8.2. Boolean Functions, Applications

8.2.1. Introduction. A *Boolean function* is a function from $\mathbb{Z}_2^n$ to $\mathbb{Z}_2$. For instance, consider the *exclusive-or* function, defined by the following table:

x_1	x_2	$x_1 \veebar x_2$
1	1	0
1	0	1
0	1	1
0	0	0

The exclusive-or function can interpreted as a function $\mathbb{Z}_2^2 \to \mathbb{Z}_2$ that assigns $(1,1) \mapsto 0$, $(1,0) \mapsto 1$, $(0,1) \mapsto 1$, $(0,0) \mapsto 0$. It can also be written as a Boolean expression in the following way:

$$x_1 \veebar x_2 = (x_1 \wedge \bar{x}_2) \vee (\bar{x}_1 \wedge x_2)$$

Every Boolean function can be written as a Boolean expression as we are going to see next.

8.2.2. Disjunctive Normal Form. We start with a definition. A *minterm* in the symbols $x_1, x_2 \ldots, x_n$ is a Boolean expression of the form $y_1 \wedge y_2 \wedge \cdots \wedge y_n$, where each y_i is either x_i or $\bar{x}_i$.

Given any Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that is not identically zero, it can be represented

$$f(x_1, \ldots, x_n) = m_1 \vee m_2 \vee \cdots \vee m_k\,,$$

where $m_1, m_2, \ldots, m_k$ are all the minterms $m_i = y_1 \wedge y_2 \wedge \cdots \wedge y_n$ such that $f(a_1, a_2, \ldots, a_n) = 1$, where $y_j = x_j$ if $a_j = 1$ and $y_j = \bar{x}_j$ if $a_j = 0$. That representation is called *disjunctive normal form* of the Boolean function f.

Example: We have seen that the exclusive-or can be represented $x_1 \veebar x_2 = (x_1 \wedge \bar{x_2}) \vee (\bar{x_1} \wedge x_2)$. This provides a way to implement the exclusive-or with a combinatorial circuit as shown in figure 8.4.

8.2.3. Conjunctive Normal Form. A *maxterm* in the symbols $x_1, x_2 \ldots, x_n$ is a Boolean expression of the form $y_1 \vee y_2 \vee \cdots \vee y_n$, where each y_i is either x_i or $\bar{x}_i$.

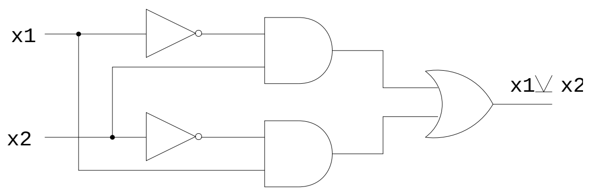

Figure 8.4. Exclusive-Or.

Given any Boolean function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ that is not identically one, it can be represented

$$f(x_1, \dots, x_n) = M_1 \wedge M_2 \wedge \cdots \wedge M_k \,,$$

where $M_1, M_2, \dots, M_k$ are all the maxterms $M_i = y_1 \vee y_2 \vee \cdots \vee y_n$ such that $f(a_1, a_2, \dots, a_n) = 0$, where $y_j = x_j$ if $a_j = 0$ and $y_j = \bar{x}_j$ if $a_j = 1$. That representation is called *conjunctive normal form* of the Boolean function f.

Example: The conjunctive normal form of the exclusive-or is

$$x_1 \veebar x_2 = (x_1 \vee x_2) \wedge (\bar{x}_1 \vee \bar{x}_2) \,.$$

8.2.4. Functionally Complete Sets of Gates. We have seen how to design combinatorial circuits using AND, OR and NOT gates. Here we will see how to do the same with other kinds of gates. In the following gates will be considered as functions from $\mathbb{Z}_2^n$ into $\mathbb{Z}_2$ intended to serve as building blocks of arbitrary boolean functions.

A set of gates $\{g_1, g_2, \dots, g_k\}$ is said to be *functionally complete* if for any integer n and any function $f : \mathbb{Z}_2^n \to \mathbb{Z}_2$ it is possible to construct a combinatorial circuit that computes f using only the gates $g_1, g_2, \dots, g_k$. *Example*: The result about the existence of a disjunctive normal form for any Boolean function proves that the set of gates $\{\text{AND}, \text{OR}, \text{NOT}\}$ is functionally complete. Next we show other sets of gates that are also functionally complete.

1. The set of gates $\{\text{AND}, \text{NOT}\}$ is functionally complete. Proof: Since we already know that $\{\text{AND}, \text{OR}, \text{NOT}\}$ is functionally complete, all we need to do is to show that we can compute $x \vee y$ using only AND and NOT gates. In fact:

   $$x \vee y = \overline{\bar{x} \wedge \bar{y}} \,,$$

   hence the combinatorial circuit of figure 8.5 computes $x \vee y$.

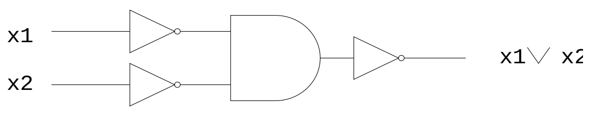

FIGURE 8.5. OR with AND and NOT.

2. The set of gates {OR, NOT} is functionally complete. The proof is similar:

$$x \wedge y = \overline{\bar{x} \vee \bar{y}}\,,$$

hence the combinatorial circuit of figure 8.6 computes $x \vee y$.

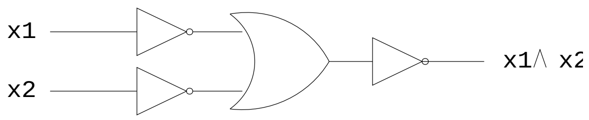

FIGURE 8.6. AND with OR and NOT.

3. The gate NAND, denoted ↑ and defined as

$$x_1 \uparrow x_2 = \begin{cases} 0 & \text{if } x_1 = 1 \text{ and } x_2 = 1 \\ 1 & \text{otherwise,} \end{cases}$$

is functionally complete.

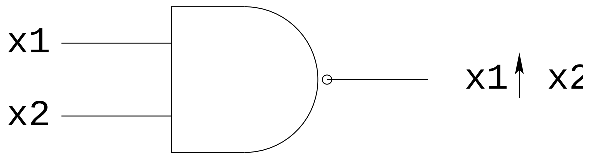

FIGURE 8.7. NAND gate.

Proof: Note that $x \uparrow y = \overline{x \wedge y}$. Hence $\bar{x} = \overline{x \wedge x} = x \uparrow x$, so the NOT gate can be implemented with a NAND gate. Also the OR gate can be implemented with NAND gates: $x \vee y = \overline{\bar{x} \wedge \bar{y}} = (x \uparrow x) \uparrow (y \uparrow y)$. Since the set {OR, NOT} is functionally complete and each of its elements can be implemented with NAND gates, the NAND gate is functionally complete.

8.2.5. Minimization of Combinatorial Circuits. Here we address the problems of finding a combinatorial circuit that computes a given Boolean function with the minimum number of gates. The idea is to simplify the corresponding Boolean expression by using algebraic

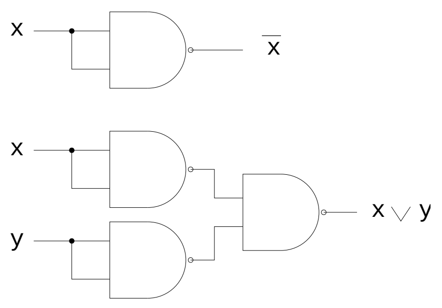

FIGURE 8.8. NOT and OR functions implemented with NAND gates.

properties such as $(E \wedge a) \vee (E \wedge \bar{a}) = E$ and $E \vee (E \wedge a) = E$, where E is any Boolean expression. For simplicity in the following we will represent $a \wedge b$ as ab, so for instance the expressions above will look like this: $Ea \vee E\bar{a} = E$ and $E \vee Ea = E$.

Example: Let $F(x, y, z)$ the Boolean function defined by the following table:

x	y	z	f(x,y,z)
1	1	1	1
1	1	0	1
1	0	1	0
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

Its disjunctive normal form is $f(x, y, z) = xyz \vee xy\bar{z} \vee x\bar{y}\bar{z}$. This function can be implemented with the combinatorial circuit of figure 8.9.

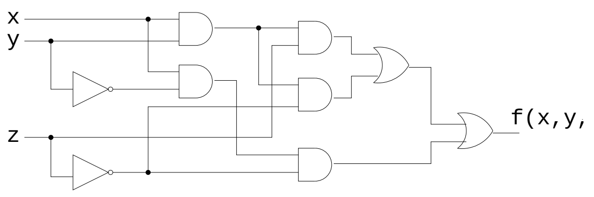

FIGURE 8.9. A circuit that computes $f(x, y, z) = xyz \vee xy\bar{z} \vee x\bar{y}\bar{z}$.

But we can do better if we simplify the expression in the following way:

$$\begin{aligned} f(x,y,z) &= \overbrace{xyz \vee xy\bar{z}}^{xy} \vee x\bar{y}\bar{z} \\ &= xy \vee x\bar{y}\bar{z} \\ &= x(y \vee \bar{y}\bar{z}) \\ &= x(y \vee \bar{y})(y \vee \bar{z}) \\ &= x(y \vee \bar{z}), \end{aligned}$$

which corresponds to the circuit of figure 8.10.

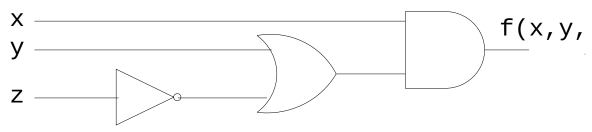

FIGURE 8.10. A simpler circuit that computes $f(x,y,z) = xyz \vee xy\bar{z} \vee x\bar{y}\bar{z}$.

8.2.6. Multi-Output Combinatorial Circuits. *Example*: *Half-Adder*. A half-adder is a combinatorial circuit with two inputs x and y and two outputs s and c, where s represents the sum of x and y and c is the carry bit. Its table is as follows:

x	y	s	c
1	1	0	1
1	0	1	0
0	1	1	0
0	0	0	0

So the sum is $s = x \veebar y$ (exclusive-or) and the carry bit is $c = x \wedge y$. Figure 8.11 shows a half-adder circuit.

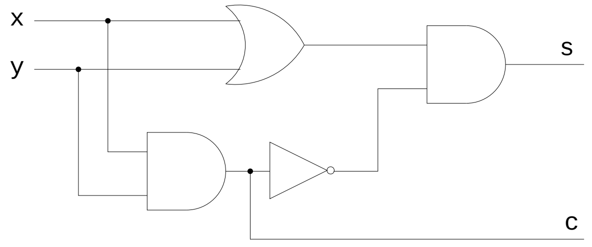

FIGURE 8.11. Half-adder circuit.