1
Boolean Algebra

The vast majority of computers are digital computers; that is, computers based on a set of two numbers: $\mathbb{B} = \{0, 1\}$. We call the mathematics based on these two numbers Boolean algebra (named after the Irish mathematician George Boole). A Boolean variable or bit can thus take only two different values: either 0 or 1. We call $f(A_1, A_2, \ldots, A_n)$ a Boolean function of n independent Boolean variables A_1, A_2, …, A_{n-1}, and A_n. It takes either the value 0 or the value 1, depending on the values of its arguments $A_1, A_2, \ldots, A_n$. This dependency is fully described using a truth table, which tells us which value f takes for each of the 2^n different values of the (Boolean) vector $(A_1, A_2, \ldots, A_n)$.

In the present chapter, we will survey some properties of Boolean functions, which will allow us to gain a good understanding of binary reversible logic circuits. First, we will take a close look at Boolean functions $f(A)$ of a single variable, then at Boolean functions $f(A_1, A_2)$ of two variables, before we discuss Boolean functions $f(A_1, A_2, \ldots, A_n)$ of an arbitrary number of Boolean variables. Besides recording a Boolean function unambiguously by writing down its truth table, we can also fully define a Boolean function by means of a (Boolean) formula. There are many ways to write down such a formula. We will discuss some standard ways: the minterm expansion, the maxterm expansion, the Reed–Muller expansion, and the minimal ESOP expansion. Finally, we will define a few special classes of Boolean functions: true functions and balanced functions, linear functions, affine linear functions, and monotonic functions.

1.1
Boolean Functions of One Variable

There are only four Boolean functions $f(A)$ of a single Boolean variable A. Table 1.1 shows the four corresponding truth tables. However, two of these functions are not really dependent on A; they are constants:

$f(A) = 0$ (Table 1.1a)
$f(A) = 1$ (Table 1.1b) .

Reversible Computing. Alexis De Vos

ISBN: 978-3-527-40992-1

Table 1.1 Truth table of the four Boolean functions $f(A)$: (a) the constant function 0, (b) the constant function 1, (c) the identity function, and (d) the NOT function.

A	f
0	0
1	0

(a)

A	f
0	1
1	1

(b)

A	f
0	0
1	1

(c)

A	f
0	1
1	0

(d)

We thus have only two true functions (or proper functions) of A:

$$f(A) = A \quad \text{(Table 1.1c)}$$
$$f(A) = \overline{A} \quad \text{(Table 1.1d)} .$$

Here, we have introduced the following shorthand notation for the inverting function or NOT function:

$$\overline{X} = \texttt{NOT}\ X .$$

1.2 Boolean Functions of Two Variables

There are $2^4 = 16$ different Boolean functions of two variables[2]. Table 1.2 shows them all. However, some of these functions $f(A, B)$ are not actually functions of A and B; two functions are independent of both A and B. They are constants:

$$f_0(A, B) = 0$$
$$f_{15}(A, B) = 1 .$$

Another four functions are in fact functions of a single variable; f_3 and f_{12} are independent of B, whereas both f_5 and f_{10} are independent of A:

$$f_3(A, B) = A$$
$$f_5(A, B) = B$$
$$f_{10}(A, B) = \overline{B}$$
$$f_{12}(A, B) = \overline{A} .$$

This leaves only $16 - 2 - 4 = 10$ functions that are truely dependent on both A and B. We call them *true functions* of A and B.

2) Besides using the notation $A_1, A_2, \ldots, A_n$ for the variables, we will also use the letters $A, B, C, \ldots$ whenever this is more convenient.

Table 1.2 Truth table of all sixteen Boolean functions $f_i(A, B)$.

AB	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0 0	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1
0 1	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1 0	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1 1	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1

Table 1.3 Truth tables of three basic Boolean functions: (a) the AND function, (b) the OR function, and (c) the XOR function.

AB	f
0 0	0
0 1	0
1 0	0
1 1	1

(a)

AB	f
0 0	0
0 1	1
1 0	1
1 1	1

(b)

AB	f
0 0	0
0 1	1
1 0	1
1 1	0

(c)

Three out of the ten true functions of two variables (i.e., f_1, f_7, and f_6) are well known: the AND function, the OR function, and the XOR function (also known as the EXCLUSIVE OR). Table 1.3 gives the corresponding truth tables. We will use the following shorthand notations for these basic Boolean functions:

$$\begin{aligned} X\,Y &= X \,\texttt{AND}\, Y \\ X + Y &= X \,\texttt{OR}\, Y \\ X \oplus Y &= X \,\texttt{XOR}\, Y \ . \end{aligned}$$

The remaining $10 - 3 = 7$ functions are considered a combination of the NOT, AND, OR, and XOR functions. For example,

$$f_2(A, B) = A \,\texttt{AND}\, (\texttt{NOT}\, B) = A\overline{B} \ .$$

For convenience, the NOT of an AND is called a NAND, the NOT of an OR is called a NOR, and the NOT of a XOR is called a NXOR. For example,

$$f_8(A, B) = \texttt{NOT}\, (A \,\texttt{OR}\, B) = \overline{A + B} = A \,\texttt{NOR}\, B \ .$$

NXOR is also called XAND, with the shorthand notation $X \odot Y$.

We observe that all six functions AND, OR, XOR, NAND, NOR, and NXOR are commutative:

$$X \,\texttt{AND}\, Y = Y \,\texttt{AND}\, X \ ,$$

and there are similar identities for the other five functions. This is not the case for any function $f(X, Y)$. For example, the function f_2 is not commutative:

$$f_2(X, Y) \neq f_2(Y, X) \quad \text{but} \quad f_2(X, Y) = f_4(Y, X)\,.$$

We end this section by stressing that there is no fundamental difference between the functions AND, OR, NAND, and NOR. They all are true functions that have either one 0 and three 1s or three 0s and one 1 in their truth tables. The functions XOR and NXOR are fundamentally different: they display two 0s and two 1s in their truth tables. This important distinction between AND, OR, NAND, and NOR on the one hand and XOR and NXOR on the other was stressed by Yokoyama *et al.* [1]. The function A XOR B is *injective* in its first argument, as is NXOR. This means that, for each value of B, the equality A XOR $B = A'$ XOR B necessarily implies $A = A'$. The reader can easily verify that this is not the case with the AND function for example: A AND $0 = A'$ AND 0 does not imply $A = A'$ (as we could have $A = 0$ and $A' = 1$). These facts will have far-reaching consequences.

1.3 Boolean Functions of *n* Variables

There are 2^{2^n} different Boolean functions of n variables. Each is represented by a truth table consisting of $n + 1$ columns and 2^n rows. Table 1.4 gives an example for $n = 3$. This function is just one of the $2^8 = 256$ possible for $n = 3$. Among these, only 218 are true functions of the three variables A, B, and C. Among the 256 functions for $n = 3$, seventy are so-called balanced functions; that is, functions that have an equal number of 1s and 0s in the output column. The function shown in Table 1.4 is both true and balanced.

It is important to use the expressions *true* and *balanced* carefully. For example, the function $A\overline{B}$ is a true function of A and B, but is an untrue function of A, B,

Table 1.4 Truth table of a function $f(A, B, C)$ of three variables.

ABC	f
0 0 0	0
0 0 1	1
0 1 0	1
0 1 1	0
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

and C. It is not a balanced function of A and B, but it is a balanced function of the three variables A, B, and C. The reader may also notice the following property: all untrue functions are balanced.

A truth table can be summarized by a single Boolean formula. However, there are multiple ways to write a given table as a Boolean expression [2]. We will now discuss a few of them.

1.3.1 The Minterm Expansion

Using the truth table, it is immediately possible to deduce the Boolean formula called the *minterm expansion*. For example, Table 1.4 yields:

$$f(A, B, C) = \overline{A}\,\overline{B}C + \overline{A}B\overline{C} + A\overline{B}\,\overline{C} + AB\overline{C}\ .$$

This consists of the OR of different terms. There are between 0 and 2^n different terms present. Each term is called a minterm and consists of the AND of exactly n literals. Here a literal is a letter, either inverted or not; thus, whereas X is called a letter, both X and $\overline{X}$ are called literals.

The algorithm for translating the truth table in the minterm expansion is straightforward: each row of the truth table with a 1 in the column furthest to the right yields one minterm. The latter consists of an AND of all input letters; an overlining is used if a 0 appears in the corresponding column, but not if a 1 appears in the corresponding column.

In the literature, such an expansion is sometimes referred to as a *sum of products*, because an OR resembles a sum in a way, while an AND resembles a product to some extext. The abbreviation SOP is also often used.

1.3.2 The Maxterm Expansion

As there is no fundamental difference between OR and AND, it is no surprise that there is a function expansion that is like the minterm expansion but has the roles of OR and AND interchanged. Such an expansion is an AND of ORs, and is called a *maxterm expansion*. In our example (Table 1.4), we have

$$f(A, B, C) = (A + B + C)(A + \overline{B} + \overline{C})(\overline{A} + B + \overline{C})(\overline{A} + \overline{B} + \overline{C})\ .$$

The algorithm for translating the truth table into the maxterm expansion is completely analogous to the minterm algorithm: each row of the truth table with a 0 in the column furthest to the right yields one maxterm; the latter consists of an OR of all input letters, with a bar if a 1 appears in the corresponding column, and not if a 0 appears in the corresponding column. The maxterm expansion is an example of a POS or *product of sums*.

1.3.3 The Reed–Muller Expansion

A fundamentally different expansion is obtained as follows. We apply to the minterm expansion the two identities

$$\begin{aligned} \overline{X} &= 1 \oplus X \\ X + Y &= X \oplus Y \oplus XY \,. \end{aligned} \tag{1.1}$$

This leads to an XOR of ANDs. The result is subsequently simplified by applying the identities

$$\begin{aligned} X \oplus X &= 0 \\ 0 \oplus X &= X \,. \end{aligned}$$

In our example (Table 1.4), we obtain

$$f(A, B, C) = A \oplus B \oplus C \oplus AB \,. \tag{1.2}$$

A Reed–Muller expansion (named after the American mathematicians Irving Reed and David Muller) is an example of an ESOP expansion; that is, an 'EXCLUSIVE-OR sum of products'. Thus, just like the OR, the XOR function is considered a kind of sum. We note that the Reed–Muller expansion is fundamentally different from the minterm and maxterm expansions because of the *injectivity* of the XOR operation.

In many respects, a Reed–Muller expansion of a Boolean function resembles the well-known Taylor expansion of ordinary calculus. Let us assume a function f of the real numbers x, y, and z. Then, the Taylor expansion around the point $(x, y, z) = (0, 0, 0)$ looks like

$$\begin{aligned} f(x, y, z) = {} & c_{000} + c_{100}x + c_{010}y + c_{001}z \\ & + c_{110}xy + c_{101}xz + c_{011}yz + c_{200}x^2 + c_{020}y^2 + c_{002}z^2 \\ & + c_{111}xyz + c_{210}x^2y + \dots \,. \end{aligned}$$

The Reed–Muller expansion of a function f of the Boolean numbers A, B, and C looks like

$$\begin{aligned} f(A, B, C) = {} & c_{000} \oplus c_{100}A \oplus c_{010}B \oplus c_{001}C \\ & \oplus c_{110}AB \oplus c_{101}AC \oplus c_{011}BC \oplus c_{111}ABC \,. \end{aligned}$$

There are three main differences:

- The Reed–Muller coefficients c_{ijk} can only be either 0 or 1.
- Each of the exponents i, j, and k in the monomial (also known as the 'piterm') $A^i B^j C^k$ can only be either 0 or 1; as a result:
- There are only a finite number (i.e., a maximum of 2^n) of Reed–Muller terms.

Once again, we must stress that there is no fundamental difference between AND and OR, or between XOR and XAND. As the Reed–Muller expansion is an XOR of ANDs, there is a similar (dual) expansion that is an XAND of ORs [3, 4]. For example, expansion (1.2) can be rewritten as

$$f(A, B, C) = A \odot C \odot (A + C) \odot (B + C) .$$

Unfortunately, such expressions are paid little attention in the literature.

We end this section by noting that (at least for $n > 1$), the Reed–Muller expansion of a balanced function lacks the highest-degree term $A_1 A_2 \ldots A_n$. In other words, the Reed–Muller coefficient $c_{11\ldots1}$ of a balanced function is zero.

1.3.4
The Minimal ESOP Expansion

In the Reed–Muller expansion, NOT functions are not allowed.[3] If we do allow NOT operations, the 'XOR of ANDs' expansion can be shortened. The shortest expansion (i.e., the one with the fewest literals) is called the minimal ESOP expansion.

The minimal ESOP expansion is quite different from the three above expansions (i.e., the minterm expansion, the maxterm expansion, and the Reed–Muller expansion) in two respects:

- It is not unique: two or even more minimal ESOP expansions of the same Boolean function may exist, and
- There is no straightforward algorithm for finding the minimal ESOP expansion(s) (except, of course, for an exhaustive search).

The last fact explains why minimal ESOPs are only known for Boolean functions with $n = 6$ or less [5, 6].
Our example function (Table 1.4) has two different minimal ESOPs:

$$\begin{aligned} f(A, B, C) &= A \oplus C \oplus \overline{A}B \\ &= B \oplus C \oplus A\overline{B} . \end{aligned}$$

Whereas the Reed–Muller expansion (1.2) needs five literals, these minimal ESOPs contain only four literals.

1.4
Linear Functions

A function $f(A_1, A_2, \ldots, A_n)$ is linear iff ('iff' means 'if and only if') its Reed–Muller expansion only contains terms with one letter:

$$f(A_1, A_2, \ldots, A_n) = c_1 A_1 \oplus c_2 A_2 \oplus \cdots \oplus c_n A_n .$$

3) In the present book we limit ourselves to so-called 'positive-polarity Reed–Muller expansions'. We thus ignore 'negative-polarity Reed–Muller expansions' and 'mixed-polarity Reed–Muller expansions' [2].

Table 1.5 Truth tables for (a) a linear function, (b) an affine linear function, and (c) a monotonic function.

ABC	f
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	0
1 1 0	1
1 1 1	0

(a)

ABC	f
0 0 0	1
0 0 1	1
0 1 0	0
0 1 1	0
1 0 0	0
1 0 1	0
1 1 0	1
1 1 1	1

(b)

ABC	f
0 0 0	0
0 0 1	1
0 1 0	0
0 1 1	1
1 0 0	1
1 0 1	1
1 1 0	1
1 1 1	1

(c)

Because each of the n Reed–Muller coefficients c_j can take one of two values ($c_j \in \{0, 1\}$), the reader can easily verify that there are 2^n different linear functions with n arguments. It is clear that the function defined by Table 1.4 is not linear: its Reed–Muller expansion (1.2) contains a second-degree term AB. In contrast, the function defined by Table 1.5a is linear, as it equals $A \oplus C$.

1.5 Affine Linear Functions

A function $f(A_1, A_2, \ldots, A_n)$ is *affine linear*[4] iff its Reed–Muller expansion contains only terms with either zero or one letter:

$$f(A_1, A_2, \ldots, A_n) = c_0 \oplus c_1 A_1 \oplus c_2 A_2 \oplus \cdots \oplus c_n A_n \ .$$

Because each of the $n + 1$ Reed–Muller coefficients c_j can take one of two values, there are 2^{n+1} different affine linear functions of n arguments. Among these, 2^n are linear. The function defined by Table 1.5b is an example of an affine linear function, as it equals $1 \oplus A \oplus B$.

4) There seems to be some confusion about the meaning of the word 'linear'. Let us consider the ordinary functions $f(x)$ of a real variable x. Sometimes all of the functions $f(x) = ax + b$ are said to be 'linear'; sometimes only the functions $f(x) = ax$ are considered 'linear'. In the present book, we follow the latter convention, so that the functions $ax + b$ are said to be 'affine linear'.

1.6 Monotonic Functions

We consider a function of n binary variables A_i. We use the different values of the vector $(A_1, A_2, \ldots, A_n)$ as the coordinates of an n-dimensional hypercube. We can represent a Boolean function by giving each corner of the hypercube a label $f(A_1, A_2, \ldots, A_n)$. We call a path that travels from the point $(0, 0, \ldots, 0)$ to the point $(1, 1, \ldots, 1)$ via consecutive steps that each increase a single coordinate A_i from 0 to 1 a *climbing path*. Such a path necessarily contains n steps, with each being an edge of the hypercube. Note that:

- There are n possible choices for the first step,
- There are $n - 1$ possible choices for the second step,
- ...
- There are $n - j + 1$ possible choices for the jth step.

This means that there are $n!$ different climbing paths.

A Boolean function $f(A_1, A_2, \ldots, A_n)$ is *monotonic* (or monotone or unate) iff its value increases along each climbing path: $f(A'_1, A'_2, \ldots, A'_n) \geq f(A''_1, A''_2, \ldots, A''_n)$ as soon as $A'_i \geq A''_i$ for all i satisfying $1 \leq i \leq n$. There is no closed formula for the number of monotonic functions [7]. Neither Table 1.4 nor Table 1.5a nor Table 1.5b is a truth table of a monotonic function. Indeed, for each of these three functions, vertex (0, 0, 1) of the hypercube has a label of 1, whereas vertex (1, 0, 1) of the hypercube has a label of 0, such that f does not increase along the climbing path (0, 0, 0)–(0, 0, 1)–(1, 0, 1)–(1, 1, 1). In contrast, the function defined by Table 1.5c is monotonic.

1.7 Boolean Derivative

Besides Boolean algebra, there is also Boolean calculus, which describes time-dependencies for example. At this point, it is sufficient to mention the three so-called subfunctions of a Boolean function $f(A_1, A_2, \ldots, A_{j-1}, A_j, A_{j+1}, \ldots, A_n)$:

$$\begin{aligned}
&f'(A_1, A_2, \ldots, A_{j-1}, A_j, A_{j+1}, \ldots, A_n) = \\
&\quad f(A_1, A_2, \ldots, A_{j-1}, 0, A_{j+1}, \ldots, A_n) \\
&f''(A_1, A_2, \ldots, A_{j-1}, A_j, A_{j+1}, \ldots, A_n) = \\
&\quad f(A_1, A_2, \ldots, A_{j-1}, 1, A_{j+1}, \ldots, A_n) \\
&f'''(A_1, A_2, \ldots, A_{j-1}, A_j, A_{j+1}, \ldots, A_n) = f' \oplus f'' \,.
\end{aligned}$$

All three functions are independent of A_j, and are thus untrue functions of $A_1, A_2, \ldots, A_n$. Sometimes [8, 9] f''' is called the partial derivative of f with respect to A_j, and it is then denoted $\frac{\partial f}{\partial A_j}$. The reason for this name may be made

Table 1.6 Truth tables of the three subfunctions f', f'', and f''' of the function $f(A, B, C)$ of Table 1.4.

BC	f'
0 0	0
0 1	1
1 0	1
1 1	0

(a)

BC	f''
0 0	1
0 1	0
1 0	1
1 1	0

(b)

BC	f'''
0 0	1
0 1	1
1 0	0
1 1	0

(c)

clearer by

$$\frac{\partial f}{\partial A_j} = \frac{f(A_j = 1) \oplus f(A_j = 0)}{1 \oplus 0} ,$$

which is a Boolean variant of the ordinary derivative of a real function $f(x_1, x_2, \ldots, x_n)$:

$$\frac{\partial f}{\partial x_j} = \lim_{a \to 0} \frac{f(x_j = a) - f(x_j = 0)}{a - 0} .$$

We apply the above to the example function $f(A, B, C)$ of Table 1.4 by choosing A_j to be equal to A. The subfunctions f' and f'' are found by merely slicing Table 1.4 into two halves, yielding Table 1.6a and Table 1.6b, respectively. The subfunction f''' is subsequently constructed by XORing the columns f' and f'', yielding Table 1.6c.

1.8 Boolean Decompositions

Subfunctions are particularly useful when implementing a Boolean function. Assume that we want to build a hardware circuit that realizes a function f of n variables. We have the following identities:

$$\begin{aligned} f &= f' \overline{A_j} + f'' A_j \\ &= f' \oplus f''' A_j . \end{aligned} \tag{1.3}$$

The former expression is called the Shannon decomposition (named after the American engineer Claude Shannon); the latter expression is known as the Davio decomposition (after the Belgian engineer Marc Davio).[5] As the subfunctions f',

5) We limit ourselves here to the so-called 'positive Davio decomposition'. We thus ignore the 'negative Davio decomposition' $f'' \oplus f''' \overline{A_j}$.

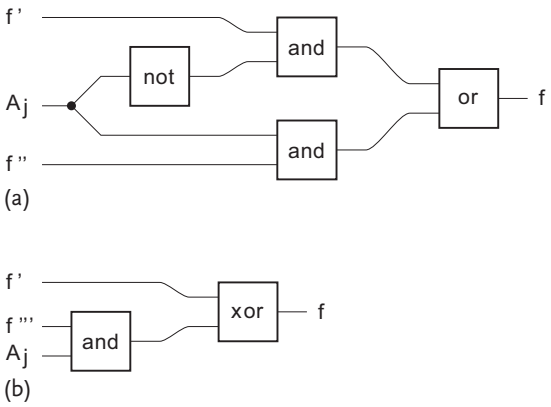

Figure 1.1 Circuit decompositions: (a) Shannon decomposition, (b) Davio decomposition.

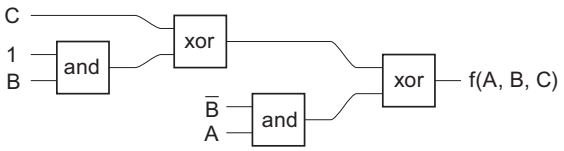

Figure 1.2 Davio decomposition.

f'', and f''' are functions of only $n-1$ variables, both identities allow us to reduce the original design problem to two smaller design problems. Applying such a decomposition over and over again (each time with another variable A_j) eventually allows the synthesis problem to be reduced to trivial functions (i.e., literals or constants). Figure 1.1 shows the two circuit decompositions.

Figure 1.2 shows the result of applying the Davio decomposition to the example function (Table 1.4) twice: once with $A_j = A$ (right side of the figure), and then with $A_j = B$ (left side).

1.9 Exercises for Chapter 1

Exercise 1.1

XOR and XAND are injective in the first argument; OR, NOR, AND, and NAND are non-injective in the first argument. Verify that the same properties hold in the second argument (why is this the case?).

Exercise 1.2

Which of the 16 functions $f(A, B)$ is both injective in the first argument and non-injective in the second argument?

Exercise 1.3
Prove the second identity in (1.1). Demonstrate that, if X and Y represent different minterms, then this simplifies to

$$X + Y = X \oplus Y \ .$$

Exercise 1.4
Apply (1.1) in order to obtain (1.2).

Exercise 1.5
Verify the dual identities of (1.1):

$$\overline{X} = 0 \odot X$$
$$XY = X \odot Y \odot (X + Y) \ .$$

Exercise 1.6
A Boolean function is said to be 'even' if it has an even number of 1s (and thus also an even number of 0s) in its truth table. Otherwise, it is said to be 'odd'. In other words, an even function has an even number of terms in its minterm expansion, while an odd function has an odd number of terms in its minterm expansion. Demonstrate the following properties:

- If both f and g are even, then $f \oplus g$ is even.
- If both f and g are odd, then $f \oplus g$ is even.
- If f is even and g is odd, then $f \oplus g$ is odd.

Exercise 1.7
Prove the property mentioned at the end of Section 1.3.3: that all balanced functions lack the highest-degree term $A_1 A_2 \ldots A_n$ in their Reed–Muller expansions. Demonstrate (with the help of a counterexample) that the inverse theorem (i.e., that all functions lacking the highest-degree Reed–Muller term are balanced) is false.

Exercise 1.8
Find the minterm expansion, the maxterm expansion, the Reed–Muller expansion, and the minimum ESOP expansions of the function $f_2(A, B)$ of Table 1.2.

Exercise 1.9
Check (1.3).