We can understand the operation of the OR gate by observing the behavior of its six transistors. If both inputs *A* and *B* are low, both T1 and T2 will be active. Furthermore, if *A* and *B* are both low, T3 and T4 will be off. In this case, the signal labeled **~(A|B)** will be high because the T1–T2 switch combination will short this signal to +3.3V. If *A* is high, T3 will be active and T1 off. Similarly, if *B* is high, T4 will be active and T2 off. Therefore if either *A* is high or if *B* is high, the signal labeled **~(A|B)** will be low because one or both of the T3, T4 switches will short this signal to ground. Transistors T5 and T6 create a logical complement, converting the signal *~(A|B)* into the desired result of *A|B*. We use the OR operation to set individual bits.

When writing software we will have two kinds of logic operations. When operating on numbers (collection of bits) we will perform **logic** operations bit by bit. In other words, the operation is applied independently on each bit. In C, the logic operator for AND is **&**. For example, if number *A* is 01100111 and number *B* is 11110000 then

Help

A = 01100111

B = 11110000

A&B 01100000

The other type of logic operation occurs when operating on **Boolean** values. In C, the condition false is represented by the value 0, and true is any nonzero value. In this case, if the Boolean *A* is 01100111 and *B* is 11110000 then both *A* and *B* are true. The standard value for true is the value 1. In C, the Boolean operator for AND is **&&**. Performing Boolean operation yields

A = 01100111

B = 11110000

A&&B 1

In C, the logic operator for OR is **|**. The logic operation is applied independently on each bit**.** E.g.,

A = 01100111

B = 11110000

*A***|***B* 11110111

In C, the Boolean operator for OR is **||**. Performing Boolean operation of **true** OR **true** yields **true**. Although 1 is the standard value for a true, any nonzero value is considered as true.

A =	01100111
B =	11110000
A || *B*	1

Other convenient logical operators are shown as digital gates in Figure 4.6. The **NAND** operation is defined by an AND followed by a NOT. If you compare the transistor-level circuits in Figures 4.5 and 4.6, it would be more precise to say AND is defined as a NAND followed by a NOT. Similarly, the OR operation is a **NOR** followed by a NOT. The **exclusive NOR** operation implements the bit-wise equals operation.

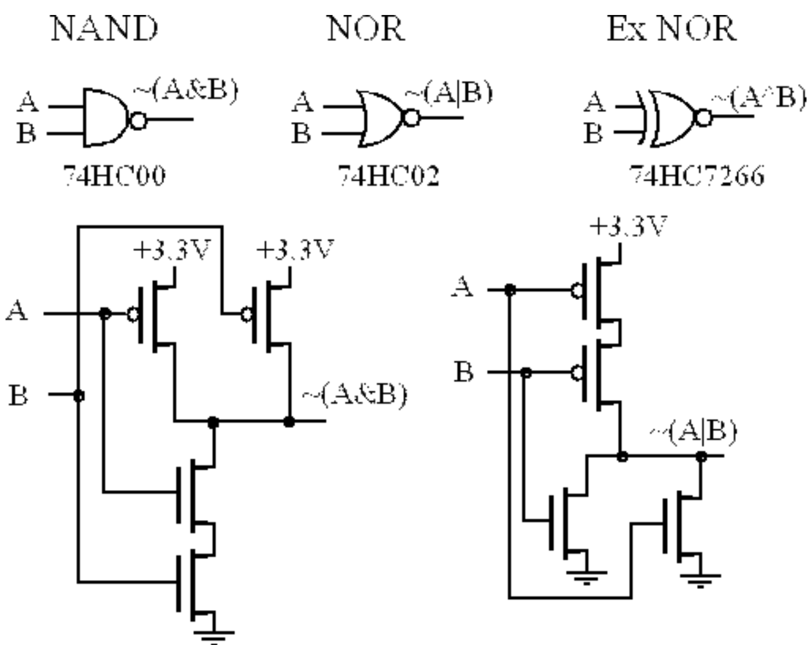

Figure 4.6. Other logical operations can also be implemented with MOS transistors.

DESIGN OF A NAND GATE

JONATHAN VALVANO Next we're going to show you how to build a NAND gate.

A NAND gate has two inputs, A and B, and one output, such that, if A and B

are true, then the output is false.

Conversely, if either A or B are 0, then the

	0:00 / 4:40	1.0x				

output will be true.

OK, that's a NAND gate.

We'll begin with a NOT gate that we saw from the last video.

If this signal is A and this signal is OUT, if A is high, OUT will be low.

And if A is low, OUT will be high, a NOT gate.

And we see from last time, the way to create a +3 output is, on the P

channel, we generate a positive voltage across the

source and the gate.

So if we need to create a 3.3 for the B, we're going to create another

transistor right here, another P channel, tie it to 3.3 volts, tie this

output there, and connect my B signal to the gate here.

Help