# Understanding Negative Logic

## *Taking the mystery out of a logic type that can greatly simplify circuit design and analysis*

By Delton T. Horn

With the proliferation of digital circuits in most modern electronic and many electrical products, electronics hobbyists and technicians usually have at least some basic understanding of digital circuitry. Though you might be familiar with positive logic—the kind in popular use in modern digital circuits—you may not understand and even be intimidated by its counterpart, *negative* logic. This is unfortunate because negative logic can greatly simplify design and analysis of many digital circuits. Fortuitously, negative logic is also really no more difficult to grasp than is positive logic.

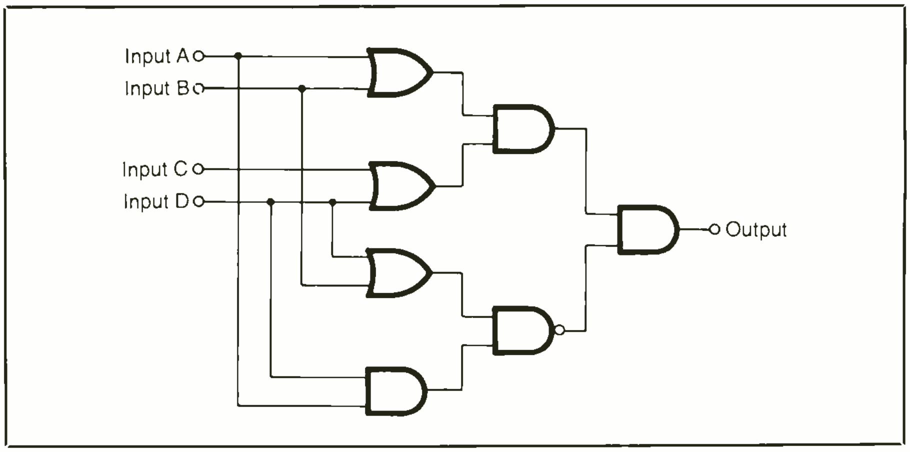


*Fig. 1. An example of a multiple-gate positive-logic circuit.*

### Some Basics

In positive logic, a logic 1 is represented by a more-positive signal voltage than a negative logic signal. Negative logic reverses this relationship: the more-positive signal level represents a logic 0 and a near-ground signal level is a logic 1.

In terms of functionality, no difference exists between the two logic types. They are simply two different ways of viewing the same action performed by digital gates. For example, consider the truth table for a 2-input AND gate that uses positive logic:

| Input A B | Output |
|---|---|
| 0 0 | 0 |
| 0 1 | 0 |
| 1 0 | 0 |
| 1 1 | 1 |

Each 1 in the table indicates a high signal voltage, each 0 a low signal voltage. The truth table for the same device using negative logic is:

| Inputs A B | Output |
|---|---|
| 1 1 | 1 |
| 1 0 | 1 |
| 0 1 | 1 |
| 0 0 | 0 |

This time, each 1 represents a low signal voltage, each 0 a high signal voltage. As you can see, each input and output signal level has reversed its normal (positive) state, even though the actual voltages in the circuit remain exactly the same. The only thing that has changed is the names used to identify each of the possible input and output conditions.

Taking a close look at the negative-logic truth table for the AND gate, you'll notice something very interesting. The output from the gate is a logic 0 if and only if both inputs are at logic 0. Stated differently, if either input A or input B or both are a logic 1, the output will be a logic 1. A circuit element that behaves as an AND gate in positive logic functions as an OR gate in negative logic. It is important to keep in mind that no change has been made in the circuitry, which is operating in exactly the same manner in both cases. The only thing that has been altered is your way of interpreting the action of the circuit.

You can compare the positive- and negative-logic functions of each of the basic digital gates as follows:

| Positive Logic | Negative Logic |
|---|---|
| AND | OR |
| NAND | NOR |
| OR | AND |
| NOR | NAND |

Noticeably absent from the above are the inverter and buffer. The reason

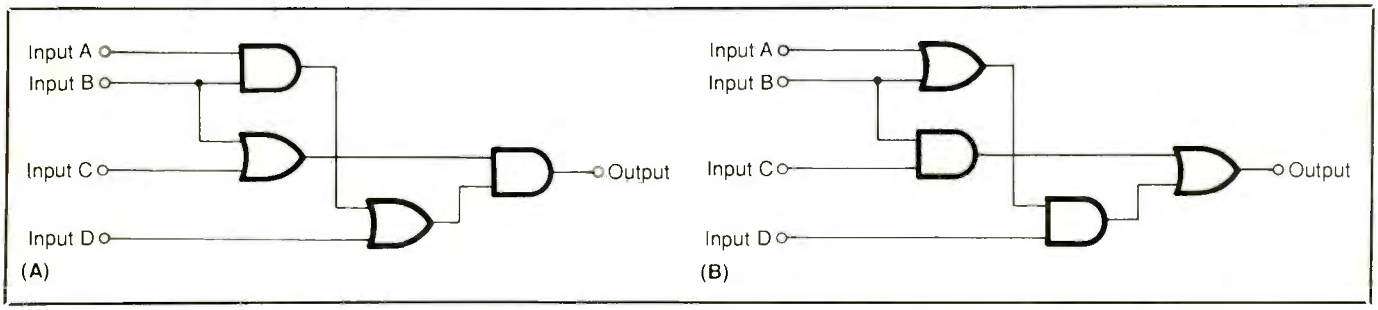


*Fig. 2. Shown here are (A) a positive- logic circuit and (B) the equivalent negative-logic circuit.*

for this is that both logic elements function in the same manner in both logic systems.

## *Why Negative Logic?*

Often in the course of designing digital electronic circuits, you will encounter a truth table that appears to be quite complex and requires a large number of gates to implement. Consider for example the following truth table:

| Inputs A B C D | Output |
|---|---|
| 0 0 0 0 | 0 |
| 0 0 0 1 | 0 |
| 0 0 1 0 | 0 |
| 0 0 1 1 | 0 |
| 0 1 0 0 | 0 |
| 0 1 0 1 | 1 |
| 0 1 1 0 | 1 |
| 0 1 1 1 | 1 |
| 1 0 0 0 | 0 |
| 1 0 0 1 | 1 |
| 1 0 1 0 | 0 |
| 1 0 1 1 | 1 |
| 1 1 0 0 | 0 |
| 1 1 0 1 | 1 |
| 1 1 1 0 | 1 |
| 1 1 1 1 | 1 |

This would appear to be a very easy truth table pattern to generate—at least not if you stick to positive logic. You might come up with the seven-gate circuit shown in Fig. 1. This circuit will work, but it's rather awkward to implement in hardware.

Now let's try converting the above truth table into negative logic. For easier readability, we'll reverse the order of the table entries so that the inputs still run from 0000 to 111, as follows:

| Inputs A B C D | Output |
|---|---|
| 0 0 0 0 | 0 |
| 0 0 0 1 | 0 |
| 0 0 1 0 | 0 |
| 0 0 1 1 | 1 |
| 0 1 0 0 | 0 |
| 0 1 0 1 | 1 |
| 0 1 1 0 | 0 |
| 0 1 1 1 | 1 |
| 1 0 0 0 | 0 |
| 1 0 0 1 | 01 |
| 1 0 1 0 | 0 |
| 1 0 1 1 | 1 |
| 1 1 0 0 | 1 |
| 1 1 0 1 | 1 |
| 1 1 1 0 | 1 |
| 1 1 1 1 | 1 |

A simpler solution should now become more obvious. This negative-logic truth table can be generated with the four-gate circuit shown in Fig. 2. The circuits shown in these two schematics are functionally identical. They generate the same truth table in either positive or negative logic. However, you might not have thought of the Fig. 2(B) circuit, which is more efficient that the Fig. 2(A) circuit, if negative logic had not been used.

Obviously, it is possible to arrive at the Fig. 2(B) circuit, even using just the positive-logic truth table. Using the negative-logic truth table simply makes the solution easier to arrive at.

Some circuit designs will be easier to implement with positive logic, others with negative logic. In many cases, it won't make a difference which type of logic is used. It is even okay to mix both positive and negative logic throughout a circuit design of a single system, as long as you manage to keep the two separate. Don't confuse the two types of logic or you won't obtain the results you want. ME