| | |
|---|---|
| Author | Alan Clements |
| Year | 2006 |
| Title of Article/Chapter | 2.2 Fundamental Gates |
| Title of Journal/Book | Principles of Computer Hardware |
| Vol/part/pages | 28-31 |
| Publisher | Oxford University Press |




ISBN/ISSN: 9780199273133

hole? Such a system would require extremely precise electronics.

A single binary digit is known as a bit (BInary digiT) and is the smallest unit of information possible; that is, a bit can't be subdivided into smaller units. Ideally, if a computer runs off, say, 3 V, a low level would be represented by 0.0 V and a high level by 3.0 V.

## 2.2 Fundamental gates

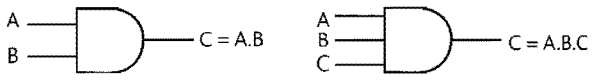
The digital computer consists of nothing more than the interconnection of three types of primitive elements called AND, OR, and NOT gates. Other gates called NAND, NOR, and EOR gates can be derived from these gates. We shall see that all digital circuits, may be designed from the appropriate interconnection of NAND (or NOR) gates alone. In other words, the most complex digital computer can be reduced to a mass of NAND gates. This statement doesn't devalue the computer any more than saying that the human brain is just a lot of neurons joined in a particularly complex way devalues the brain.

We don't use gates to build computers because we like them or because Boolean algebra is great fun. We use gates because they provide a way of mass producing cheap and reliable digital computers.

### 2.2.1 The AND gate

The AND gate is a circuit with two or more inputs and a single output. The output of an AND gate is true if and only if each of its inputs is also in a true state. Conversely, if one or more of the inputs to the AND gate is false, the output will also be false. Figure 2.3 provides the circuit symbol for both a two-input AND gate and a three-input AND gate. Note that the shape of the gate indicates its AND function (this will become clearer when we introduce the OR gate).

An AND gate is visualized in terms of an electric circuit or a highway as illustrated in Fig. 2.4. Electric current (or traffic) flows along the circuit (road) only if switches (bridges) *A* and *B* are closed. The logical symbol for the AND operator is a dot, so that A AND B can be written A · B. As in normal algebra, the dot is often omitted and A · B can be written AB. The logical AND operator behaves like the multiplier operator in conventional algebra; for example, the expression (A + B)·(C + D) = A·C + A·D + B·C + B·D in both Boolean and conventional algebra.



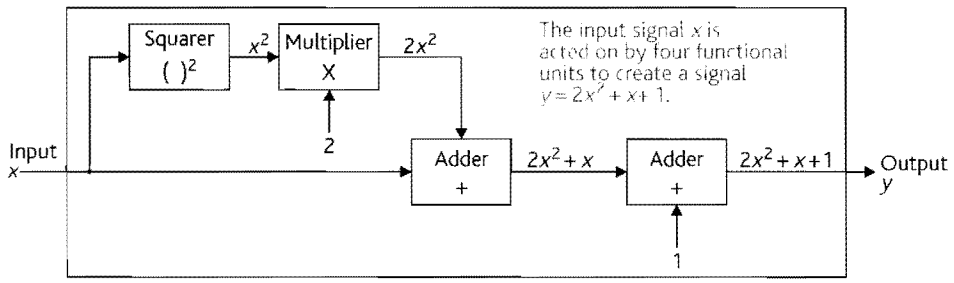

(a) Two-input AND gate (b) Three-input AND gate

Figure 2.3 The AND gate.

WHAT IS A GATE?

The word gate conveys the idea of a two-state device—open or shut. A gate may be thought of as a black box with one or more input terminals and an output terminal. The gate processes the digital signals at its input terminals to produce a digital signal at its output terminal. The particular type of the gate determines the actual processing involved. The output C of a gate with two input terminals *A* and *B* can be expressed in conventional algebra as $C = F(A,B)$, where *A*, *B*, and *C* are two-valued variables and *F* is a logical function.

The output of a gate is a function only of its inputs. When we introduce the sequential circuit, we will discover that the sequential circuit's output depends on its previous output as well as its current inputs. We can demonstrate the concept of a gate by means of an example from the analog world. Consider the algebraic expression $y = F(x) = 2x^2 + x + 1$. If we think of *x* as the input to a black box and y its output, the block diagram demonstrates how y is generated by a sequence of operations on *x*. The operations performed on the input are those of addition, multiplication, and squaring. Variable *x* enters the 'squarer' and comes out as $x^2$. The output from the squarer enters a multiplier (along with the constant 2) and comes out as $2x^2$, and so on. By applying all the operations to input *x*, we end up with output $2x^2 + x + 1$. The boxes carrying out these operations are entirely analogous to gates in the digital world—except that gates don't do anything as complicated as addition or multiplication.

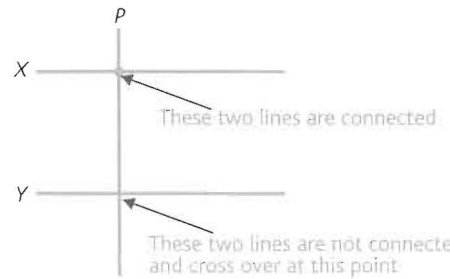


## CIRCUIT CONVENTIONS

Because we write from left to right, many logic circuits are also read from left to right; that is, information flows from left to right with the inputs of gates on the left and the outputs on the right.

Because a circuit often contains many signal paths, some of these paths may have to cross over each other when the diagram is drawn on two-dimensional paper. We need a means of distinguishing between wires that join and wires that simply cross each other (rather like highways that merge and highways that fly over each other). The standard procedure is to regard two lines that simply cross as not being connected as the diagram illustrates. The connection of two lines is denoted by a dot at their intersection.

The voltage at any point along a conductor is constant and therefore the logical state is the same everywhere on the line. If a line is connected to the input of several gates, the input to each gate is the same. In this diagram, the value of *X* and *P* must be the same because the two lines are connected.




A corollary of the statement that the same logic state exists everywhere on a conductor is that a line must not be connected to the output of more than one circuit—otherwise the state of the line will be undefined if the outputs differ. At the end of this chapter we will introduce gates with special tri-state outputs that can be connected together without causing havoc.

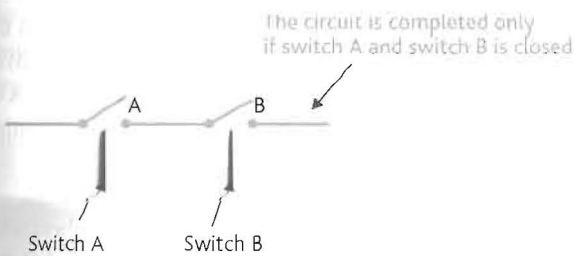



Figure 2.4 The representation of an AND gate.

A useful way of describing the relationship between the inputs of a gate and its output is the truth table. In a truth table the value of each output is tabulated for every possible combination of the inputs. Because the inputs are two valued (i.e. binary with states 0 and 1), a circuit with $n$ inputs has $2^n$ lines in its truth table. The order in which the $2^n$ possible inputs are taken is not important but by convention the order corresponds to the natural binary sequence (we discuss binary numbers in Chapter 4). Table 2.1 describes the natural binary sequences for values of $n$ from 1 to 4.

| $n = 1$ | $n = 2$ | $n = 3$ | $n = 4$ |
|---|---|---|---|
| 0 | 00 | 000 | 0000 |
| 1 | 01 | 001 | 0001 |
| | ... | ... | ... |
| | 11 | 011 | 0011 |
| | | 100 | 0100 |
| | | 101 | 0101 |
| | | 110 | 0110 |
| | | 111 | 0111 |
| | | | 1000 |
| | | | 1001 |
| | | | 1010 |
| | | | 1011 |
| | | | 1100 |
| | | | 1101 |
| | | | 1110 |
| | | | 1111 |

Table 2.1 The $2^n$ possible values of an $n$-bit variable for $n = 1$ to 4.

Table 2.2 illustrates the truth table for a two-input AND gate, although there's no reason why we can't have any number of inputs to an AND gate. Some real gates have three or four inputs and some have 10 or more inputs. However, it doesn't matter how many inputs an AND gate has. Only one line in the truth table will contain a 1 entry because all inputs must be true for the output to be true.

| Inputs A | Inputs B | Output F = A · B | |
|---|---|---|---|
| 0 | 0 | 0 | False because one or more inputs is false |
| 0 | 1 | 0 | |
| 1 | 0 | 0 | |
| 1 | 1 | 1 | True because both inputs are true |

Table 2.2 Truth table for the AND gate.

When we introduce computer arithmetic, computer architecture, and assembly language programming, we will see that computers don't operate on bits in isolation. Computers process entire groups of bits at a time. These groups are called words and are typically 8, 16, 32, or 64 bits wide. The AND

operation, when applied to words, is called a logical operation to distinguish it from an arithmetic operation such as addition, subtraction, or multiplication. When two words take part in a logical operation such as an AND, the operation takes place between the individual pairs of bits; for example bit $a_i$ of word A is ANDed with bit $b_i$ of word B to produce bit $c_i$ of word C. Consider the effect of ANDing of the following two 8-bit words A = 11011100 and B = 01100101.x

```
1 1 0 1 1 1 0 0 ← word A
0 1 1 0 0 1 0 1 ← word B
0 1 0 0 0 1 0 0 ← C = A·B
```

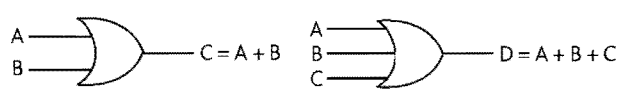
In this example the result C = A · B is given by 01000100. Why should anyone want to AND together two words? If you AND bit $x$ with 1, the result is $x$ (because Table 2.2 demonstrates that 1.0 = 0 and 1.1 = 1). If you AND bit $x$ with 0 the result is 0 (because the output of an AND gate is true only if both inputs are true). Consequently, a logical AND is used to mask certain bits in a word by forcing them to zero. For example, if we wish to clear the leftmost four bits of an 8-bit word to zero, ANDing the word with 00001111 will do the trick. The following example demonstrates the effect of an AND operation with a 00001111 mask.

```
1 1 0 1 1 0 1 1 ← source word
0 0 0 0 1 1 1 1 ← mask
0 0 0 0 1 0 1 1 ← result
```

## 2.2.2 The OR gate

The output of an OR gate is true if any one (or more than one) of its inputs is true. Notice the difference between AND and OR operations. The output of an AND is true only if *all* inputs are true whereas the output of an OR is true if at least one input is true. The circuit symbol for a two-input and a three-input OR gate is given in Fig. 2.5. The logical symbol for an OR operation is an addition sign, so that the logical operation A OR B is written as A + B. The logical OR operator is the same as the conventional addition symbol because the OR operator behaves like the addition operator in algebra (the reasons for this will become clear when we introduce Boolean algebra). Table 2.3 provides the truth table for a two-input OR gate.

The behavior of an OR gate can be represented by the switching circuit of Fig. 2.6. A path exists from input to output if either of the two switches is closed.




(a) Two-input OR gate. (b) Three-input OR gate.

Figure 2.5 The OR gate.

| Inputs A | B | Output F=A+B |
|---|---|---|
| 0 | 0 | 0 |
| 0 | 1 | 1 |
| 1 | 0 | 1 |
| 1 | 1 | 1 |

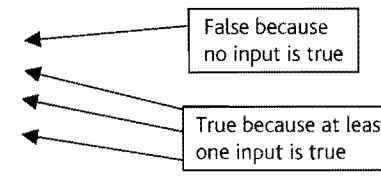



Table 2.3 Truth table for the OR gate.

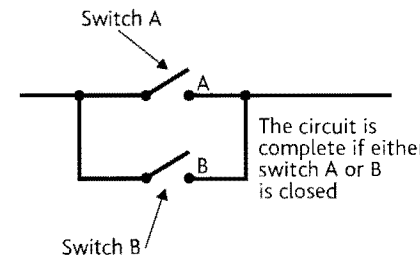



Figure 2.6 The representation of an OR gate.

The use of the term *OR* here is rather different from the English usage of *or*. The Boolean OR means (either A or B) or (both A and B), whereas the English usage often means A or B but not (A and B). For example, consider the contrasting use of the word or in the two phrases: 'Would you like tea or coffee?' and 'Reduced fees are charged to members who are registered students or under 25'. We shall see that the more common English use of the word or corresponds to the Boolean function known as the EXCLUSIVE OR, an important function that is frequently abbreviated to EOR or XOR.

A computer can also perform a logical OR on words as the following example illustrates.

```
1 0 0 1 1 1 0 0 ← word A
0 0 1 0 0 1 0 1 ← word B
1 0 1 1 1 1 0 1 ← C = A + B
```

The logical OR operation is used to set one or more bits in a word to a logical 1. The term set means make a logical one, just as clear means reset to a logical zero. For example, the least-significant bit of a word is set by ORing it with 00 . . . 01. By applying both AND and OR operations to a word we can selectively clear or set its bits. Suppose we have an 8-bit binary word and we wish to clear bits 6 and 7 and set bits 4 and 5. If the bits of the word are $d_0$ to $d_7$, we can write:

| $d_7$ | $d_6$ | $d_5$ | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ | |
|---|---|---|---|---|---|---|---|---|
| 0 | 0 | 1 | 1 | 1 | 1 | 1 | 1 | Source word<br>AND mask |
| 0 | 0 | $d_5$ | $d_4$ | $d_3$ | $d_2$ | $d_1$ | $d_0$ | First result |
| 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | OR mask |
| 0 | 0 | 1 | 1 | $d_3$ | $d_2$ | $d_1$ | $d_0$ | Final result |

## 2.2.3 The NOT gate

The NOT gate is also called an inverter or a complementer and is a two-terminal device with a single input and a single output. If the input of an inverter is X, its output is NOT X which is written $\overline{X}$ or X'. Figure 2.7 illustrates the symbol for an inverter and Table 2.4 provides its truth table. Some teachers vocalize $\overline{X}$ as 'not X' and others as 'X not'. The inverter is the simplest of gates because the output is the opposite of the input. If the input is 1 the output is 0 and vice versa. By the way, the triangle in Fig. 2.7 doesn't represent an inverter. The small circle at the output of the inverter indicates the inversion operation. We shall see that this circle indicates logical inversion wherever it appears in a circuit.

We can visualize the operation of the NOT gate in terms of the relay illustrated in Fig. 2.8. A relay is an electromechanical switch (i.e. a device that is partially electronic and partially mechanical) consisting of an iron core around which a coil of wire is wrapped. When a current flows through a coil, it generates a magnetic field that causes the iron core to act as a magnet. Situated close to the iron core is a pair of contacts, the lower of which is mounted on a springy strip of iron. If switch A is open, no current flows through the coil and the iron core remains unmagnetized. The relay's contacts are normally closed so that they form a switch that is closed when switch A is open.

If switch A is closed, a current flows through the coil to generate a magnetic field that magnetizes the iron core. The contact on the iron strip is pulled toward the core, opening the contacts and breaking the circuit. In other words, closing switch A opens the relay's switch and vice versa. The system in Fig. 2.8 behaves like a NOT gate. The relay is used by a computer to control external devices and is described further when we deal with input and output devices.

Like both the AND and OR operations, the NOT function can also be applied to words:

```
1 1 0 1 1 1 0 0  ← word A
0 0 1 0 0 0 1 1  ← B = Ā
```

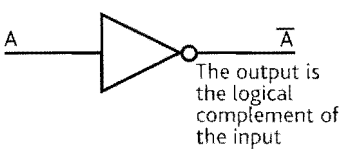



**Figure 2.7** The NOT gate or inverter.

| Input A | Output F=Ā |
|---|---|
| 0 | 1 |
| 1 | 0 |

**Table 2.4** Truth table for the inverter.

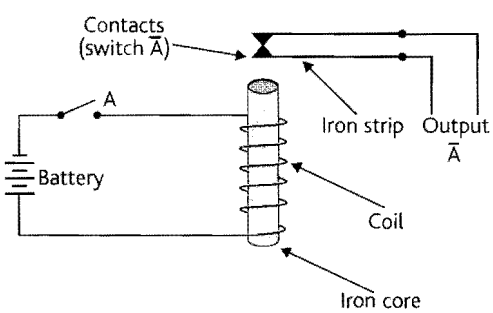



**Figure 2.8** The operation of a relay.

## 2.2.4 The NAND and NOR gates

The two most widely used gates in real circuits are the NAND and NOR gates. These aren't fundamental gates because the NAND gate is derived from an AND gate followed by an inverter (Not AND) and the NOR gate is derived from an OR gate followed by an inverter (Not OR), respectively. The circuit symbols for the NAND and NOR gates are given in Fig. 2.9. The little circle at the output of a NAND gate represents the symbol for inversion or complementation. It is this circle that converts the AND gate to a NAND gate and an OR gate to a NOR gate. Later, when we introduce the concept of mixed logic, we will discover that this circle can be applied to the inputs of gates as well as to their outputs.

Table 2.5 gives the truth table for the NAND and the NOR gates. As you can see, the output columns in the NAND and NOR tables are just the complements of the outputs in the corresponding AND and OR tables.

We can get a better feeling for the effect that different gates have on two inputs, A and B, by putting all the gates together in a single table (Table 2.6). We have also included the EXCLUSIVE OR (i.e. EOR) and its complement the EXCLUSIVE NOR (i.e. EXNOR) in Table 2.6 for reference. The EOR gate is derived from AND, OR, and NOT gates and is described in more detail later in this chapter. It should be noted here that $\overline{A \cdot B}$ is not the same as $\overline{A} \cdot \overline{B}$, just as $\overline{A+B}$ is not the same as $\overline{A}+\overline{B}$.

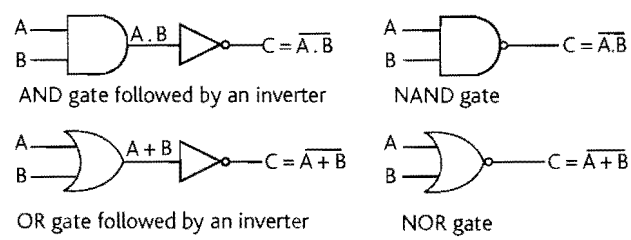



**Figure 2.9** Circuit symbols for the NAND and NOR gates.