Chapter 1

Logic Design Section (will be Section 13.7)

...

1.1 A Brief Introduction to Switching Theory and Logic Design

Disclaimer: I'm still looking for a good application for drawing logic gates. The figures here are quite rough.

Early computers relied on many switches to perform the logical operations needed for computation. This was true as late as the 1970's when early personal computers such as the Altair (Figure 1.1.1) started to appear. Pioneering computer scientists such as Claude Shannon realized that the operation of these computers could be simplified by making use of an isomorphism between computer circuits and boolean algebra. The term **Switching Theory** was used at the time. Logical gates realized through increasingly smaller and smaller integrated circuits still perform the same functions as in early computers, but using purely electronic means. In this section, we give examples of some switching circuits. Soon afterward, we will transition to the more modern form of circuits that are studied in **Logic Design**, where gates replace switches. Out main goal is to give you an overview of how boolean functions corresponds to any such circuit. We will introduce the common system notation used in logic design and show how it corresponds with the mathematical notation of Boolean algebras. Any computer scientist should be familiar with both systems.

Figure 1.1.1: The Altair Computer, an early PC, by Todd Dailey, Creative Commons

The simplest switching device is the on-off switch. If the switch is closed/ ON, current will pass through it; if it is open/OFF, current will not pass through it. If we designate ON by 1, and OFF by 0, we can describe electrical circuits containing switches by Boolean expressions with the variables representing the variable states of switches or the variable bits passing through gates.

The electronics involved in these switches take into account whether we are negating a switch or not. For electromagnetic switches, a magnet is used to control whether the switch is open or closed. The magnets themselves may be controlled by simple ON/OFF switches. There are two types of electromagnetic switches. One is normally open (OFF) when the magnet is not activated, but activating the magnet will close the circuit and the switch is then ON. A separate type of switch corresponds with a negated switch. For that type, the switch is closed when the magnet is not activated, and when the magnet is activated, the switch opens. We won't be overly concerned with the details of these switches or the electronics corresponding to logical gates. We will simply assume they are available to plug into a circuit. For simplicity, we use the inversion symbol on a varible that labels a switch to indicate that it is a switch of the second type, as in Figure 1.1.3.

Standby power generators that many people have in their homes use a transfer switch to connect the generator to the home power system. This switch is open (OFF) if there is power coming from the normal municipal power supply. It stays OFF because a magnet is keeping it open. When power is lost, the magnet is no longer activated, and the switch closes and is ON. So the transfer switch is a normally ON switch.

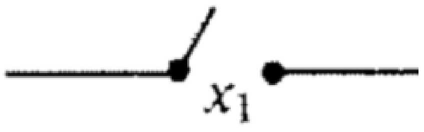

Figure 1.1.2: Representation of a normally OFF switch controlled by variable x_1

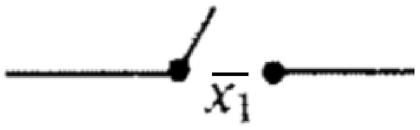

Figure 1.1.3: Representation of a normally ON switch controlled by variable x_1

The standard notation used for Boolean algebra operations in switching theory and logic design is $+$ for join, instead of $\vee$; and $\cdot$ for meet, instead of

$\wedge$. Complementation is the same in both notational systems, denoted with an overline.

The expression $x_1 \cdot x_2$ represents the situation in which a series of two switches appears in sequence as in Figure 1.1.4. In order for current to flow through the circuit, both switches must be ON; that is, they must both have the value 1. Similarly, a pair of parallel switches, as in Figure 1.1.5, is described algebraically by $x_1 + x_2$. Here, current flows through this part of the circuit as long as at least on of the switches is ON.

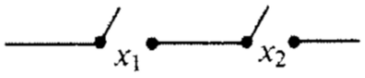

Figure 1.1.4: Two switches in AND configuration realizing $x_1 \cdot x_2$

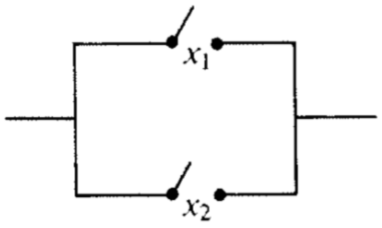

Figure 1.1.5: Two switches in OR configuration realizing $x_1 + x_2$

All laws and concepts developed previously for Boolean algebras hold. The only change is purely notational. We make the change in this section solely to introduce the reader to another frequently used system of notation.

Many of the laws of Boolean algebra can be visualized thought switching theory. For example, the distributive law of meet over join is expressed as

$$x_1 \cdot (x_2 + x_3) = x_1 \cdot x_2 + x_1 \cdot x_3.$$

The switching circuit analogue of the above statement is that the circuits in Figure 1.1.6 are equivalent. In circuit (b), the presence of two x_1's represents two electromagnetic switches controlled by the same magnet.

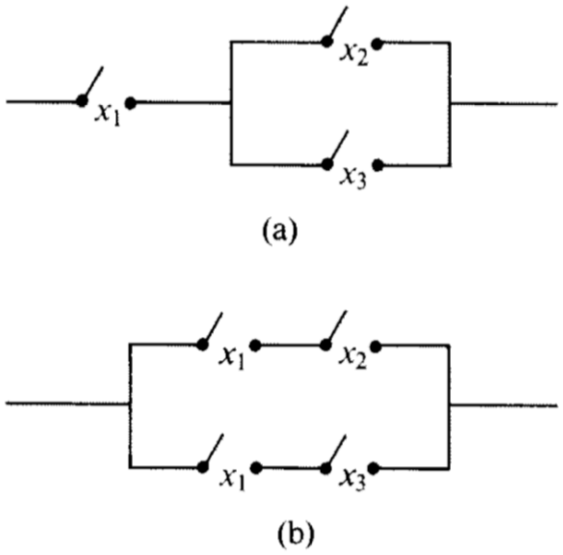

Figure 1.1.6: Two equivalent switching circuits

The circuits in a computer are now composed of large quantities of gates, which serve the same purpose as switches, but can be miniaturized to a great degree. For example, the OR gate, usually drawn as in Figure 1.1.7 implements the logical OR function. This happens electronically, but is equivalent to Figure 1.1.5. The AND gate, which is equivalent to two sequential switches is shown in Figure 1.1.7.

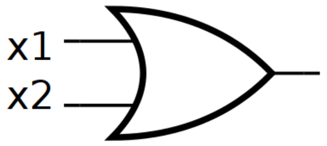

Figure 1.1.7: An OR gate

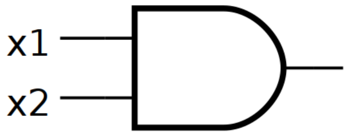

Figure 1.1.8: An AND gate

The complementation process is represented in a gate diagram by an inverter, as pictured in Figure 1.1.9.

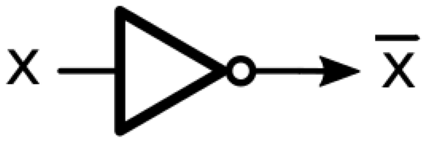

Figure 1.1.9: Inverter, or NOT gate

When drawing more complex circuits, multiple AND's or OR's are sometimes depicted using a more general gate drawing. For example if we want to depict an OR gate with three inputs that is ON as long as at least one input is ON, we would draw it as in Figure 1.1.10, although this would really be two binary gates, as in Figure 1.1.11. Both diagrams are realizing the boolean expression $x_1 + x_2 + x_3$. Strictly speaking, the gates in Figure 1.1.11 represent $(x_1 + x_2) + x_3$, but the associative law for join tells us that the grouping doesn't matter.

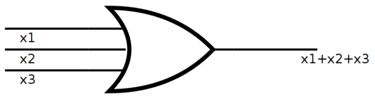

Figure 1.1.10: Simple version of a ternary OR gate

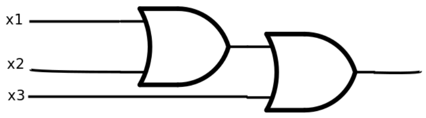

Figure 1.1.11: A ternary OR gate created with binary OR gates

In Figure 1.1.12, we show a few other commonly used gates, XOR, NAND, and NOR, which correspond to the boolean exressions $x_1 \oplus x_2$, $\overline{x_1 \cdot x_2}$, and $\overline{x_1 + x_2}$, respectively.

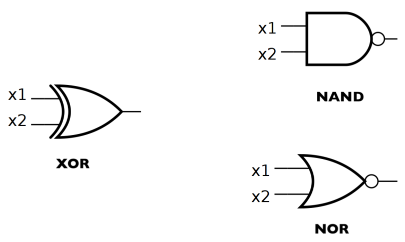

Figure 1.1.12

Let's start with a logic circuit and see how the laws of boolean algebra can help us simplify it.

Example 1.1.13 Simplification of a circuit. Consider the circuit in Figure 1.1.14. As usual, we assume that three inputs enter on the left and the output exits on the right.

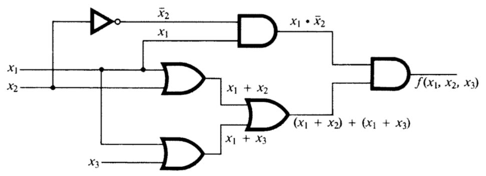

Figure 1.1.14: Initial gate diagram

If we trace the inputs through the gates we see that this circuit realizes the boolean function

$$f(x_1, x_2, x_3) = x_1 \cdot \overline{x_2} \cdot ((x_1 + x_2) + (x_1 + x_3)).$$

We simplify the boolean expression that defines f, simplifying the circuit in so doing. You should be able to identify the laws of Boolean algebra that are used in each of the steps. See Exercise 1.1.1.

$$\begin{aligned} x_1 \cdot \overline{x_2} \cdot ((x_1 + x_2) + (x_1 + x_3)) &= x_1 \cdot \overline{x_2} \cdot (x_1 + x_2 + x_3) \\ &= x_1 \cdot \overline{x_2} \cdot x_1 + x_1 \cdot \overline{x_2} \cdot x_2 + x_1 \cdot \overline{x_2} \cdot x_3 \\ &= x_1 \cdot \overline{x_2} + 0 \cdot x_1 + x_3 \cdot x_1 \cdot \overline{x_2} \\ &= x_1 \cdot \overline{x_2} + x_3 \cdot x_1 \cdot \overline{x_2} \\ &= x_1 \cdot \overline{x_2} \cdot (1 + x_3) \\ &= x_1 \cdot \overline{x_2} \end{aligned}$$

Therefore, $f(x_1, x_2, x_3) = x_1 \cdot \overline{x_2}$, which can be realized with the much simpler circuit in Figure 1.1.15, without using the input x_3.

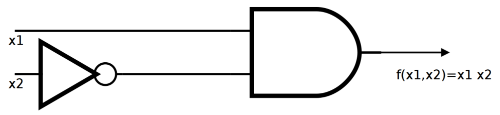

Figure 1.1.15: Simplified gate diagram

□

Next, we start with a table of desired outputs based on three bits of input and design an efficient circuit to realize this output.

Example 1.1.16 Consider the following table of desired outputs for the three input bits x_1, x_2, x_3.

x_1	x_2	x_3	$f(x_1, x_2, x_3)$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	0
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	0

Table 1.1.17: Desired output table

The first step is to write the minterm normal form of f. Since we are working with the two value Boolean algebra, B_2, the constants in each minterm are either 0 or 1, and we simply list the minterms that have a 1. These correspond with the rows of the table above that have an output of 1. We will then attempt to simplify the expression as much as possible.

$$\begin{aligned} f(x_1, x_2, x_3) &= (\overline{x_1} \cdot \overline{x_2} \cdot x_3) + (x_1 \cdot \overline{x_2} \cdot \overline{x_3}) + (x_1 \cdot \overline{x_2} \cdot x_3) \\ &= \overline{x_2} \cdot (\overline{x_1} \cdot x_3) + (x_1 \cdot \overline{x_3}) + (x_1 \cdot x_3) \\ &= \overline{x_2} \cdot (\overline{x_1} \cdot x_3) + x_1 \cdot (\overline{x_3} + x_3) \\ &= \overline{x_2} \cdot ((\overline{x_1} \cdot x_3) + x_1) \end{aligned}$$

Therefore we can realize our table with the boolean function $f(x_1, x_2, x_3) = \overline{x_2} \cdot ((\overline{x_1} \cdot x_3) + x_1)$. A circuit diagram for this function is Figure 1.1.18. But is this the simplest circuit that realizes the table? See Exercise 1.1.3.

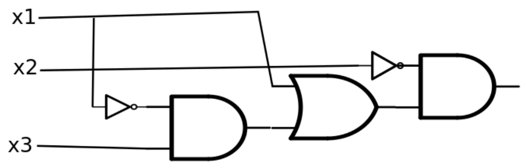

Figure 1.1.18: A realization of the table of desired outputs.

□

Exercises

1. List the laws of boolean algebra that justify the steps in the simplification of the boolean function $f(x_1, x_2, x_3)$ in Example 1.1.13. Some steps use more than one law.

Answer.

(1) Associative, commutative, and idempotent laws.

(2) Distributive law.

(3) Idempotent and complement laws.

(4) Null and identity laws

(5) Distributive law.

(6) Null and identity laws.

2. Write the following Boolean expression in the notation of logic design.

$$(x_1 \wedge \overline{x_2}) \vee (x_1 \wedge x_2) \vee (\overline{x_1} \wedge x_2).$$

Answer.

$$(x_1 \cdot \overline{x_2}) + (x_1 \cdot x_2) + (\overline{x_1} \cdot x_2).$$

3. Find a further simplification of the boolean function in Example 1.1.16, and draw the corresponding gate diagram for the circuit that it realizes.

Answer. A simpler boolean expression for the function is $\overline{x_2} \cdot (x_1 + x_3)$.

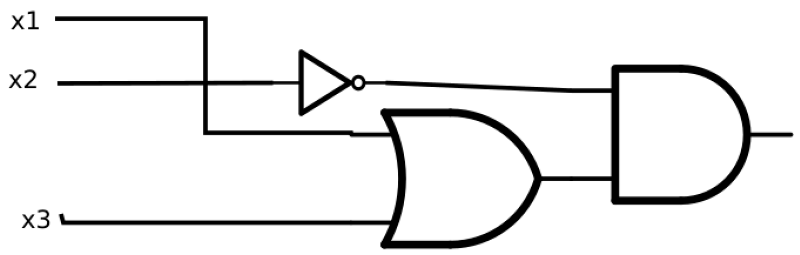

Figure 1.1.19: An even simpler circuit

4. Consider the switching circuit in Figure 1.1.20.

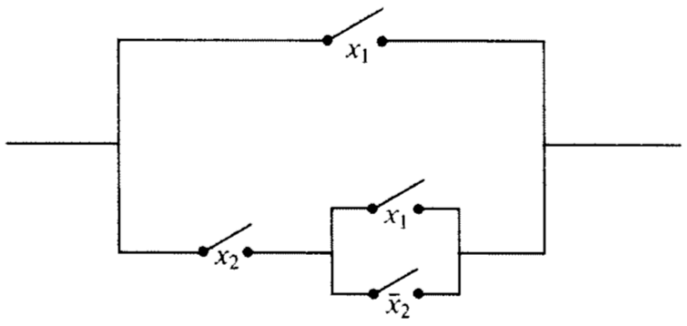

Figure 1.1.20: Can this circuit be simplifed?

(a) Draw the corresponding gate diagram for this circuit.

(b) Construct a table of outputs for each of the eight inputs to this circuit.

(c) Determine the minterm normal of the Boolean function based on the table.

(d) Simplify the circuit as much as possible.

5. Consider the circuit in Figure 1.1.21.

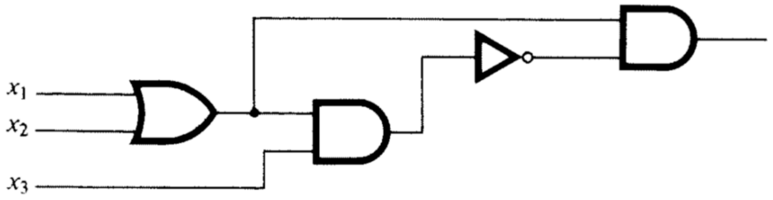

Figure 1.1.21: Can this circuit be simplifed?

(a) Trace the inputs though this circuit and determine the Boolean function that it realizes.

(b) Construct a table of outputs for each of the eight inputs to this circuit.

(c) Find the minterm normal form of f.

(d) Draw the circuit based on the minterm normal form.

(e) Simplify the circuit algebraically and draw the resulting circuit.

6. Consider the Boolean function $f\left(x_1, x_2, x_3, x_4\right) = x_1 + \left(x_2 \cdot \left(\overline{x_1} + x_4\right) + x_3 \cdot \left(\overline{x_2} + \overline{x_4}\right)\right)$.

(a) Simplify f algebraically.

(b) Draw the gate diagram based on the simplified version of f.

7. Draw a logic circuit using only AND, OR and NOT gates that realizes an XOR gate.