Chapter 3

Boolean Algebra

3.1 Why Boolean Algebra?

We have represented digital logic circuits in three different ways: as a circuit diagram, as a truth table, and as a Boolean expression. At this point in your study of digital logic, you should be comfortable shifting between these three representations. For example, given a circuit diagram, you should be comfortable developing a Boolean expression that represents the circuit diagram.

Most students—at this point—have an appreciation for circuit diagrams and truth tables, but not so much for Boolean algebra. Think about this: Suppose your friend told you with excitement that he had designed a new digital logic circuit, and he wanted to share it with you. Suppose, with great anticipation he handed you a paper with his design, and, looking down at the paper you saw…a Boolean expression. Your first reaction would likely be: “Uh, that’s nice, but could you show me the circuit diagram or the truth table?”

In Chapter 2 we saw that Boolean expressions serve as a useful intermediary between circuit diagrams and truth tables. For instance, if given a circuit diagram and asked for the equivalent truth table, it is more convenient to first develop the Boolean expression corresponding to the circuit diagram, and then, from this Boolean expression, develop the truth table. But do Boolean expressions have any value in their own right, apart from serving in this mediating role?

This chapter will convince you that the answer to the preceding question is: Yes!

3.2 Logical Equivalence

Consider the circuit diagram shown below:

![Circuit Diagram]

We can develop the truth table in the usual way, using the techniques presented in Chapter 2. The Boolean expression governing the output is ________ and the truth table is:

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
Deleting the intermediate columns, the truth table for the circuit is:

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Does this truth table look familiar? It should! This is the truth table for the OR gate!

Think about what this means. Looking at just the inputs and the outputs, the circuit diagram on the preceding page operates precisely the same as an OR gate. An observer, examining just the inputs and output, would have no way of knowing if the circuit was an OR gate, or the more complex circuit on the previous page. These two circuits, shown below, are *logically equivalent*.

But the two circuits are certainly *not* physically equivalent. Most evidently, the circuit on the left has three gates, and the circuit on the right has one. If each gate has the same cost, the circuit on the left would cost three times as much as the circuit on the right. If each gate generated the same amount of undesired heat, the circuit on the left would need more cooling than the circuit on the right. If each gate occupied the same amount of precious space on an IC chip, the circuit on the left would need three times as much space as the circuit on the right. Although the two circuits above perform the same—in the sense that they have the same truth tables—the circuit on the right is far preferable.

Suppose we have a logic circuit governed by a complex Boolean expression and its attendant truth table. If a simpler Boolean expression provides the same identical truth table, then we say the simpler Boolean expression is logically equivalent to the complex Boolean expression. The circuit corresponding to the simpler Boolean expression will then provide the same input-output relationship as the original logic circuit.

More precisely, two Boolean expressions are *logically equivalent* if and only if they have identical output values for all possible combinations of values for the inputs. So, to test if two Boolean expressions $A$ and $B$ are logically equivalent, we construct truth tables for $A$ and $B$. If the truth values of $A$ and $B$ match for all rows in the truth table, then $A$ and $B$ are logically equivalent.

If we find that two Boolean expressions are logically equivalent, the circuits corresponding to these Boolean expressions will also be logically equivalent (i.e., they will both provide the same input/output relationship). This will in many cases allow us to replace a complex circuit with a simpler circuit.
Example

Are $p$ and $(p')'$ logically equivalent?

Solution: Constructing the truth table for $p$ and $q = (p')'$ we find:

The foregoing example means that if I have the circuit shown below (either by itself or as a component within a larger circuit)

![Logic Circuit Diagram]

I can replace it by a wire running directly from the input to the output:

![Wired Circuits Diagram]

Example

Show that $a + bc$ is logically equivalent to $(a + b)(a + c)$. Draw the logic circuits for each.

Solution:
Example

Your friend says that \(a(a' + b) = ab'\). Is he correct?

Solution:

3.3 Boolean Algebra

A number of logical equivalences are well known and are presented in a table on the next page. This table appears in the textbook *Introduction to Logic Design* by Alan Marcovitz.

Each of the properties can be proven by showing the logical equivalence of the Boolean expressions on each side of the equals sign, using the approach described above. In fact, you should take a moment to convince yourself that by working through the examples in Section 3.2, you have proven properties P10a, P7 and P8b.

Again, the point: If we have a Boolean expression \(a + a' b\), we can use Property 10a to immediately replace this with the expression \(a + b\). Ditto with the properties in the rest of the table.

Note also that the properties are general in that each of the terms can be replaced by more complex expressions. For example, Property 1a, says that \(a + b = b + a\). If we substitute \(xy\) for \(a\) and \(w' z'\) for \(b\), Property 1a becomes \(xy + w' z' = w' z' + xy\).

We now are equipped to describe how to use Boolean algebra to simplify logic circuits. Given a digital logic circuit, we can attempt to design a simpler digital logic circuit that performs the equivalent function by performing the following steps:

- Find the Boolean expression that corresponds to this logic circuit.
• Use the properties of Boolean algebra from the table below to find a simpler logically equivalent Boolean expression.

• Given this simpler Boolean expression, draw the digital logic circuit that represents this expression. This simpler circuit will perform precisely the same way as the original (more complex) circuit.

Example

Suppose we find that a logic circuit has the Boolean expression \( x' y z' + x' y z \). Simplify this circuit using Boolean algebra. Draw the circuits before and after simplification.

Solution: \( x' y z' + x' y z \)

Example

Simplify the expression \( xy' + xyz \). Draw the circuits before and after simplification.

Solution: \( xy' + xyz \)
Example

Simplify the digital logic circuit shown below.

Solution:
Example

Simplify the expression \(( w' + x' + y + z' )( w' + x' + y + z )\). Draw the circuits before and after simplification.

Solution:

Having done some examples, you might now be wondering: *What is the precise step-by-step formal algorithmic method for simplifying expressions using the properties of Boolean algebra?*

Example

Simplify the expression \(xyz + x'y + x'y'\)

Example

Simplify the expression \(( x + y)( x + y + z' ) + y'\)
3.4 Boolean Algebra—A Closer Look

The Simpler Properties

Some of the properties of Boolean algebra are not intuitive at all. Referring to the table on page 5, is Property 14a intuitively obvious? (Answer: No!)

But Properties 1-7 are (or should soon be!) intuitively obvious. Of course each property could be proven by constructing a truth table, but let’s go through the properties shown in the left-hand column, one-by-one, offering an intuitive explanation of each.

Consider Property P1a, the Commutative Property, which says \( a + b = b + a \). The truth table for the OR operation is:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Suppose we take the columns for \( a \) and \( b \) and swap them. Only the second and third lines of this truth table would change (they would swap positions). But for both of these lines, the output is a one. So the output, \( b + a \), is the same as before (for \( a + b \)).

Example

Explain in words why Property P2a, the Associate Property, which says that \( a + (b + c) = (a + b) + c \), makes sense.

Solution:

Example

Explain Property P2a in terms of digital logic circuits.

Solution:

Example

Explain in words why Property P3a, the Identity Property, which says that \( a + 0 = a \), makes sense.

Solution:
Example

Explain in words why Property P4a, the Null Property, which says that $a + 1 = 1$, makes sense.
Solution:

Example

Explain in words why Property P5a, the Complement Property, which says that $a + a' = 1$, makes sense.
Solution:

Example

Explain in words why Property P6a, the Idempotency Property, which says that $a + a = a$, makes sense.
Solution:

Example

Explain in words why Property P7, the Involution Property, which says that $(a')' = a$, makes sense.
Solution:

Example

Suppose I have only several three-input AND gates available, but I want to implement a two-input AND gate. How can I do this?
Solution:

Example

Suppose I have only several three-input OR gates available, but I want to implement a two-input OR gate. How can I do this?
Solution:
Duality

Consider again the truth table for the simple two-input AND gate:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Suppose, in the truth table above, we change every zero to a one, and every one to a zero. Let’s look at the resulting truth table:

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

Look carefully at this truth table. What should we label the output column?

So, starting with the truth table for an AND gate, if we change every zero to a one, and every one to a zero, we arrive at the correct truth table for an OR gate. You should convince yourself that if we had instead started with the truth table for an OR gate and complemented all entries, we would have arrived at the correct truth table for an AND gate.

More generally, in any property $X$, if we change all the 0’s to 1’s and all the 1’s to 0’s, and change all ANDs to ORs and all ORs to ANDs, we will arrive at another correct property, called the dual of $X$.

You may have noticed in the table on page 5 that the properties are presented in pairs; next to Property P1a there is a Property P1b, next to Property P2a there is a Property P2b, and so forth. The properties in the Table are presented in dual pairs.

To obtain the dual pair of property $X$:

- Change all ORs in $X$ to ANDs and change all ANDs in $X$ to ORs
- Change all 1’s in $X$ to 0’s and change all 0’s in $X$ to 1’s

It is important to note that the dual of $X$ is in no way equal or equivalent to $X$. All we can say is that if $X$ is a valid property, then the dual of $X$ will also be valid.

**Example**

Show that P2b is the dual of P2a.

Solution: We start with P2a: $a + (b + c) = (a + b) + c$
Example

Show that P3b is the dual of P3a.

Solution: We start with P3a: $a + 0 = a$

Example

Show that P5aa is the dual of P5bb.

Solution: We start with P5bb: $a' \cdot a = 0$

DeMorgan’s Laws

DeMorgan’s Laws, listed in the table of properties as P11a, P11b, P11aa, P11bb are used so frequently that they deserve special mention. Let’s first convince ourselves that this is true by proving P11b.

Example

Prove Property 11b: $(pq)' = p' + q'$.

Solution:

As a practical matter, Property P11b means that the two logic circuits below are interchangeable:

Circuit 1

Circuit 2

Note also that the circuit on the left is equivalent to a NAND gate.

Example

Prove Property 11a, $(p + q)' = p'q'$ without using a truth table.

Solution:
As a practical matter, Property P11a means that the two logic circuits below are interchangeable:

Note also that the circuit on the left is equivalent to a NOR gate.

Example

Simplify the expression $(p' q)' (p + q)$

Solution:

Consensus

At this point you should be able to interpret all of the properties of Boolean algebra shown on page 5—with one exception. P13a (and its dual: Property 13b) look somewhat mysterious with $t_1$’s and $t_2$’s running around. P13 is called the “Consensus Property”.

Here is the idea: Suppose, in a SOP expression, we have two product terms, where exactly one variable appears uncomplemented in one of the product terms and complemented in the other. In such an instance, the product of all of the remaining literals in both of the product terms is called the consensus.

Example

Suppose we have two product terms: $a' b' c'$ and $a' be'$. Do we have a consensus term?

Solution:

Example

Suppose we have two product terms: $a' bcd$ and $ab'ef$. Do we have a consensus term?

Solution:

The Consensus Property states that if we have a consensus term that also appears in the SOP expression, the consensus term can be deleted.

$$a t_1 + a' t_2 + t_1 t_2 = a t_1 + a' t_2$$
Reduce the following Boolean expression: \( a' b' c' + e' bf + a' c' e' f \)

Solution:

Reduce the following Boolean expression: \( bc' + abd + acd \)

Solution:

Reduce the expression \( a' c' + a' b + b' c' \)

Solution:

### 3.5 The Complement of an Expression

We will see that it is often useful to compute the complement of a Boolean expression. Specifically, given the Boolean expression \( f \), it is often useful to determine \( f' \). DeMorgan’s Law is used (repeatedly—as many times as necessary) to find the complement of a Boolean expression.

Suppose a Boolean expression, denoted \( f \), is given by \( f = wx' + xy' + xz \). Determine the complement of \( f \).

Solution:

Suppose a Boolean expression, denoted \( f \), is given by \( f = a' bc' + ab' c' + abc \). Determine \( f' \).

Solution:
3.5 The SOP and POS Forms

Before moving forward to look at design problems, let’s pause and do an example that shows again the point of manipulating this Boolean algebra. While we are at it, we will also introduce a shorthand notation (the sigma notation) for the sum of products (SOP) form.

Suppose we are given the following truth table, and asked to implement it as a digital logic circuit.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Output</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>

Recall that we form a Boolean expression that governs this truth table by examining all the rows where the output is 1. For each of these rows, we form a product term, called a minterm, which will have all variables included. Recall that to form the minterm for each row, each variable is complemented if its value is zero in the column for the specific row under examination; otherwise it is uncomplemented.

We then sum these products together, forming a Sum of Products (SOP) expression.

The SOP expression for the truth table above is:

With this notation, we would write the output above as

\[ \text{Output} = \sum m(4, 5, 6, 7) \]
Example

Sketch the digital logic circuit which implements this SOP expression.

Solution:

---

Example

Use the properties of switching algebra to simplify the SOP expression and sketch the digital logic circuit for the simplified expression.

Solution:
In some applications we would prefer to have a product of sums (POS) form instead of a sum of products form. In terms of logic gates, a POS form would consist of the AND-ing of a number of OR-ed terms.

Terminology:
- A *sum term* is one or more literals connected by OR operators
- A *maxterm* is a sum term that includes every variable of the problem (complemented or uncomplemented)
- A *product of sums* (POS) is one or more sum terms connected by AND operators
- A *canonical product* is a POS expression in which all of the sum terms are maxterms

The easiest way to obtain the POS expression for a truth table is:
- Let the Boolean expression for the truth table be denoted $f$
- Find the SOP expression for $f'$ by considering those lines in the truth table for which the function is 0
- Complement $f'$ to get a POS expression for $f$

**Example**

Find the POS expression for the truth table shown on the top of page 15.

Solution: