an error and must be avoided. The actual voltage on a node with contention may be somewhere between 0 and V_{DD}, depending on the relative strengths of the gates driving HIGH and LOW. It is often, but not always, in the forbidden zone. Contention also can cause large amounts of power to flow between the fighting gates, resulting in the circuit getting hot and possibly damaged.

X values are also sometimes used by circuit simulators to indicate an uninitialized value. For example, if you forget to specify the value of an input, the simulator may assume it is an X to warn you of the problem.

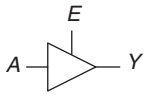
As mentioned in Section 2.4, digital designers also use the symbol X to indicate "don't care" values in truth tables. Be sure not to mix up the two meanings. When X appears in a truth table, it indicates that the value of the variable in the truth table is unimportant (can be either 0 or 1). When X appears in a circuit, it means that the circuit node has an unknown or illegal value.

2.6.2 Floating Value: Z

The symbol Z indicates that a node is being driven neither HIGH nor LOW. The node is said to be *floating*, *high impedance*, or *high Z*. A typical misconception is that a floating or undriven node is the same as a logic 0. In reality, a floating node might be 0, might be 1, or might be at some voltage in between, depending on the history of the system. A floating node does not always mean there is an error in the circuit, so long as some other circuit element does drive the node to a valid logic level when the value of the node is relevant to circuit operation.

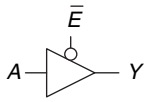
One common way to produce a floating node is to forget to connect a voltage to a circuit input, or to assume that an unconnected input is the same as an input with the value of 0. This mistake may cause the circuit to behave erratically as the floating input randomly changes from 0 to 1. Indeed, touching the circuit may be enough to trigger the change by means of static electricity from the body. We have seen circuits that operate correctly only as long as the student keeps a finger pressed on a chip.

Tristate Buffer



E	A	Y
0	0	Z
0	1	Z
1	0	0
1	1	1

Figure 2.40 Tristate buffer



$\bar{E}$	A	Y
0	0	0
0	1	1
1	0	Z
1	1	Z

Figure 2.41 Tristate buffer with active low enable

The *tristate buffer*, shown in Figure 2.40, has three possible output states: HIGH (1), LOW (0), and floating (Z). The tristate buffer has an input A, output Y, and *enable* E. When the enable is TRUE, the tristate buffer acts as a simple buffer, transferring the input value to the output. When the enable is FALSE, the output is allowed to float (Z).

The tristate buffer in Figure 2.40 has an *active high* enable. That is, when the enable is HIGH (1), the buffer is enabled. Figure 2.41 shows a tristate buffer with an *active low* enable. When the enable is LOW (0),

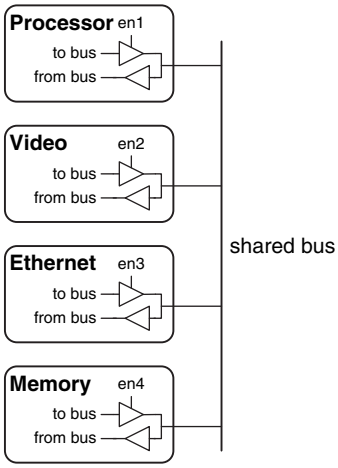


Figure 2.42 Tristate bus connecting multiple chips

the buffer is enabled. We show that the signal is active low by putting a bubble on its input wire. We often indicate an active low input by drawing a bar over its name, $\overline{E}$, or appending the letters "b" or "bar" after its name, *Eb* or *Ebar*.

Tristate buffers are commonly used on *busses* that connect multiple chips. For example, a microprocessor, a video controller, and an Ethernet controller might all need to communicate with the memory system in a personal computer. Each chip can connect to a shared memory bus using tristate buffers, as shown in Figure 2.42. Only one chip at a time is allowed to assert its enable signal to drive a value onto the bus. The other chips must produce floating outputs so that they do not cause contention with the chip talking to the memory. Any chip can read the information from the shared bus at any time. Such tristate busses were once common. However, in modern computers, higher speeds are possible with *point-to-point links*, in which chips are connected to each other directly rather than over a shared bus.

2.7 KARNAUGH MAPS

After working through several minimizations of Boolean equations using Boolean algebra, you will realize that, if you're not careful, you sometimes end up with a completely *different* equation instead of a simplified equation. *Karnaugh maps* (*K-maps*) are a graphical method for simplifying Boolean equations. They were invented in 1953 by Maurice Karnaugh, a telecommunications engineer at Bell Labs. K-maps work well for problems

Maurice Karnaugh, 1924–. Graduated with a bachelor's degree in physics from the City College of New York in 1948 and earned a Ph.D. in physics from Yale in 1952.

Worked at Bell Labs and IBM from 1952 to 1993 and as a computer science professor at the Polytechnic University of New York from 1980 to 1999.