overline{A}+B+C$ (M_4) | $\overline{A}+B+\overline{C}$ (M_5) | $\overline{A}+\overline{B}+C$ (M_6) | $\overline{A}+\overline{B}+\overline{C}$ (M_7) |
|-----|-----|-----|----------------------|---------------------------------|---------------------------------|--|---------------------------------|--|--|---|
| 0 | 0 | 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 0 | 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 |
| 0 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 | 1 | 1 | 1 | 0 | 1 | 1 | 1 |
| 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 0 |

Figure 4-7 Truth Table of Maxterms.

Just as each minterm is denoted m_i , each maxterm is denoted M_i , where M_i is zero on row i of the truth table. Minterms and maxterms are logical complements of each other, that is:

$$M_i = \overline{m_i} \text{ and } m_i = \overline{M_i} \quad (4.9)$$

Like minterms, maxterms can also be used to form a logical expression for any output in a truth table. The expression can be formed by taking the product of the maxterm(s) corresponding to those rows in the truth table where the output is 0. If, for example, this approach is applied to the prime number detector, we get *another* logical expression for *PRIME*:

$$PRIME = (B_2 + B_1 + B_0)(B_2 + B_1 + \overline{B_0})(\overline{B_2} + B_1 + B_0)(\overline{B_2} + \overline{B_1} + B_0) \quad (4.10)$$

Equation 4.9 is logically equivalent to Equation 4.6. Both generate exactly the same result for every possible combination of inputs. This expression can also be written:

$$PRIME = M_0M_1M_4M_6 \quad (4.11)$$

Some authors prefer a shorthand notation for the product of maxterms shown below:

$$PRIME = \prod M(0,1,4,6) \quad (4.12)$$

4.3 Properties and Theorems

Whether the logic expression can be derived directly from the problem statement or it is obtained using a truth table, chances are, it will not be in simplest form. It may surprise you to learn that all but one of the logic expressions we derived in Sections 2.1 and 2.2 can be simplified. Complicated expressions lead to complicated (and more expensive) digital circuits, so for a design to be optimal, it is usually necessary to simplify.

To simplify an expression in any algebraic system, it is necessary to have a clear understanding of the properties and the applicable theorems for that system. In Boolean Algebra, most of these come in pairs. The *dual* of a property or theorem can be obtained by exchanging the operators \cdot and $+$, and exchanging the constants 0 and 1. Figure 4-8 shows the basic properties for Boolean algebra.

| | Property (a) | Dual (b) | |
|------------------------|---------------------------------|-----------------------------|--------|
| Properties of 0 and 1: | $X + 0 = X$ | $X \cdot 1 = X$ | (4.13) |
| | $X + 1 = 1$ | $X \cdot 0 = 0$ | (4.14) |
| Idempotent: | $X + X = X$ | $X \cdot X = X$ | (4.15) |
| Complementarity: | $X + \overline{X} = 1$ | $X \cdot \overline{X} = 0$ | (4.16) |
| Involution: | $\overline{(\overline{X})} = X$ | | (4.17) |
| Reflexive | $X = X$ | | (4.18) |
| Commutative: | $X + Y = Y + X$ | $XY = YX$ | (4.19) |
| Associative: | $(X + Y) + Z = X + (Y + Z)$ | $(XY)Z = X(YZ)$ | (4.20) |
| Distributive: | $X \cdot (Y + Z) = XY + XZ$ | $X + (YZ) = (X + Y)(X + Z)$ | (4.21) |

Figure 4-8 Properties of Boolean Algebra.

These properties follow from the definitions of **and**, **or** and **not** in Figure 4-2, and can easily be proven by exhaustion. Simply try every possible variable combination and verify that both sides of the equation yield the same value.

Example 4.2. Prove property 4.16a from Figure 4-8.

Solution: This property can be proved by exhaustion. Since there is only variable, there are $2^1 = 2$ cases. For each of these cases, we must verify that the expression $X + \bar{X} = 1$. These 2 cases and the proofs for each case are enumerated below:

| X | Proof | |
|---|---------------------|---|
| 0 | $0+1 = 1. 1+0 = 1.$ | ✓ |
| 1 | $1+0 = 1. 0+1 = 1.$ | ✓ |

Example 4.3. Prove property 4.21b from Figure 4-8.

Solution: This property can be proved by exhaustion. Since there are three variables, there are $2^3 = 8$ cases. For each of these cases, we must verify that the expression $X+(Y \cdot Z) = (X+Y)(X+Z)$. These 8 cases and the proofs for each case are enumerated below:

| X | Y | Z | Proof | |
|---|---|---|--|---|
| 0 | 0 | 0 | $0+(0 \cdot 0) = 0+0 = 0. (0+0)(0+0) = 0 \cdot 0 = 0.$ | ✓ |
| 0 | 0 | 1 | $0+(0 \cdot 1) = 0+0 = 0. (0+0)(0+1) = 0 \cdot 1 = 0.$ | ✓ |
| 0 | 1 | 0 | $0+(1 \cdot 0) = 0+0 = 0. (0+1)(0+0) = 1 \cdot 0 = 0.$ | ✓ |
| 0 | 1 | 1 | $0+(1 \cdot 1) = 0+1 = 1. (0+1)(0+1) = 1 \cdot 1 = 1.$ | ✓ |
| 1 | 0 | 0 | $1+(0 \cdot 0) = 1+0 = 1. (1+0)(1+0) = 1 \cdot 1 = 1.$ | ✓ |
| 1 | 0 | 1 | $1+(0 \cdot 1) = 1+0 = 1. (1+0)(1+1) = 1 \cdot 1 = 1.$ | ✓ |
| 1 | 1 | 0 | $1+(1 \cdot 0) = 1+0 = 1. (1+1)(1+0) = 1 \cdot 1 = 1.$ | ✓ |
| 1 | 1 | 1 | $1+(1 \cdot 1) = 1+1 = 1. (1+1)(1+1) = 1 \cdot 1 = 1.$ | ✓ |

There are also three theorems that are useful for simplifying logic expressions. These theorems are shown in Figure 4-9.

| | Theorem (a) | Dual (b) | |
|------------------|-------------------------------------|-------------------------------------|--------|
| Absorption: | $X + XY = X$ | $X(X + Y) = X$ | (4.22) |
| Logic Adjacency: | $XY + X\bar{Y} = X$ | $(X + Y)(X + \bar{Y}) = X$ | (4.23) |
| De Morgan's: | $\overline{X + Y} = \bar{X}\bar{Y}$ | $\overline{XY} = \bar{X} + \bar{Y}$ | (4.24) |

Figure 4-9 Useful Boolean Algebra Theorems.

The Absorption and Logic Adjacency can be proven exhaustively, but it is easier to prove them using the properties in Figure 4-8. For example, the Logic Adjacency Theorem (2.23a) can be proven as follows:

$$XY + X\bar{Y} = X(Y + \bar{Y}) = X \cdot 1 = X.$$

De Morgan's theorem can only be proven exhaustively because it makes use of the relationship between **and** and **or** that is not assumed by any of the properties in Figure 4-8. The proof of De Morgan's theorem is left as an exercise for the student.

De Morgan's theorem is interesting because it can be applied to an expression of arbitrary complexity. To simplify a complemented expression, simply exchange the **and** and **or** operators, then complement each variable or subexpression. Often it is a good idea to insert explicit parenthesis and **and** operators (\cdot) before applying De Morgan's theorem.

Example 4.4. Apply De Morgan's theorem to simplify the expression $\overline{(\bar{A} + B)\bar{C}\bar{D}}$.

Solution: First, insert an explicit **and** operator (explicit parentheses are not necessary):

$$\overline{(\bar{A} + B) \cdot \bar{C}\bar{D}}$$

Then, exchange \cdot and $+$, and complement \bar{A} , B and $\bar{C}\bar{D}$:

$$\overline{(\bar{A} + B) \cdot \bar{C}\bar{D}} = (\bar{\bar{A}} \cdot \bar{B}) + \overline{\bar{C}\bar{D}} = A\bar{B} + CD$$

The Logical Adjacency theorem is the basis for most simplification of logic expressions. Recall the expression for PRIME from the example design in Section 2.2:

$$PRIME = \bar{B}_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_1\bar{B}_0 + B_2\bar{B}_1B_0 + B_2B_1B_0 \quad (4.6)$$

Using the logic adjacency theorem, $\bar{B}_2\bar{B}_1\bar{B}_0 + \bar{B}_2B_1\bar{B}_0 = \bar{B}_2\bar{B}_1$ and $B_2\bar{B}_1B_0 + B_2B_1B_0 = B_2B_0$, so

$$PRIME = \bar{B}_2\bar{B}_1 + B_2B_0 \quad (4.25)$$

Example 4.5. Find another simplified expression for PRIME based on Equation (4.10)

Solution: Equation (4.10) states

$$PRIME = (B_2 + B_1 + B_0)(B_2 + B_1 + \bar{B}_0)(\bar{B}_2 + B_1 + B_0)(\bar{B}_2 + \bar{B}_1 + B_0)$$

Using the logic adjacency theorem, $(B_2 + B_1 + B_0)(B_2 + B_1 + \bar{B}_0) = B_2 + B_1$ and $(\bar{B}_2 + B_1 + B_0)(\bar{B}_2 + \bar{B}_1 + B_0) = \bar{B}_2 + B_0$, so

$$PRIME = (B_2 + B_1)(\bar{B}_2 + B_0)$$

Obviously, reducing a complex expression like $XY + X\bar{Y}$ to X has a substantial benefit. What is not obvious is that sometimes the process must be reversed to find the simplest form. Consider how you would approach the logic expression below:

$$AB + BC + \bar{A}B\bar{C} \quad (4.26)$$

Often, the trick is to expand the terms by applying the logic adjacency theorem in reverse, then combine terms in a different way. In this case we can replace AB with $ABC + AB\bar{C}$ and get:

$$ABC + AB\bar{C} + BC + \bar{A}B\bar{C} \quad (4.27)$$

By absorption, $ABC + BC = BC$, and by logic adjacency, $AB\bar{C} + \bar{A}B\bar{C} = B\bar{C}$. Making these substitutions we have

$$BC + B\bar{C}, \quad (4.28)$$

Which, through one more application of logic adjacency, reduces to B .

Example 4.6 Simplify the expression for the hot water valve for the washing machine design:

$$HOT_VALVE = WASH \cdot (HOT + WARM) \cdot \overline{WARM \cdot LID}$$

Solution: First, apply De Morgan's theorem to replace $\overline{WARM \cdot LID}$ with $(\overline{WARM} + \overline{LID})$, then reduce using properties (2.21a) twice, (2.16b) and (2.13a):

$$\begin{aligned} HOT_VALVE &= WASH \cdot (HOT + WARM) \cdot (\overline{WARM} + \overline{LID}) \\ &= WASH \cdot (HOT \cdot \overline{WARM} + HOT \cdot \overline{LID} + WARM \cdot \overline{WARM} + WARM \cdot \overline{LID}) \\ &= WASH \cdot (HOT \cdot \overline{WARM} + HOT \cdot \overline{LID} + 0 + WARM \cdot \overline{LID}) \\ &= WASH \cdot (HOT \cdot \overline{WARM} + HOT \cdot \overline{LID} + WARM \cdot \overline{LID}) \end{aligned}$$

Now, use logic adjacency in reverse to replace $HOT \cdot \overline{LID}$ with $HOT \cdot \overline{WARM} \cdot \overline{LID} + HOT \cdot WARM \cdot \overline{LID}$:

$$HOT_VALVE = WASH \cdot (HOT \cdot \overline{WARM} + HOT \cdot \overline{WARM} \cdot \overline{LID} + HOT \cdot WARM \cdot \overline{LID} + WARM \cdot \overline{LID})$$

Finally, combine terms using absorption:

$$HOT_VALVE = WASH \cdot (HOT \cdot \overline{WARM} + WARM \cdot \overline{LID})$$

The astute reader may have noticed that the term $HOT \cdot \overline{WARM}$ in the solution of Example 4.4 seems redundant. If HOT is true, $WARM$ has to be false, right? That's correct, but the information that HOT and $WARM$ are mutually exclusive was lost when we derived the logic expression in Section 4.1. A solution to this problem will be presented in Chapter 7.

One simplification approach that often works when some terms in an expression contain more variables than others, is to look for ways to introduce new terms that can combine with the larger terms using logic adjacency. For example, consider the following expression:

$$AB + \bar{A}BC \quad (4.29)$$

It is possible to use absorption to introduce a new term in the expression because $AB = AB + ABC$:

$$AB + ABC + \bar{A}BC \quad (4.30)$$

By logic adjacency, $ABC + \bar{A}BC = BC$, so the expression simplifies to $AB + BC$.

Example 4.7 Simplify the equation $F = \overline{AB + BC + D} + \bar{A}BC\bar{D} + \bar{B}C\bar{D}$.

Solution: First, apply De Morgan's Theorem to get:

$$F = (A + \bar{B})(B + \bar{C})\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{B}C\bar{D}.$$

Repeated application of Property 4.21a gives us:

$$F = (AB + A\bar{C} + \bar{B}B + \bar{B}\bar{C})\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{B}C\bar{D} = AB\bar{D} + A\bar{C}\bar{D} + \bar{B}C\bar{D} + \bar{A}\bar{B}C\bar{D} + \bar{B}C\bar{D}$$

Note that $AB\bar{D} = AB\bar{D} + ABC\bar{D}$ and that $ABC\bar{D} + \bar{A}\bar{B}C\bar{D} = AC\bar{D}$, so

$$F = AB\bar{D} + A\bar{C}\bar{D} + \bar{B}C\bar{D} + AC\bar{D} + \bar{B}C\bar{D}.$$

Now $A\bar{C}\bar{D} + AC\bar{D} = A\bar{D}$ and $\bar{B}C\bar{D} + \bar{B}C\bar{D} = \bar{B}C$. Making these substitutions we have:

$$F = AB\bar{D} + A\bar{D} + \bar{B}C.$$

Finally, $AB\bar{D} + A\bar{D} = A\bar{D}$, so $F = A\bar{D} + \bar{B}C$.

Example 4.8. Simplify the expression for UNLOCK in Example 4.1.

Solution: From Example 2.1, the logic expression for UNLOCK is

$$\begin{aligned} \text{UNLOCK} &= (\text{KEY1} + \text{KEY2}) \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2} \cdot \overline{\text{BUSINESS_HOURS}} \\ &= \text{KEY1} \cdot \text{BUSINESS_HOURS} + \text{KEY2} \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2} \cdot \overline{\text{BUSINESS_HOURS}} \end{aligned}$$

Note that

$$\text{KEY2} \cdot \text{BUSINESS_HOURS} = \text{KEY2} \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2} \cdot \text{BUSINESS_HOURS}$$

and

$$\text{KEY1} \cdot \text{KEY2} \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2} \cdot \overline{\text{BUSINESS_HOURS}} = \text{KEY1} \cdot \text{KEY2}.$$

After making these substitutions, we have

$$\text{UNLOCK} = \text{KEY1} \cdot \text{BUSINESS_HOURS} + \text{KEY2} \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2}$$

Or:

$$\text{UNLOCK} = (\text{KEY1} + \text{KEY2}) \cdot \text{BUSINESS_HOURS} + \text{KEY1} \cdot \text{KEY2}$$

Exercises

1. A manufacturer of home security systems wants to integrate a smoke detector and a carbon monoxide detector into an existing intruder alarm. When one or more of the three alarm conditions is detected, an audible alarm is generated. When a responder arrives, he or she can silence the alarm with a key switch. (a) list the input and output signals that the system requires, (b) write down a verbal description for what the system should do, (c) define the Boolean variables you would use for input and output, and (d) write a logic expression for the output using Boolean Algebra.
2. A digital system takes three inputs and generates an output of a one if and only if the total number of (input) ones is odd. (a) Write a truth table for this circuit. (b) Find a logic expression for this system using minterms. (c) Find a logic expression for this system using maxterms.
3. Simplify the following logic expressions:
 - (a) TBD
 - (b) TBD
 - (c) TBD
4. Simplify the following logic expressions.
 - (a) TBD
 - (b) TBD
 - (c) TBD
5. A digital system takes three inputs and generates an output of a one if and only if the total number of (input) ones is two or more. (a) Write a truth table for this circuit. (b) Find a logic expression for this system using minterms. (c) Simplify your answer. (There should be a total of 3 **ands** and 2 **ors**.) (d) Find a logic expression for this system using maxterms. (e) Simplify your answer. (There should be a total of 3 **ors** and 2 **ands**.)
6. A digital system takes a 3-bit binary number (B_2, B_1, B_0) in the range 1 to 6 and lights seven LEDs representing the pips on a die. There are four outputs designated *a-d* as shown in Figure 4-10. (Pips *a* through *c* always light in pairs, so only four outputs are required.)
 - (a) Write a truth table for this system. (Include all four outputs on a single table.) Hint: It will simplify the logic if an input of 0 lights no pips and an input of 7 lights all pips. (b) Find an unsimplified logic expression for each output *a-d*. (c) Simplify each logic expressions you derived in part (b). As a check, the number of operators (**and** or **or**) for each simplified output are given: *a*:1, *b*:1, *c*: 0, *d*: 0.

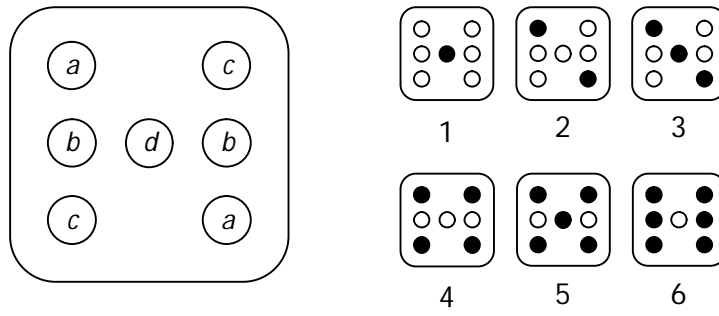


Figure 4-10 Die pips (Problem 3)

Bibliography