Chapter 2: Introduction to Digital Logic

Objectives of Chapter 2

- Learn the electronic circuit basis for digital logic gates
- Understand how modern CMOS logic works
- Become familiar with the basics of logic gates

Remember the simple battery and flashlight circuit we saw in figure 1.14? The two on/off switches, wired as they were in series, implemented the logical **AND** function. We can express that example as, "If switch A is closed **AND** switch B is closed, **THEN** lamp C will be illuminated." Admittedly, this is pretty far removed from your PC, but the logical function implemented by the two switches in this circuit is one of the four key elements of a modern computer.

It may surprise you to know that all of the primary digital elements of a modern computer, the central processing unit (CPU), memory and I/O can be constructed from four primary logical function: ***AND, OR, NOT*** and ***Tri-State (TS)***. Now TS is not a logical function, rather, it is actually closer to an electronic circuit implementation tool. However, without tri-state logic, modern computers would be impossible to build.

As we'll soon see, tri-state logic introduces a third logical condition called ***Hi-Z***. "Z" is the electronic symbol for impedance, a measure of the ease by which electrical current can flow in a circuit. So Hi-Z, seems to imply a lot of resistance to current flow. As you'll soon see, this is critical for building our system.

It is interesting to note that we could build a computer from the ground up using the four fundamental logic gates: AND, OR, NOT and TS. But it doesn't necessarily follow that we would build it that way. This is because engineers will usually take implementation short-cuts in designing more complex functions and these design efficiencies will tend to blur the distinctions between the fundamental logic elements. However, it does not diminish the conceptual importance of these four fundamental logical functions.

In writing this I was struck by the similarity between the DNA molecule and its four nucleotides, Adenine, Cytosine, Guanine, and Thymine, abbreviated, A, C, G and T, and the fact that a computer can also be described by four "electronic nucleotides". We shouldn't look too deeply into this coincidence because the differences certainly far outweigh the similarities. Anyway, it's fun to imagine an "electronic DNA molecule" of the future that can be used as a blueprint for replicating itself. Could there be a science-fiction novel in this somewhere? Figure 2.1 illustrates the admittedly weak analogy between DNA and computer hardware primitives.

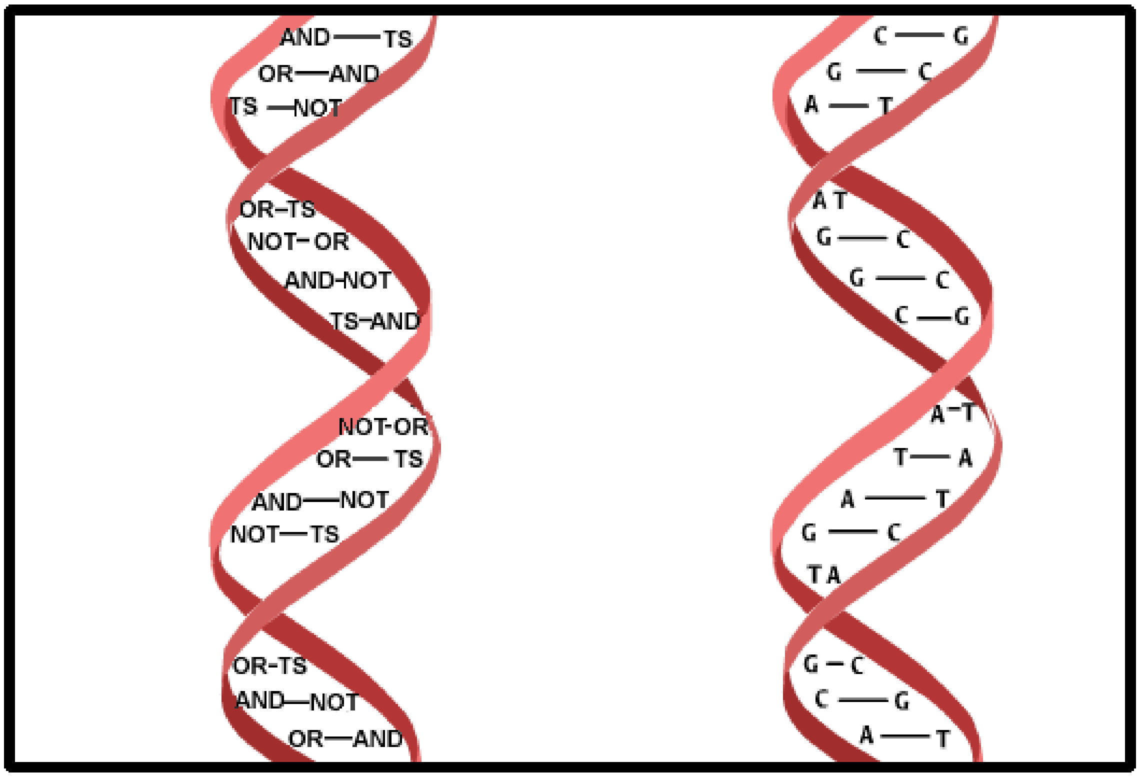


Figure 2.1: Thinking about a computer built from "logical DNA" on the left and schematic picture of a portion of a real DNA molecule on the right. *DNA molecule picture from the Dolan Learning Center, Cold Springs Harbor Laboratory*[1]*.*

Enough of the DNA analogy! Let's move on. Let me make one last point before we do leave. The key point of making the analogy is that everything that we will be doing from now on in the realm of digital hardware will be based upon these four fundamental logic functions. That may not make it any easier for you, but that's where we're headed.

Figure 2.2 is a schematic diagram of a digital logic gate which can execute the logical "**AND"** function. The symbol, represented by the label F(A,B) is the standard symbol that you would use to represent an **AND** gate in the schematic diagram for a digital circuit design. The output, C, is a function of the two binary input variables A and B. A and B are binary variables, but in this circuit they are represented by values of voltage. In the case of *positive logic*, where the more positive (higher) voltage is a "1" and the less positive (lower) voltage is a "0" this circuit element is most commonly used in circuits where a logic "0" is represented by a voltage in the range of 0 to 0.8 volts and a "1" is a voltage between approximately 3.0 and 5.0 volts[1]. From 0.8 volts to 3.0 volts is "no man's land." Signals travel through this region very quickly on their way up or down, but don't dwell there. If a logic value were to be measured in this region, there would be an electrical fault in the circuit.

The logical function of figure 1.16 can be described in several equivalent ways:

[1] Assuming that we are using the older, 5 volt logic families rather than the newer 3.3 volt families.

- **IF** A is TRUE **AND** B is TRUE **THEN** C is TRUE
- **IF** A is HIGH **AND** B is HIGH **THEN** C is HIGH
- **IF** A is 1 **AND** B is 1 **THEN** C is 1
- **IF** A is ON **AND** B is ON **THEN** C is ON
- **IF** A is 5 volts **AND** B is 5 volts **THEN** C is 5 volts

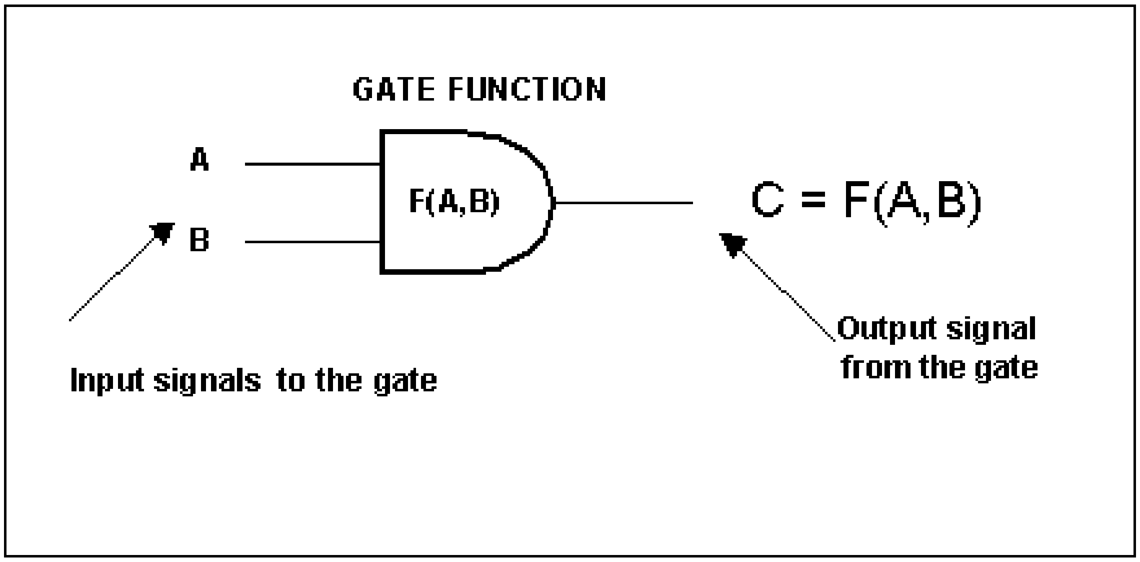


Figure 2.2: Logical AND gate. The gate itself is an electronic switching circuit that implements the logical AND function for the two input values A and B.

The last bullet is a little iffy because we allow the values to exist in ranges, rather than absolute numbers. It's just easier to say "5 volts" instead of "a range of 3 to 5 volts."

As we've discussed in the last chapter, in a real circuit, A and B are signals on individual wires. These wires may be actual wires that are used to form a circuit, or they may be extremely fine copper conducting paths (traces) on a printed circuit (PC) board, as in figure 1.11, or they may be microscopic paths on an integrated circuit. In all cases it is a single wire conducting a digital signal that may take on two digital values, "0" or "1."

The next point that we want to consider is why we call this circuit element a "gate." You can imagine a real gate in front of your house. Someone has to open the gate in order to gain entrance. The AND gate can be thought of in exactly the same way. It is a gate for the passage of the logical signal. The bulleted statements, above, describe the AND gate as a logic statement. However, we can also describe it this way:

- **IF** A is 1 **THEN** C **EQUALS** B
- **IF** A is 0 **THEN** C **EQUALS** 0

Of course, we could exchange A and B because these inputs are equivalent. We can see this graphically in figure 2.3. Notice the strange line at inputs B and output C. This is a way to represent a digital signal that is varying with time. This is called a ***waveform*** representation, or a ***timing diagram.*** The idea is that the waveform is on some kind of graph paper, such as a strip chart recorder, and time is changing on the horizontal axis. The vertical axis represents the logical value at a point in the circuit, in this case points B and C.

When A=1, the same signal appears at output C that is input at B. The change that occurs at C as a result of the change at B is not instantaneous because the speed of light is finite, and circuits are not infinitely fast. However, in terms of a human scale, it's pretty fast. In a typical circuit, if input B went from a 0 value to a 1, the same change would occur at output C, delayed by about 5 billionths of a second (5 nanoseconds).

You've actually seen these timing diagrams before in the context of an EKG (electrocardiogram) when a physician checks your heart. Each of the signals on the chart represents the electrical voltage at various parts of your heart muscles over time. Time is traveling along the long axis of the chart and the voltage at any point in time is represented by the vertical displacement of the ink. Since typical digital signals change much more rapidly than we could see on a strip chart recorder, we need specialized equipment, such as oscilloscopes and logic analyzers, to record the waveforms and display them for us in a way that we can comprehend. We'll discuss waveforms and timing diagrams in the upcoming lessons.

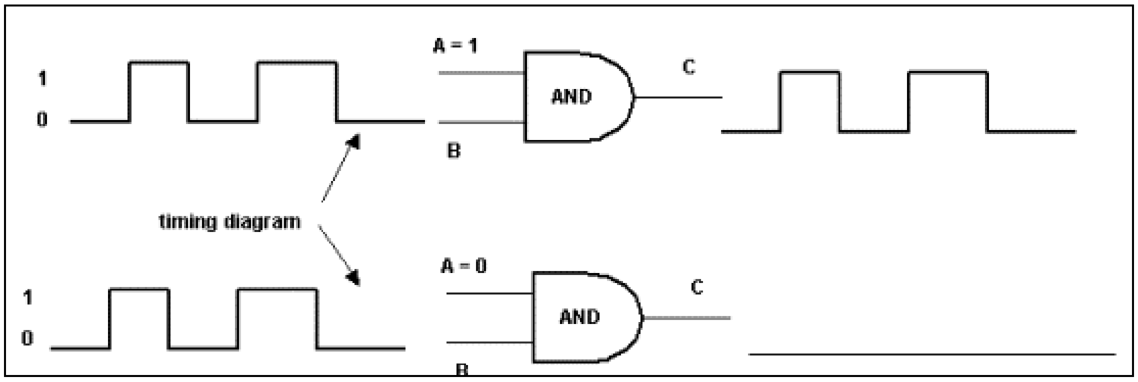


Figure 2.3: The logical AND circuit represented as a gating device. When the input A=1, output C follows input B (the gate is open). When input A=0, output C=0, independent of how input B varies with time (the gate is closed).

Thus, figure 2.3 shows us an input waveform at B. If we could get really small, and we had a very fast stopwatch and a fast voltmeter, we could imagine that we are sitting on the wire connected to the gate at point B. As the value of the voltage at B changes, we check our watch and plot the voltage versus time on the graph. That's the waveform that is shown in figure 2.3.

Also notice the vertical line representing the transition from the logic level 0 to the logic level 1. This is called the ***rising edge.*** Likewise, the vertical line that represents the transition from logic level 1 to logic level 0 is called the ***falling edge***. We typically want the time durations of the rising edge and falling edge to be very small, a few billionths of a second (nanoseconds) because in digital systems, the space between 0 and 1 is not defined, and we want to get through there as quickly as possible. That's not to say that these edges are useless to us. Quite the contrary, they are pretty important. In fact, you'll see the value of these edges when we study system clocks later on.

If, in figure 2.3, A=1 (upper figure), the output at C follows the input at A with a slight time delay, called the ***propagation delay***, because it represents the time required for the input change to work its way, or propagate, through the circuit element. When the ***control input*** at A=0, output C will always equal 0, independent of whatever input B does. Thus, input B is ***gated*** by input A. For now, we'll leave the timing diagram in the wonderful world of hardware design and return to our studies of logic gates.

Earlier, we touched on the fact that the AND gate is one of three logic gates. We'll keep the tri-state gate separate for now and study it in more depth when we discuss bus organization in a later chapter. In terms of "atomic", or unique elements, there are actually three types of logical gates, **AND, OR** and **NOT.** These are the fundamental building blocks of all the complex digital logic circuits to follow. Consider figure 2.4.

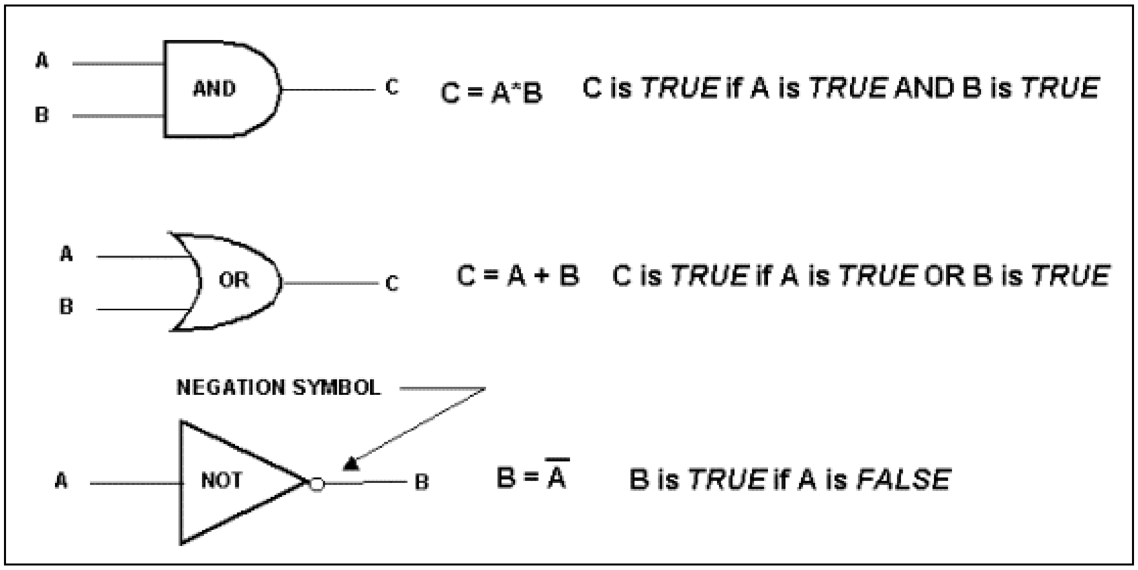


Figure 2.4: The three "atomic" logic gates: AND, OR and NOT.

The symbol for the AND function is the same as the multiplication symbol that we use in algebra. As we'll see later on, the AND operation is similar to multiplying two binary numbers, since 1x1 = 1 and 1x 0 = 0. The asterisk is a convenient symbol to use for "ANDing" two variables.

The symbol for the OR function is the plus sign, and it is "sort of" like addition, because 0 + 0 = 0, 1 + 0 = 1. The analogy fails with 1 + 1, because if we're adding, then 1 + 1 = 0 (carry the 1) but if we are ORing, then 1 + 1 = 1. OK, so they overloaded the + sign. Don't get angry with me, I'm only the messenger.

The negation symbol takes many forms, mostly because it is difficult to draw a line over a variable using only ASCII text characters. In figure 2.4 we use the bar over the variable A to indicate that the output B is the ***negation***, or opposite, of the input A. If A=1, then B=0. If A=0, then B=1. Using only ASCII text characters, you might see the NOT symbol written as, **B = /A,** or **B = ~A** with the forward slash or the tilde representing negation. Negation is also called the ***complement.***

The NOT gate also uses a small open circle on the output to indicate negation. A gate with a single input and a single output, but without the negation symbol, is called a ***buffer***. The output waveform of a buffer always follows the input waveform, minus the propagation delay. Logically there is no obvious need for a buffer gate, but electrically (those pesky hardware engineers again!) the buffer is an important circuit element. We'll actually return to the concept of the buffer gate when we study analog to digital conversion in a future lesson. Unlike the AND gate and the OR gate, the NOT gate always has a single input and a single output. Furthermore, for all of it's simplicity, there is no obvious way to show the NOT gate in terms of a simple flashlight bulb circuit. So we'll turn our attention to the OR gate.

We can look at the OR gate in the same way we first examined the AND gate. We'll use our simple flashlight circuit from figure 1.10, but this time we'll rearrange the switches to create the logical OR function. Figure 2.5 is our flashlight circuit.

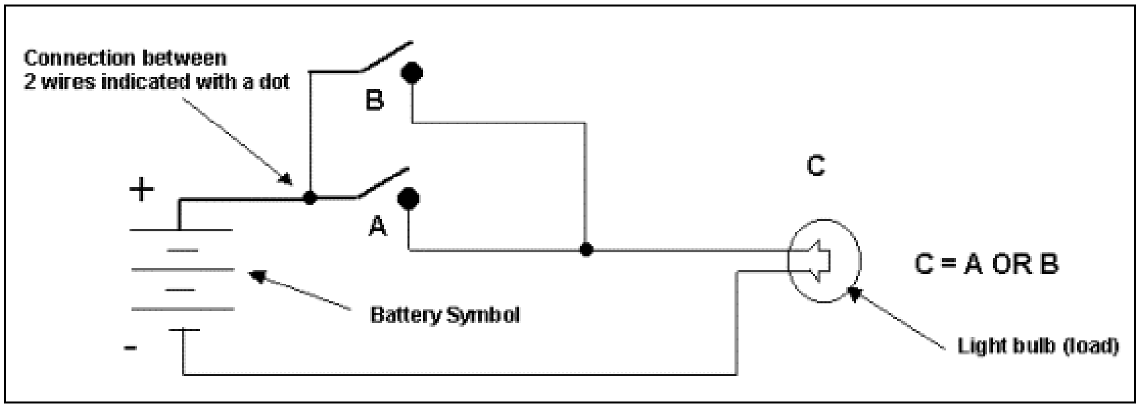
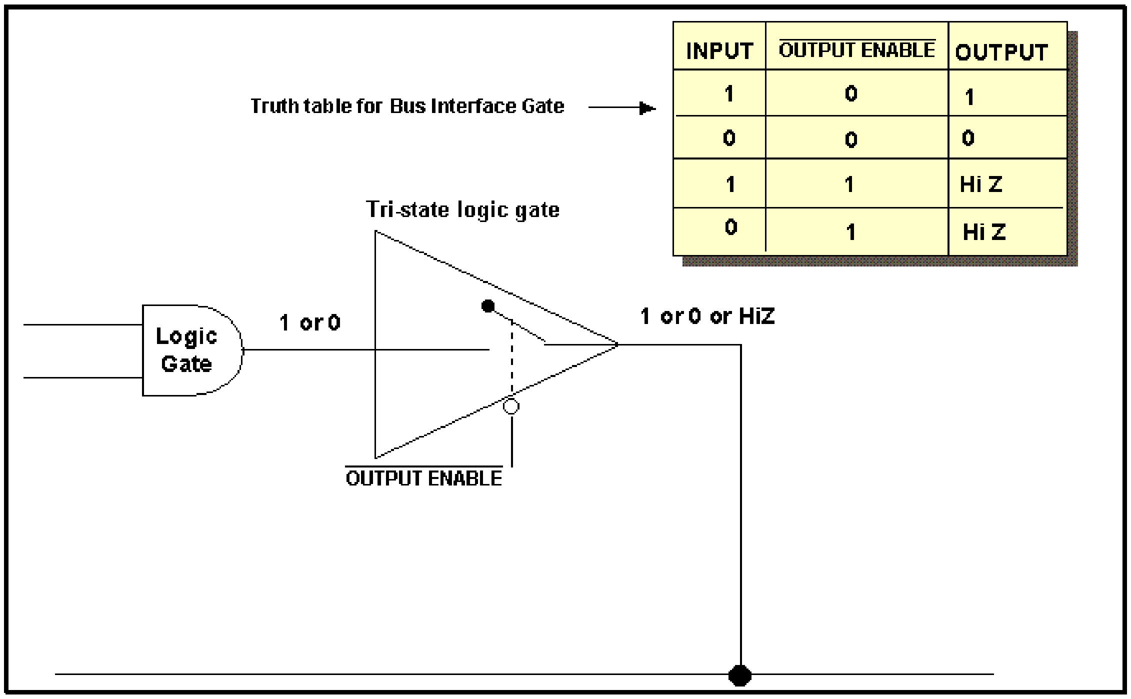


Figure 2.5: The logical OR function implemented as two switches in parallel. Closing either switch A or B turns on the light bulb, C.

The circuit in figure 2.5 shows the two switches, A and B, wired in parallel. Closing either switch will allow the current from the battery to flow through the switch into the bulb and turn it on. Closing both switches doesn't change the fact that the lamp is illuminated. The lamp won't shine any brighter with both switches closed. The only way to turn it off is to open both switches, thereby interrupting the flow of current to the bulb.

Finally, there is a small dot in figure 2.5 indicating that the two wires come together and actually make electrical contact at that point. As we've discussed earlier, since our schematic diagram is only two-dimensional, and printed circuit boards often have 10 layers of electrical conductors insulated from each other, it is sometimes confusing to see two wires cross each other and not be physically connected together. Usually when two wires cross each other, we get lots of sparks and the house goes dark. In our schematic diagrams, we'll represent two wires that actually touch each other with a small dot. Any other wires that cross each other we'll consider to be insulated from each other.

We still haven't looked at the tri-state logic gate. Perhaps we should, even though the reason for the gate's existence won't be obvious to you now. So, at the risk of letting the cat out of the bag, let's look at the fourth member of our atomic group of logic elements, the tri-state logic gate shown schematically in figure 2.6.


Truth table for Bus Interface Gate
Tri-state logic gate
Logic Gate
1 or 0
1 or 0 or HiZ
$\overline{\text{OUTPUT ENABLE}}$

INPUT	$\overline{\text{OUTPUT ENABLE}}$	OUTPUT
1	0	1
0	0	0
1	1	Hi Z
0	1	Hi Z

Figure 2.6: The tri-state (TS) logic gate. The output of the gate follows the input as long as the OUTPUT ENABLE (~OE) input is low. When the ~OE input goes high, the output of the gate enters the Hi-Z state.

There are several important concepts here so we should spend some time on this figure. First, the schematic diagram for the gate appears to be the same as an inverter, except there is no ***negation bubble*** on the output. That means that the logic sense of the output follows the logic sense of the input, after the propagation delay. That makes this gate a ***buffer gate***. This doesn't imply that we couldn't have a tri-state gate with a built-in NOT gate, but that would not be atomic. That would be a ***compound gate***.

The TS gate of figure 2.6 has another input, labeled ***~output enable***. The input on the gate also has a negation bubble, indicating that the gate is ***active low.*** We've introduced a new term here. What does "active low" mean? In the previous chapter we made a passing reference to the fact that logic levels were somewhat arbitrary. For convenience, we assigned the more positive voltage a value of 1, or TRUE, and the less positive voltage a value of 0, or FALSE. We call this convention ***positive logic.*** There's nothing special about positive logic, it is simply the convention that we've adopted.

However, there are times when we will want to assign the "TRUENESS" of a logic state to be low, or 0. Also, there is a more fundamental issue here. Even though we are dealing with a logical condition, the meaning of the signal is not so clear-cut. There are many instances where the signal is used as a controlling, or enabling, device. Under these conditions, TRUE and FALSE don't really apply in the same way as they would if the signal was part of a complex logical equation. This is the situation we are faced with in the case of a tri-state buffer.

The ~Output Enable (~OE) input to the tri-state buffer is active when the signal is in its low state. In the case of the TS buffer, when the ~OE input is low, the buffer is active, and the output will follow the input. Now, at this point you might be getting ready to say, "Whoa there Bucko, that's an AND gate. Isn't it?" and you'd almost be correct.

In figure 2.3 we introduced the idea of the AND logic function as a gate and when the A input was 0, the output of the gate was also 0, independent of the logic state of the B input. The TS buffer is different in a critical way. When ~OE is high, from an electrical circuit point of view, the output of the gate ceases to exist. It is just as if the gate wasn't there. This is the Hi-Z logic state. So, in figure 2.6 we have the unique situation that the TS buffer acts behaves like a closed switch when ~OE is low, and it acts like an open switch when ~OE is high. In other words, Hi-Z is not a 1 or 0 logic state, it is a unique state of its own, and has less to do with digital logic than with the electronic realities of building computers. We'll return to tri-state buffers in a later chapter, stay tuned!

There's one last new concept that we've introduced in figure 2.6. Notice the ***truth table*** in the upper right of the figure. A truth table is a shorthand way of describing all of the possible states of a logical system. In this case, the input to the TS buffer can have two states and the ~OE control input can have two states, so we have a total of 4 possible combinations for this gate. When ~OE is low, the output agrees with the input, when ~OE is high, the output is in the Hi-Z logic state and the input cannot be seen. Thus, we've described in a tidy little chart all of the possible operational states of this device.

We now have in our vocabulary of logical elements AND, OR, NOT and TS. Just like the building blocks of life in DNA, these are the building blocks of digital systems. In actuality, these three gates are most often combined to form slightly different gates called NAND, NOR and XOR. The NAND, NOR and XOR gates are ***compound gates***, because they are constructed by combining the AND, OR and NOT gates. Electrically, these compound circuits are just as fast as the primary circuits because the compound function is easily implemented by itself. It is only from a logical perspective that we draw a distinction between them.

Figure 2.7 shows the compound gates NAND and NOR. The NAND gate is an AND gate followed by a NOT gate. The logical function of the NAND gate may be stated as:

- OUTPUT C goes LOW if and only if input A is HIGH AND input B is HIGH.

The logical function of the NOR gate may be stated as:

- OUTPUT C goes LOW if input A is HIGH, or input B is HIGH, or if both inputs A and B are HIGH.

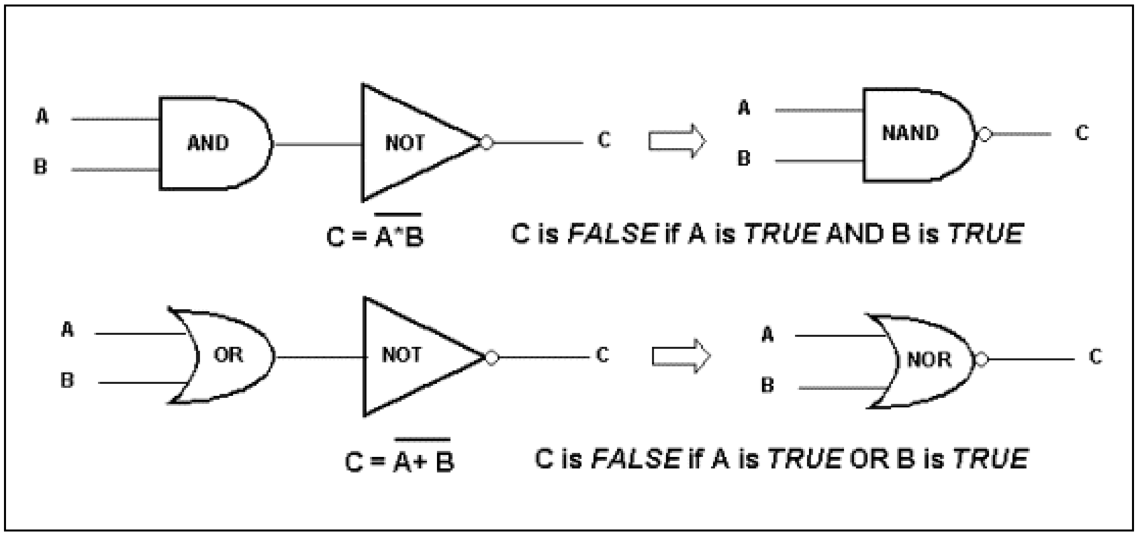

A
B
AND
NOT
C
A
B
NAND
C
C = $\overline{A*B}$
C is *FALSE* if A is *TRUE* AND B is *TRUE*
A
B
OR
NOT
C
A
B
NOR
C
C = $\overline{A+B}$
C is *FALSE* if A is *TRUE* OR B is *TRUE*

Figure 2.7: A schematic representation of the NAND and NOR gates as a combination of the AND gate with the NOT gate, and the OR gate with the NOT gate, respectively.

Finally, we want to study one more compound gate construct, the XOR gate. XOR is a shorthand notation for the ***exclusive OR gate*** (pronounced as "ex or"). The XOR gate is almost like an OR gate except that the condition when both inputs A and B equals 1 will cause the output C to be 0, rather than 1.

Figure 2.8 illustrates the circuit diagram for the XOR compound gate. Since this is a lot more complex than anything we've seen so far, let's take our time and walk through it. The XOR gate has two inputs, A and B, and a single output, C. Input A goes to AND gate #3 and to NOT gate #1, where it is inverted. Likewise, input B goes to AND gate #4 and its ***complement*** (negation) goes to AND gate #3. Thus, each of the AND gates has as its input one of the variables A or B, and the complement (negation), or opposite sense of the other variable, /A or /B, respectively. As an aside, you should now appreciate the value of the black dot on the schematic diagram. Without it, we would not be able to discern wires that are connected to each other from wires that are simply crossing over each other.

Thus, the output of AND gate #3 can be represented as the logical expression A*~B and similarly, the output of AND gate #4 is B*~A. Finally, OR gate #5 is used to combine the two expressions and allow us to express the output variable C as a function of the two input variables, A and B as, C = A*~B + B*~A. The symbol for the compound XOR gate is shown in figure 2.8 as the OR gate with an added line on the input. The XOR symbol is the plus sign with a circle around it.

Let's walk through the circuit to verify that it does, indeed, do what we think it does. Suppose that A and B are both 0. This means that the two AND gates see one input as a 0, so their outputs must be zero as well. Gate #5, the OR gate, has both inputs equal to 0, so its output is also 0. If A and B are both 1, then the two NOT gates, #1 and #2, negate the value, and we have the same situation as before, each AND gate has one input equal to 0.

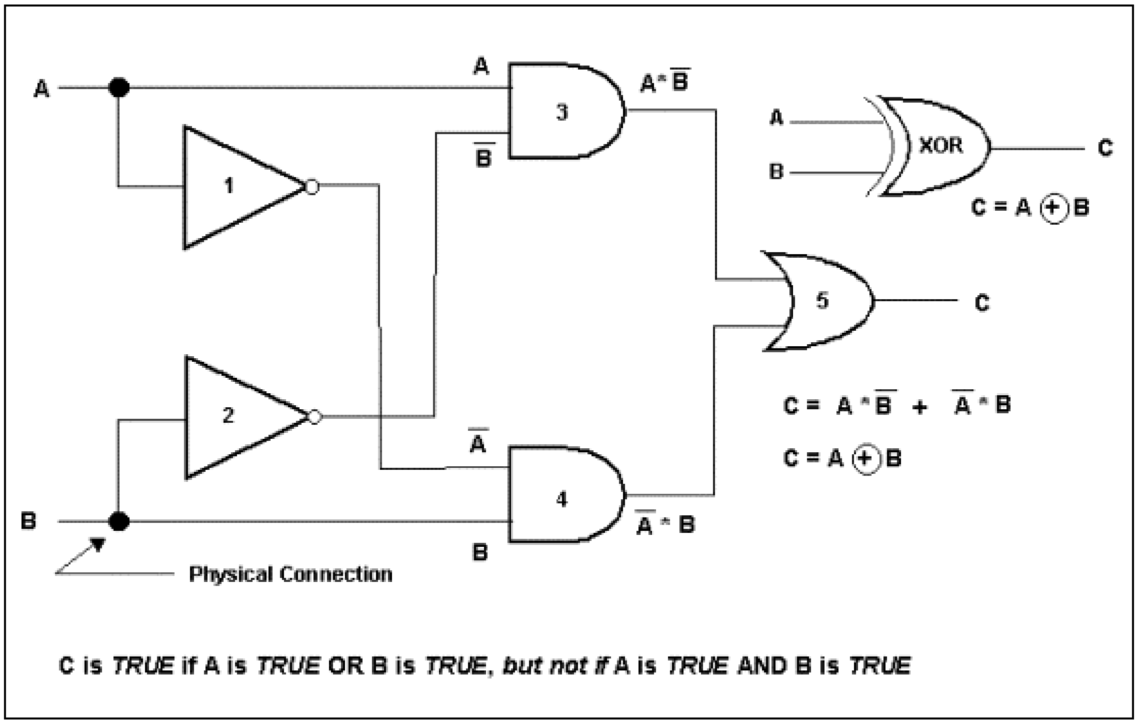


Figure 2.8: Schematic circuit diagram for an exclusive OR (XOR) gate.

In the third situation, either A is 0 and B is 1, or vice versa. In either case, one of the AND gates will have both of its inputs equal to 1 so the output of the gate will also be 1. This means that at least one input to the OR gate will be 1, so the output of the OR gate will also be 1. Whew! Another way to describe the XOR gate is to say that the output is TRUE if either input A is TRUE OR input B is TRUE, but not if ***both*** inputs A and B are TRUE.

You might be asking yourself, "What's all the fuss about?" As you'll soon see, the XOR gate forms one of the key circuit elements of a computer, the addition circuit. In order to understand this, let's suppose that we are adding two single-bit binary numbers together. We can have the following possibilities for A and B:

- A = 0 and B = 0: A + B = 0
- A = 1 and B = 0: A + B = 1
- A = 0 and B = 1: A + B = 1
- A = 1 and B = 1: A + B = 0, carry the 1

These conditions look suspiciously like A XOR B, provided that we can somehow figure out what to do about the "carry" situation.

Also, notice in figure 2.8 that we've shown that the output of the XOR gate, C, can be expressed as a logical equation. In this case,

C = ~A*B + ~B*A

This is another reason why we call the XOR gate a compound gate. We can describe it in terms of a logical equation of the more atomic gates that we've previously discussed.

Now suppose that we modify the XOR gate slightly and add a NOT gate to it after the output. In effect, we'll construct an XNOR gate. In this situation the output will be 1, if both inputs are the same, and the output will be 0 if the inputs are different. Thus, we've just constructed a circuit element that detects equality. With 32 XNOR gates we can immediately tell (after the appropriate propagation delay) if two 32-bit numbers are equal to each other or are different from each other.

Just as we might use 32 XNOR gates to compare the values of two 32-bit numbers, let's see how that might also work if we wanted to do a logical AND operation on two 32-bit numbers. You already know from your other programming courses that you can do a logical AND operation on variables that evaluate to TRUE or FALSE (Booleans), but what does ANDing two 32-bit numbers mean? In this case, the AND operation is called a ***bitwise AND***, because it ANDs together the corresponding bit pairs of the two numbers. Refer to figure 2.9. For the sake of simplicity, we'll show the bitwise AND operation on two 8-bit numbers, rather than two 32-bit numbers. The only difference in the two situations is the number of AND gates used in the operation.

In figure 2.9, we are performing the bitwise AND operation on the two byte values 0xAA and 0x55. Since each AND gate has one input equal to 1 and the other equal to 0, every AND gate has a 0 for its output. In C/C++, the ampersand, &, is the bitwise AND operator, so we can write the code snippet:

Code Example

```
char   inA = 0xAA;

char   inB = 0x55;

char   outC = inA & inB;

cout << "The bitwise ANDing of 0x55 and 0xAA = ," << outC << endl;
```

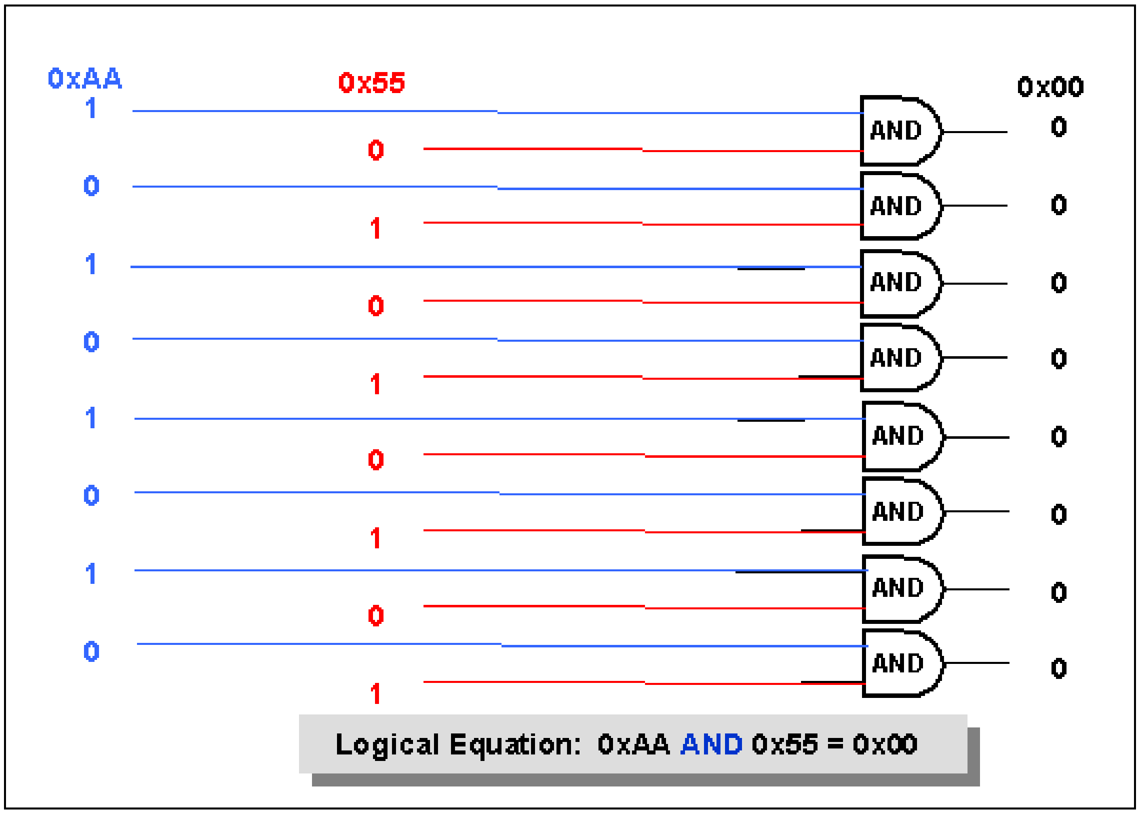


Figure 2.9: Bitwise AND operation on two 8-bit (byte) values. The operation performed is called a bitwise AND because it operates on the corresponding bit pairs of the two numbers. In C and C++, a hexadecimal number is represented with the prefix 0x.

If we wrote a real program and ran it, the result would be 0. As a preview of 68,000 assembly language, this same equation would be written:

Code Example

```
MOVE.B   #$AA,D0

ANDI.B   #$55,D0
```

The first assembly language instruction copies the byte, 0xAA, into an internal storage register, D0. The second instruction does a bitwise AND of the byte 0x55 with the contents of D0, 0xAA. The "ANDI.B" is interpreted as, "Do a logical AND operation on the byte value portion of register D0 with the immediate (literal) value $55." We could have written "ANDI.W" or "ANDI.L" to indicate 16-bit or 32-bit operations, respectively. The result, 0, is now stored in the register D0. If this doesn't make any sense to you now, don't worry. It might not make any sense to you later (but it probably will).

Up to now the AND, OR and their derivative gates were all represented with two inputs and one output. However, the AND gate and the OR gate may have an arbitrary number of inputs. To see this, let's consider figure 2.10

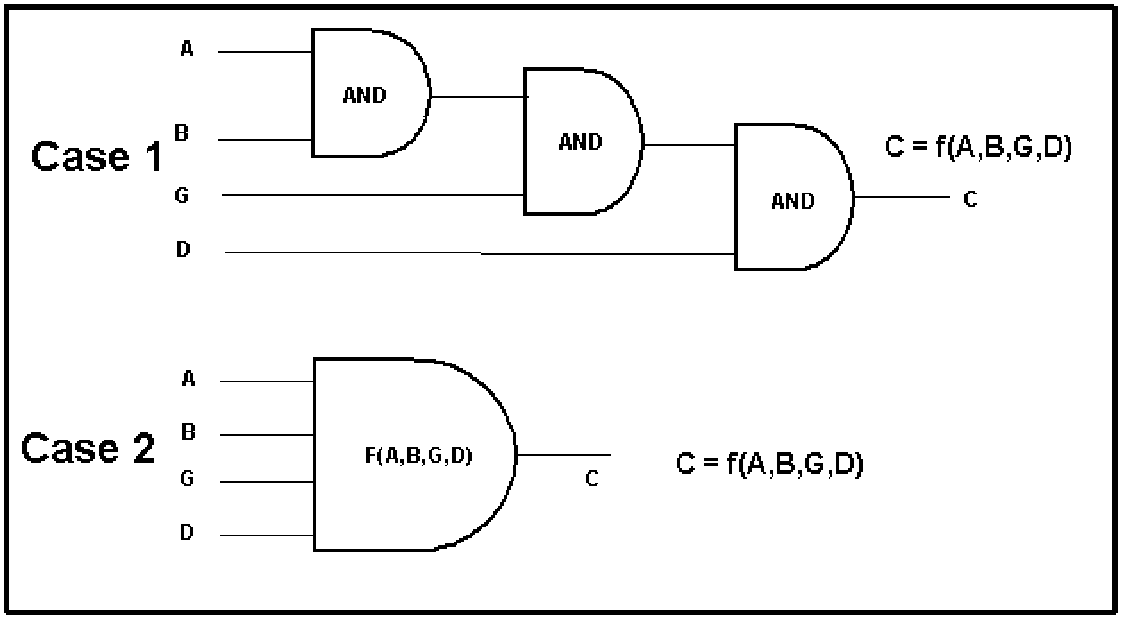


Figure 2.10: Logical equivalence of AND gates with more than 2 inputs. Although Case 1 and Case 2 perform identical logical functions, but they are not identical when logic speed (propagation delay) is taken into account.

Case 1 shows 3, two-input AND gates connected such that output C will be true if and only if inputs A,B,G and D are all true. This is simplified in Case 2. In fact, we can design the electrical circuit of Case 2 to be almost identical to the circuit of one of the gates in Case 1, the exception being the number of inputs being AND'ed together. However, if we assume that all the gates in figure 2.10 have exactly the same propagation delay, then the 4-input gate in Case 2 would have a propagation delay approximately 1/3 that of Case 1. You can apply the same analysis to the OR gate. Unfortunately, this fails with the XOR gate because the mathematical function of exclusive OR is always applied to only two input variables at a time.

Electronic gate description

Today, most gates are electronic devices made from integrated circuits (ICs). You might be wondering what is inside of a gate. That's a fair question. It's not the purpose of this book to also teach transistor theory or IC design, even though it might be fun to try. Let's let the Electrical Engineers have their mysteries; after all, you've seen what a mess they make when they try to write software. Let's take a quick peek at such an IC. Figure 2.11 is a picture of an industry standard part, the 74LS00.

The number '74' refers to the logic series of parts and the temperature range over which these parts are designed to work. '74' series parts are considered to be commercial parts, while '54' series are designated as "military" grade parts and are designed to operate over a wider range of temperatures. The letters 'LS' are an abbreviation for ***Low-power Schottky***, one of several standard IC manufacturing technologies. The last designation, '00', is the designation for an IC package containing 4 NAND gates with each NAND gate having two inputs and a single output. The part is contained in a plastic package with 14 pins, 7 on each side. This is called a ***DIP*** package, an abbreviation for dual-inline plastic. ICs like this are often called "bugs", for obvious reasons.

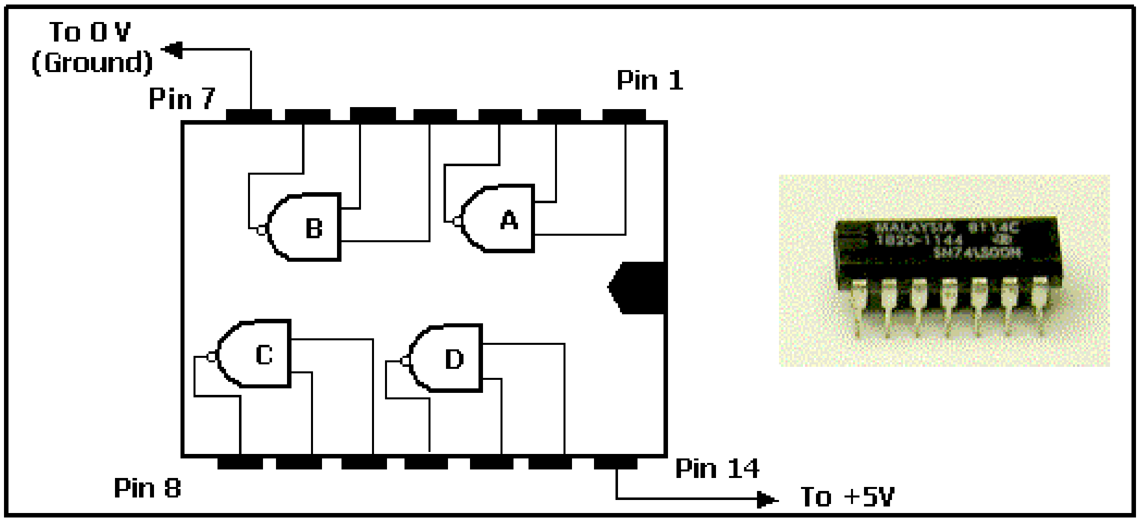


Figure 2.11: A quad, 2-input NAND gate (Industry Standard Designation 74LS00). The plastic package contains 4 independent NAND gates. The integrated circuit is encapsulated in a plastic package (shown on the right). The input pins for gate A are 1 and 2 and the output pin is pin 3.

The integrated circuit, or IC, shown in figure 2.11 contains 4 independent NAND gates. Each gate has two inputs and one output. Thus, the package requires 12 I/O pins for the gates themselves, plus one pin for power, in the case of the LS logic family +5 volts, and one pin for ground. This package costs about 10 cents to buy and any one of the gates has a propagation delay from input change to output response of about 5 nanoseconds.

We still haven't answered the question of what's actually inside a gate. I did assert that transistors make good switches and switches make good logic circuits, so let's see how it works. Rather than discuss the LS logic family, which is built on an older technology, let’s perform our analysis with a more modern technology called ***CMOS***, which is pronounced “sea moss”. CMOS is an abbreviation for ***Complementary Metal-Oxide Silicon***. CMOS is the dominant integrated circuit technology today and will be into the foreseeable future. Almost all modern microprocessors are built in CMOS technology. Also, you may have heard about this technology in the context of the image sensors for digital cameras. Since this is such an important technology, we should spend a few moments and try to understand it, at least in a cursory way.

In order to understand the basic CMOS construct, let’s return to something we do understand, mechanical switches. Figure 2.12

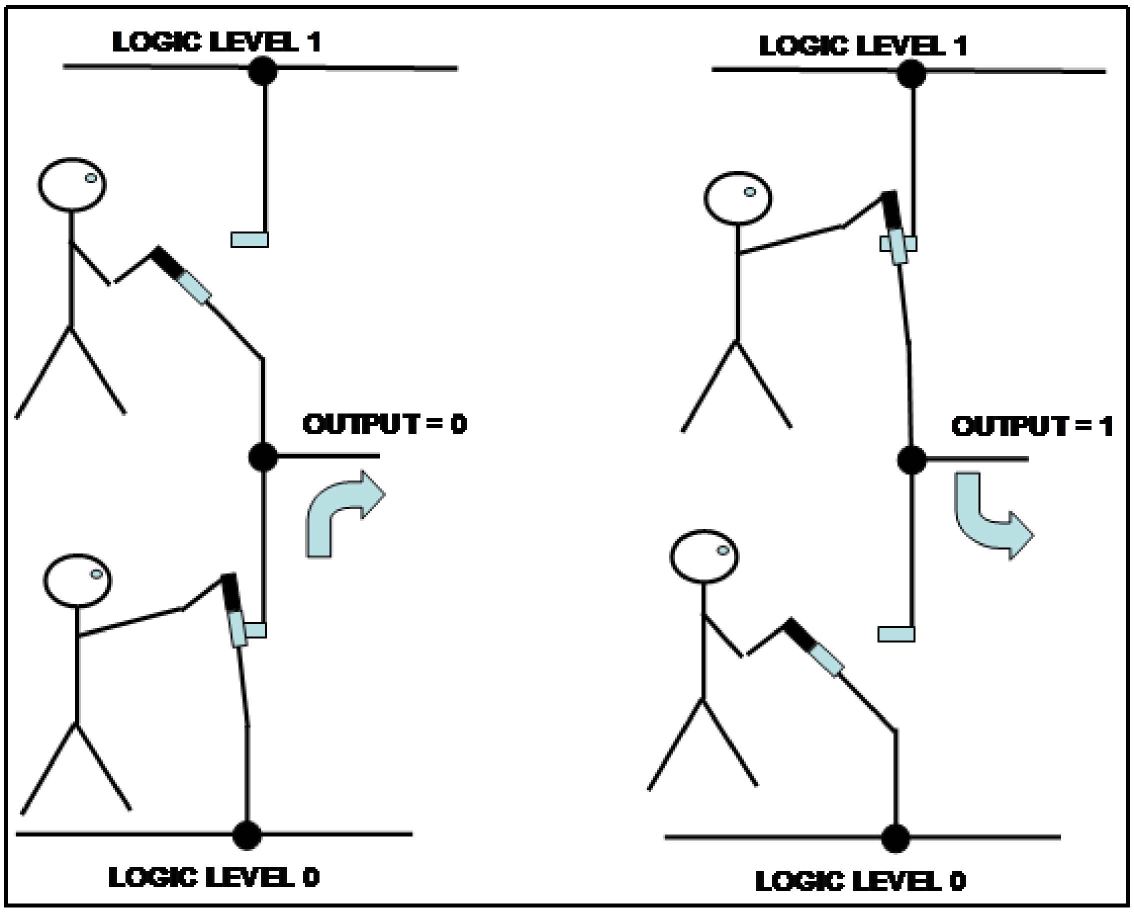


Figure 2.12: Mechanical switch representation of a CMOS switching circuit.

Try not to pay too much attention to the quality of the art work. Even with my stick figures the operation of the circuit should be clear. Imagine that we have two mechanical switches connected in series (one after the other) between our logic level 1 and logic level 0. In a real circuit, logic 1 would be the power supply and logic 0 would be the circuit ground, or 0 volts. Now, it should be obvious to you that you would never want to have both switches closed at the same time, otherwise bad things could happen. This would be similar to what happened when I was 5 years old and un-bent a paper clip to see what would happen if I stuck it into the light socket on the wall. Although my experiment was a bit more spectacular, and perhaps more of a learning experience than figure 2.12 might provide, conceptually, we have the same result.

In order to output a logic level 0, we close the switch to the logic 0 line and open the switch to the logic 1 line. A observer looking into the output of the circuit would see a logic 0, because that switch is connecting us to that logic level. The alternative situation gives us a logic 1 condition. When we close the upper switch and open the lower switch, we "see" logic level 1.

Hopefully, one important fact should now be clear to you. When we first discussed the concept of rising and falling edges for logic signals, I stated that these edges must be very rapid transitions between the logic states. Figure 2.12 shows us why. We never want to have a situation where both of these switches are closed at the same time. So, if the logic transition is somehow controlling the opening and closing of these switches, then any overlap that might occur between one switch opening and the other switch closing should be as brief as possible.

Figure 2.13 is an actual circuit configuration for a CMOS logic gate. It happens to be a NOT gate. Let me first say that I'm showing you this figure with great trepidation. For one, I am concerned that I am letting too much sacred information out of the bag and the brotherhood and sisterhood of EEs will seek retribution. Also, I'm not sure that in the greater subject area of Computer Architecture that this is an essential discussion. Anyway, let's forge ahead. You can be the judge.

The two circuit devices represented by the symbols in the green and yellow circles are called ***MOSFETs.*** MOSFET is an abbreviation for Metal Oxide Silicon Field-Effect Transistor. Whew! MOSFETs come in two flavors, n-type, or n-channel and p-type, or p-channel. The symbol for the n-channel device has the arrowhead pointing towards the device and the arrowhead in the p-channel device has the arrowhead pointing away from the device. Whether you have an n-type or p-type depends upon how the device is manufactured. In particular, the resulting device depends upon what type of impurities are added to the silicon in order to impart the desired electrical characteristics.

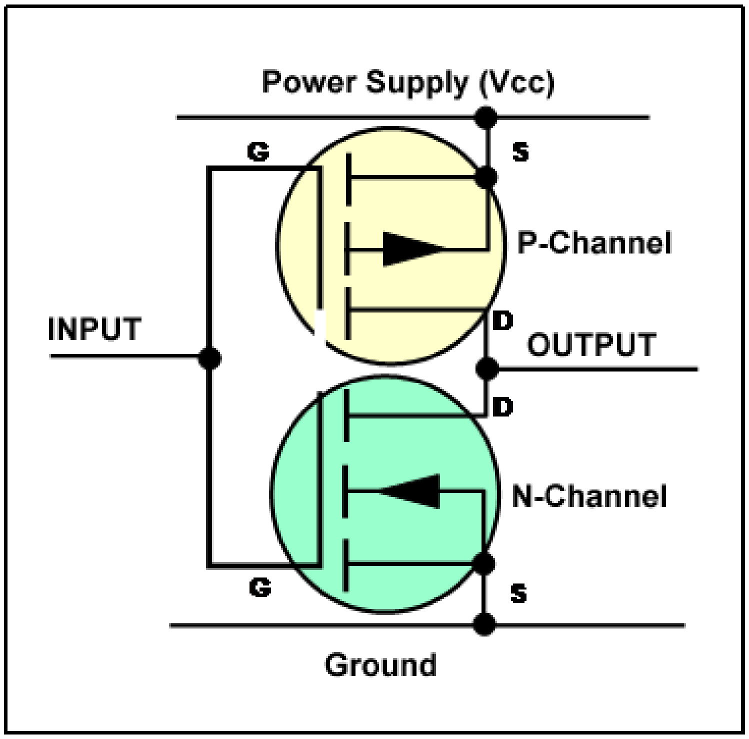


Figure 2.13: CMOS circuit configuration

CMOS devices use the two MOSFETs in pairs. An n-type and a p-type are paired together. Aside from their manufacturing differences, the main distinguishing characteristic is that the n-channel device is used in the sense of a positive voltage, and the p-channel device is used in a negative voltage sense. If this doesn't mean anything to you now, keep reading. I promise that it gets better. Anyway, aside from the fact that one transistor is sort of a positive device and the other is sort of a negative device, both transistors behave pretty nearly identically, so it is customary to call them ***complementary***. Thus, by pairing an n-type MOSFET with a p-type MOSFET of similar characteristics, we have a complementary pair, or a CMOS gate.

Each of the devices has 3 terminals of interest. The terminal labeled with a 'G' is the ***gate***, the terminal with the 'D' is the ***drain***, and the terminal with the 'S' is called the ***source.*** There is sometimes a 4th terminal, the ***substrate,*** or body (B). This terminal is sometimes brought out as a separate control, but in our circuit configuration it is connected to the source and we need not be concerned about it. In order to see how the CMOS gate behaves the way it does, we should take a quick peek at a simple graph of the behavior of a MOSFET device. Figure 2.14 shows schematically the electrical resistance from the drain to the source (R_{DS}) of a MOSFET device as a function of voltage on the gate, relative to the source, or V_{GS}. Now, its true that we really haven't defined resistance yet, so I don't expect that it should mean much to you at this point, but conceptually, it is the same as the electrical impedance that we discussed earlier in the context of the tri-state gate.

Consider the curve for n-channel device in figure 2.14. We see that the resistance of the device drops dramatically from a value of several 10's of millions of ***ohms*** to a value around 10 ohms as the voltage on the gate, relative to the source increases. In real terms, the amount that the voltage on the gate has to vary to cause this dramatic change is exactly the voltage swing from logic 0 to logic 1. As far as our electrical circuit analogy of figure 2.12 is concerned, this is equivalent to closing the switch as we raise the voltage on the gate. As you can see, the p-channel device behaves in a similar manner as the voltage on the gate becomes more negative than the voltage of the source. In other words, it exhibits complementary behavior. Now, we can finally put this all together and understand CMOS gates in general and figure 2.13 in particular.

Refer back to figure 2.13. Recall that this is a NOT gate. Let's see why. Assume that the voltage on the gate is a logic 0. The n-type transistor is essentially an open switch. The resistance is extremely high (because V_{GS} is nearly zero). However, the complementary device, the p-type device has a very low resistance because the voltage on the gate is much lower then the voltage of the source ($-V_{GS}$ is large). This is the closed switch of figure 2.12.

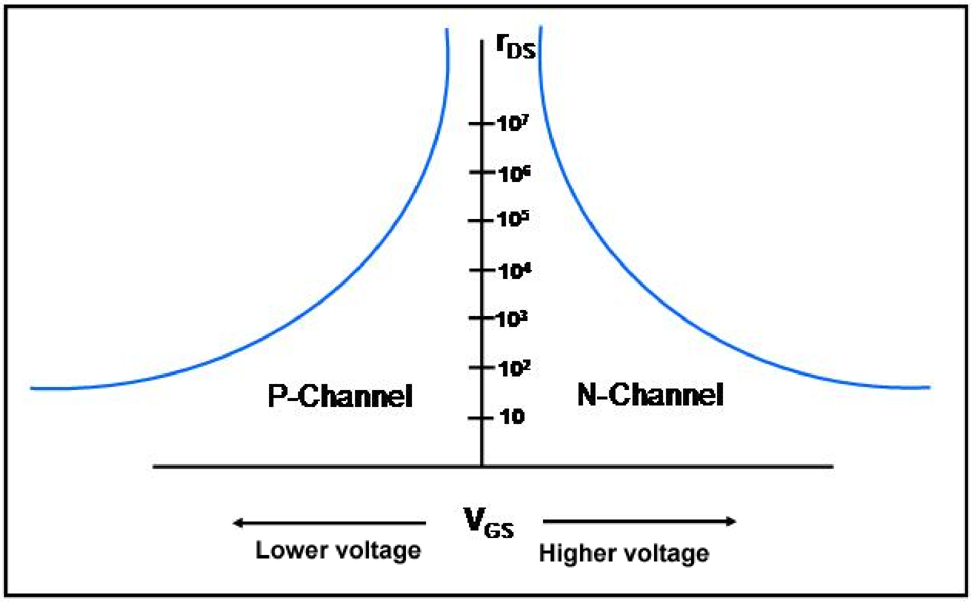


Figure 2.14: Electrical behavior of an n-channel and p-channel MOSFET device. The electric resistance across the device is plotted on a logarithmic scale as a function of the voltage on the gate of the device. *From Watson[2].*

Thus, at the output of the gate we see a low resistance (closed switch) to logic 1 and a high resistance (open switch) to logic 0 and the output of the gate is logic 1. So, when the input to the gate is logic 0, the output from the gate is logic 1. You should be able to analyze the complementary situation for an input of logic level 1. So, we can summarize the behavior of the CMOS NOT gate as follows. When the input is at logic 0, the output "sees" the power supply voltage, or logic 1. When the input is logic 1, the output sees the ground reference voltage, or logic 0.

Figure 2.14 illustrates the electrical behavior of MOS transistors. Consider the N-channel device on the right hand side of the graph. As the voltage on the gate, measured relative to the source (V_{GS}), becomes higher, or more positive, the electrical resistance between the drain and the source decreases exponentially. Conversely, when V_{GS} approaches zero, or becomes negative, the resistance between the drain and the source approaches infinity. Essentially, we have an open switch when V_{GS} is zero or negative, and almost a closed switch when V_{GS} is a few volts. The behavior of the P-channel device is identical to the N-channel device, except that the relative polarity of the voltages are reversed.

Before we leave this topic of the electrical behavior of a CMOS gate, you might be asking yourself, "What about the situation where the voltage on the gate, relative to the source, isn't all one way or the other, but in the middle?" In other words, what happens when the rising edge or the falling edge of the logic input to the gate is transitioning between the two logic states. According to the graph of figure 2.14[2], if the resistance isn't too high, and it isn't too low, which is analogous to the temperature of the porridge in "Goldilocks and the Three Bears", then at that moment, there is a path for the current to flow from the power supply to ground, so we do waste some energy. Fortunately, the transition between states is fast, so we don't waste a lot of energy, but nevertheless, there is some power dissipation occurring.

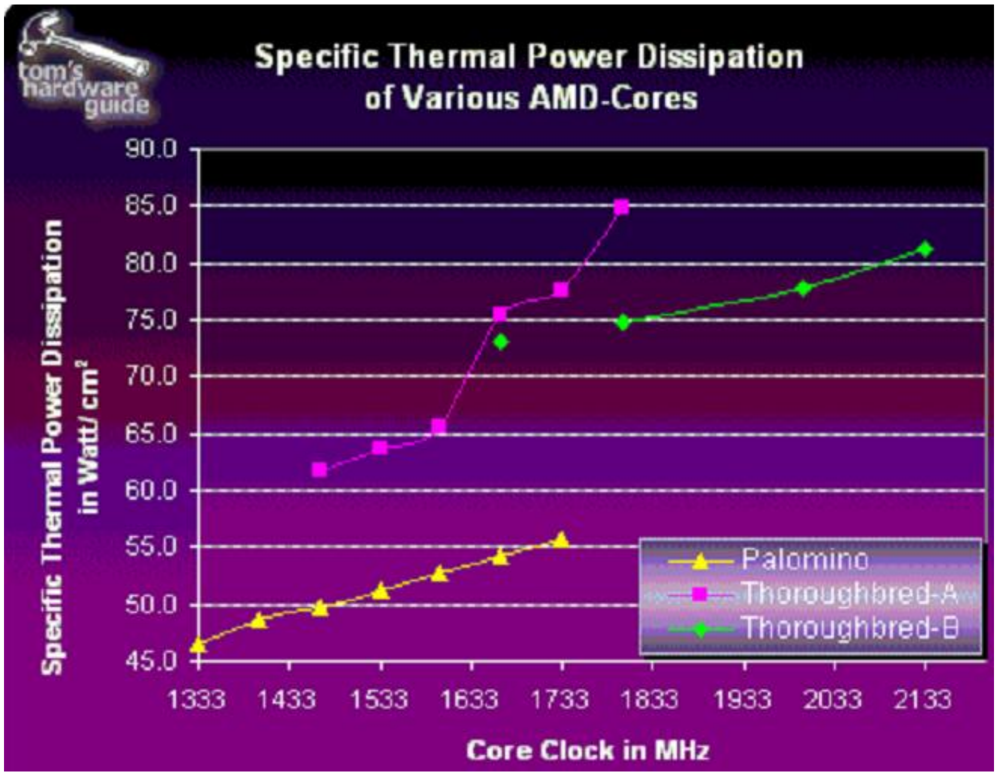


Figure 2.15: Power dissipation versus clock rate for various AMD processor families. From *www.tomshardware.com*[3].

[2] See the supplement at the end of this chapter for a more in depth explanation of how the CMOS gate works.

Well how bad is it? Recall that a modern processor has tens of millions of CMOS gates and a large fraction of these gates are switching a billion or more times per second. Figure 2.15 should give you some idea of how bad it could be.

Notice two things. First, these modern processors really run hot. In fact, they are dissipating as much heat as an ordinary 60 to 75 watt light bulb. Also, in each case, as we turn up the clock and make the processors run faster, we see the power dissipation going up as well.

You might also be asking another question, “Suppose we turn the clock down until it is really slow, or even stop it, will that have the reverse effect?” Absolutely, by slowing the clock, or shutting down portions of the chip, we can decrease the power requirements accordingly. This is a very important strategy in laptop computers, which have to make their batteries last for the duration of an airplane flight from New York to Los Angeles. This is also the strategy of how many other microprocessors, the kind that are used in embedded applications, can be attached to a collar around the neck of a moose and track the animal’s movements for over two years on no more power than a AAA battery. No wonder CMOS is so popular.

OK, we’ve seen a NOT gate. That’s a pretty simple gate. How about something a little more complicated? Let’s do a NAND gate in CMOS.

Recall that for a NAND gate, the output is 0 if all the inputs are logic 1. In figure 2.16, we see that if all of the inputs are at logic 1, then all of the n-channel devices are turned on to the low resistance state. All of the p-channel devices are turned off, so, looking back into the device from the output, we see a low resistance path to the ground reference point (logic 0). Now, if any one of the 4 inputs A, B, C or D is in logic state 0, then its, n-channel MOSFET is in the high-resistance state and its p-channel device is in the low-resistance state. This will effectively block the ground reference point from the output and open a low resistance path to the power supply through the corresponding p-channel device.

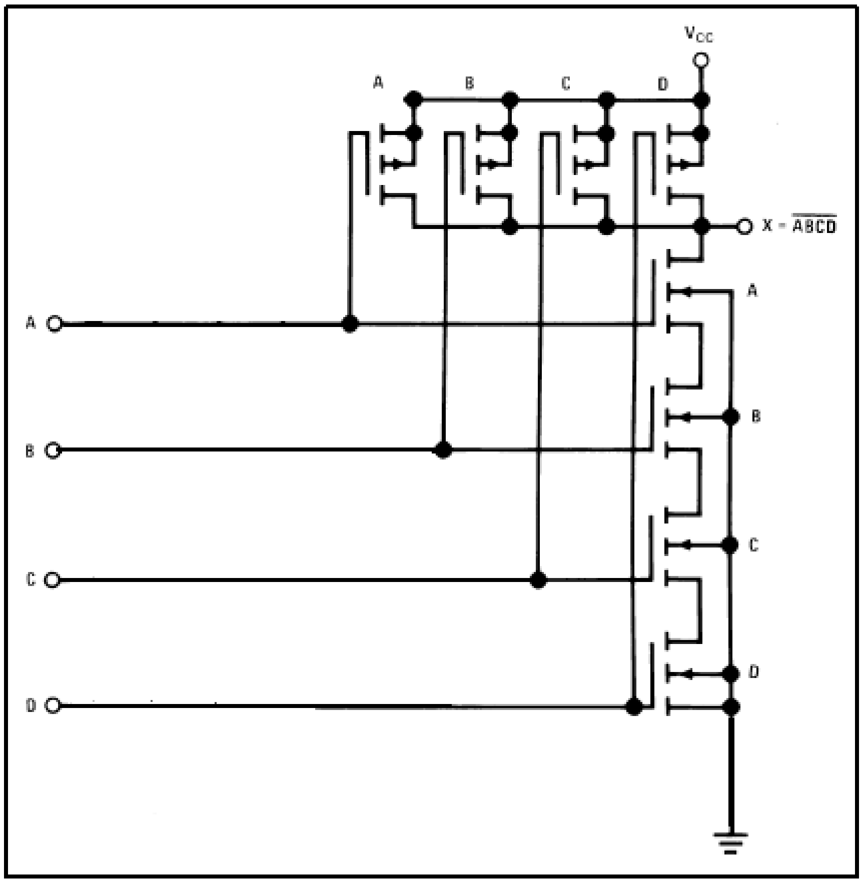


Figure 2.16: Schematic diagram of a CMOS, 4-input NAND gate. *From Fairchild.*[4]

Hopefully, at this point you are getting to be a little more comfortable with your burgeoning hardware skills and insights. Let's analyze a circuit configuration. Consider figure 2.17. This is often called the "Shakespeare Circuit." You may feel free to ponder the deeper significance of the design. Note that I included this example just to prove that all computer designers are NOT humorless geeks.

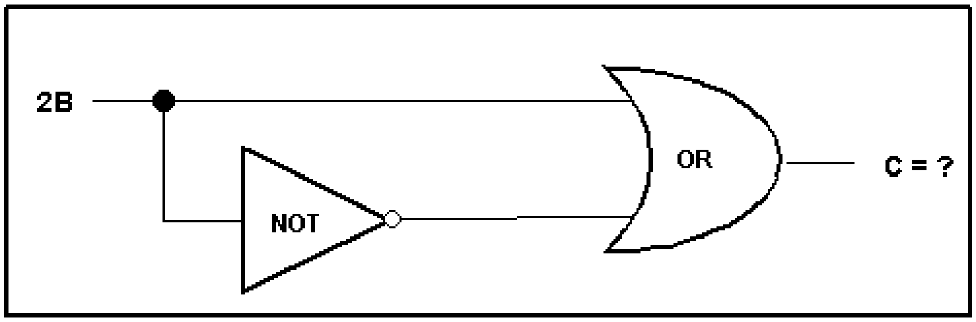
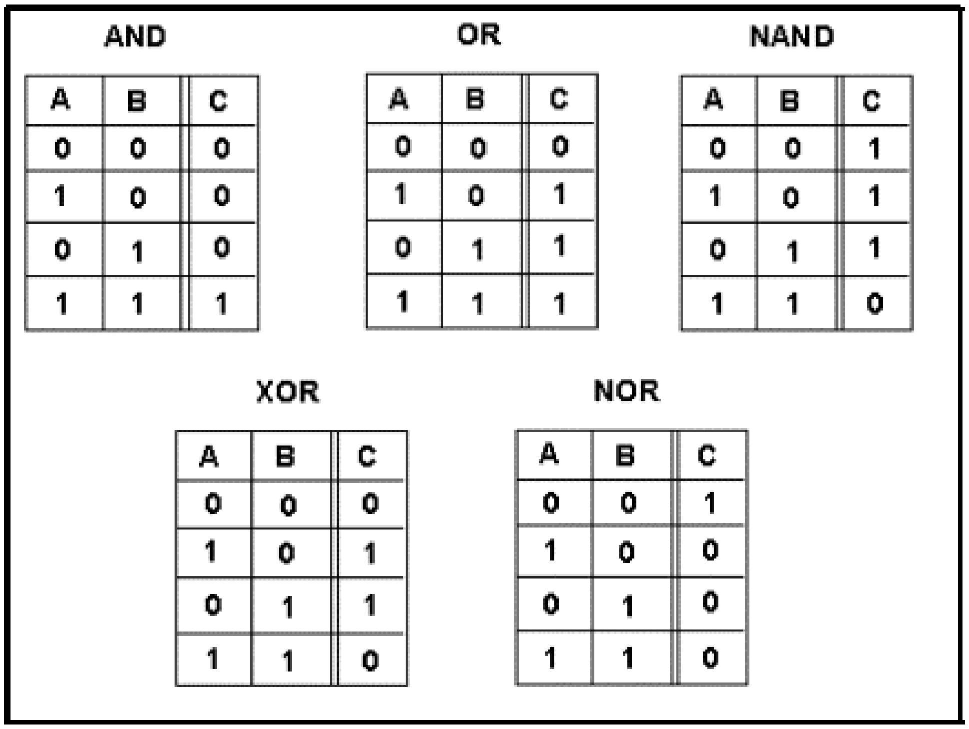


Figure 2.17: The Shakespeare Circuit

Truth Tables

The last concept that we'll discuss in this lesson is the idea of the ***Truth Table.*** You had a brief introduction to the Truth Table when we discussed the behavior of the tri-state logic gate. However, it is appropriate at this point in our analysis to discuss it more formally. The truth table is, as its name implies, a tabular form that represents the TRUE/FALSE conditions of a logical gate or system. For each of the logical gates we've studied so far, AND, OR, NAND, NOR and XOR, we've used a verbal expression to describe the function of the gate.

For example, with an AND gate we say, "The output of an AND gate is 1 if and only if both inputs are 1." This does get the logic across, but we need a better way to manipulate these gates and to design more complex systems. The method that we use is to create a truth table for the logic function. Figure 2.18 shows the truth tables for the 5 logic gates we've studied so far. The NOT gate is trivial so we won't include it here, and we've already seen the truth table for the tri-state gate, so it isn't included in the figure.



AND

A	B	C
0	0	0
1	0	0
0	1	0
1	1	1

OR

A	B	C
0	0	0
1	0	1
0	1	1
1	1	1

NAND

A	B	C
0	0	1
1	0	1
0	1	1
1	1	0

XOR

A	B	C
0	0	0
1	0	1
0	1	1
1	1	0

NOR

A	B	C
0	0	1
1	0	0
0	1	0
1	1	0

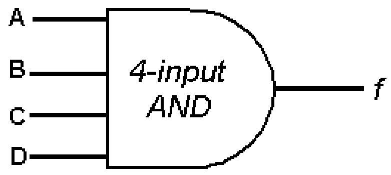
Figure 2.18: Truth table representation for the logical functions AND, OR, NAND, NOR and XOR.

The truth table shows us the value of the output variable, C, for every possible value of the input variables, A and B. Since there are two independent input variables, the pair can take on any one of four possible combinations. Referring to figure 2.18, we see that the output of the AND gate, C, is a logical 1 only when both A and B are 1. All other combinations result in C equal to 0. Likewise, the OR gate has its output equal to 1, if any one of the input variables, or both, is equal to 1. The truth table gives us a concise, graphical way to express the logical functions that we are working with.

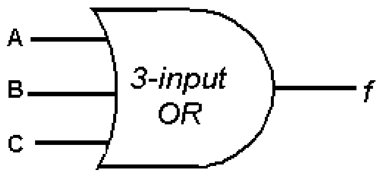
Suppose that our AND gate has three inputs, or four inputs? What does the truth table look like for that? If we have three independent input variables, A, B, and C, then the truth table would have eight possible combinations, or rows, to represent all of the possible combinations of the input variables. If we had a 3-input AND gate, only one condition out of the eight, (A=1, B=1, C=1) would produce a 1 on the gate's output. For four input variables we would need 16 possible entries in our truth table.

Thus, in general, our truth table must have $2^{\mathbf{N}}$ entries for a system of N independent input variables. There's that pesky binary number system again. Figure 2.19 shows the truth tables and logic symbols for a 4-input AND gate and a 3-input OR gate. It is possible to find commercially available AND, NAND, OR and NOR circuits with 5 or more inputs. There are also versions of the NAND circuit that are designed to be able to expand the number of inputs to arbitrarily large values, although there are probably not many reasons to do so.

The XOR gate is an exception because it is usually only defined for two inputs. While we could certainly design a circuit with, for example, 5 inputs, A through E, and 1 output, ***f***, that has the property that ***f*** = 0 if and only if all the inputs are the same, it would not be an XOR gate in the strictest sense. It would be an arbitrary digital circuit.



A	B	C	D	f
0	0	0	0	0
1	0	0	0	0
0	1	0	0	0
1	1	0	0	0
0	0	1	0	0
1	0	1	0	0
0	1	1	0	0
1	1	1	0	0
0	0	0	1	0
1	0	0	1	0
0	1	0	1	0
1	1	0	1	0
0	0	1	1	0
1	0	1	1	0
0	1	1	1	0
1	1	1	1	1



A	B	C	f
0	0	0	0
1	0	0	1
0	1	0	1
1	1	0	1
0	0	1	1
1	0	1	1
0	1	1	1
1	1	1	1

Figure 2.19 Truth table and gate diagrams for a 4-input AND gate and a 3-input OR gate.

Consider the gray box of figure 2.20. This is an arbitrary digital system that could be the control circuits of an elevator, a home heating and air conditioner, or fire alarm controller. Ultimately, we will be designing the circuit that is inside of the gray box as a circuit built from the logic gates that we've just discussed. Whatever it is, it is up to us to specify the logical relationships that each output variable has with the appropriate input variable or variables. Since the gray box has eight input variables, our truth table would have 256 entries to start with. In other words, we would design the system by first specifying the behavior of X,Y, and Z (output conditions) for each possible state of its inputs (a through h). Thus, if we were designing a heating system for a building and one of our inputs is derived from a temperature sensor that is telling us that the temperature in the room is lower than the temperature that we set on the thermostat, then it would be logical that this condition should trigger the appropriate output response (ignite burner in furnace).

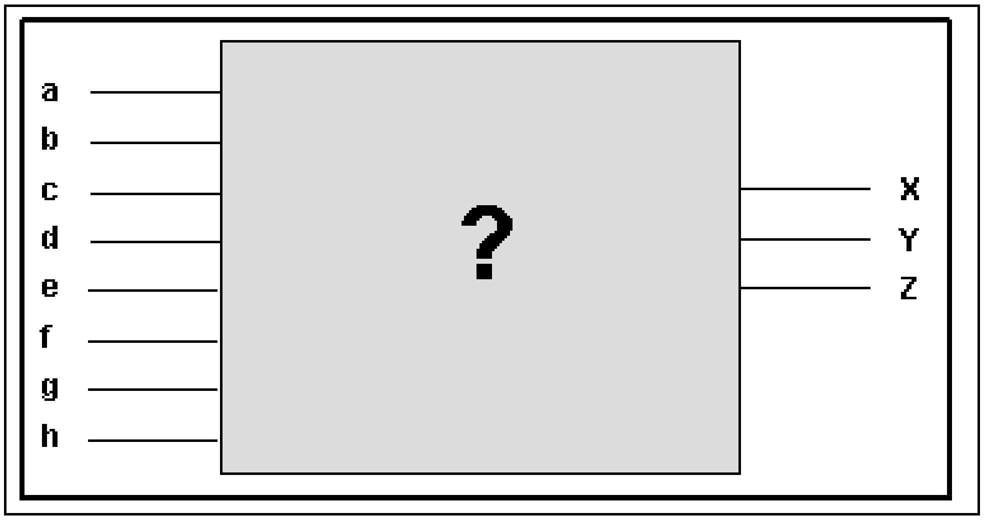


Figure 2.20: A digital system being designed. Each of the output variables X, Y and Z is described in a separate truth table in terms of the combinations of all the possible states of the input variables.

Referring to figure 2.20, we might start out our design process by creating a spreadsheet with 256 rows in it and 11 columns. We would devote one column for each of the eight input variables and a separate column for each of the three output variables. Next we would painstakingly fill in the table with all of the 256 possible combinations of the input variables, 00000000 through 11111111. Finally—and here's where the real engineering starts—for each row, we would have to decide how to assign a value to the output variables.

We'll go into the process of designing these systems in much greater detail in Chapter 3. For now, let's summarize this discussion by recognizing that we've used the truth table in two different, but related, contexts:

1. The truth table could be used as a tabular format for describing the logical behavior of one of the standard gates (AND, OR, NAND, NOR, XOR or TS).
2. We can specify an arbitrarily complex digital system by using the truth table to describe how we want the digital system to behave when we finally create it.

CMOS Supplement

Ever since I wrote the first edition of this book, students continued to come up after class and ask me for a better explanation of how the CMOS gate works than the one that I presented on the previous few pages. Therefore, I am going to add this supplemental material in the hope that it provides the interested reader with further insight into how digital logic gates operate at the transistor level.

In Chapter 12 I introduce you to the Ohm's Law for the first time. Electrical Engineering students taking this class learn Ohm's Law in their first circuits class, or perhaps even earlier, say a high school Physics class. In any case, I'm going to assume that you CS students are blissfully ignorant of how circuits work, so a little groundwork is necessary.

Ohm's Law relates the flow of electrical current through a material with the drop in voltage that accompanies the current flow. For example, a typical AA battery puts out 1.5 volts due to the chemical reaction going on inside of the battery. If you connect the battery to an appropriate flashlight bulb, current will flow through the bulb and the bulb would light.

Consider figure 2.21, below. By placing out trusty digital voltmeter across the terminals of the battery we can see that the 1.5 volts produced by the battery is also appearing across our load (the light bulb).

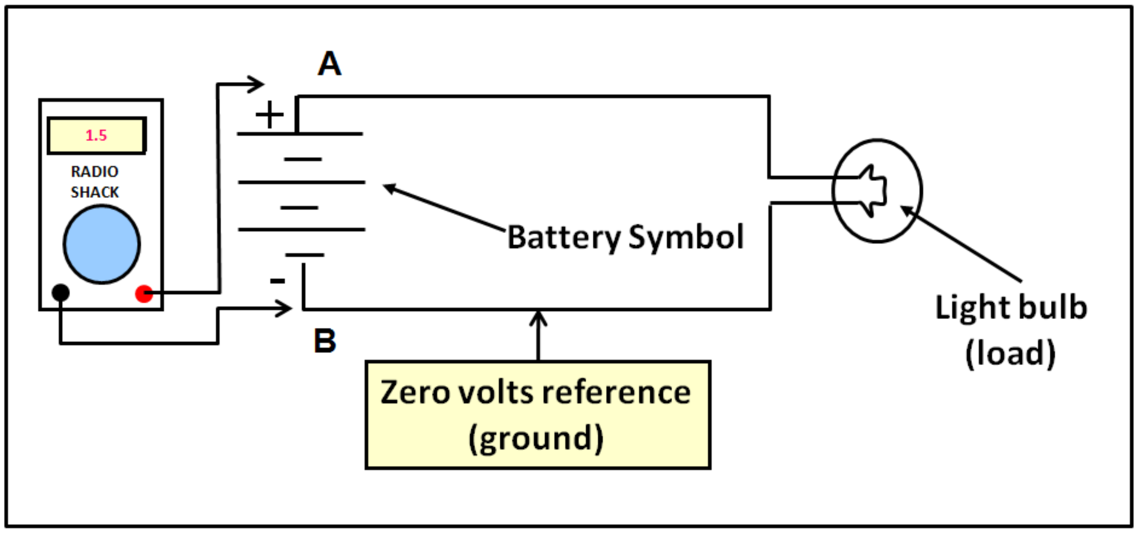


Figure 2.21 A simple battery circuit illustrates Ohm's Law.

If you placed one of the probes of a voltmeter on the positive terminal of the battery (A) and the other probe on the negative terminal of the battery (B) you would measure a voltage of around 1.5 volts. If you placed both probes on point B you would measure 0 volts. This trivial result leads us to an important concept. Namely, voltage in different points of a circuit are measured relative to each other. However, for convenience, we often say that the zero volts reference voltage is called "ground."

How much current is flowing through the load? Suppose that the resistance of the filament of the bulb is 100 units of resistance, or ohms, when the bulb is illuminated. Ohm's Law tells us that the current flowing through a resistance can be expressed as the ratio of the voltage across the resistance to the resistance of the element. Thus, in this circuit, the current I, measured in amperes, is equal to:

$I = V / \Omega$. Here the Greek letter omega (Ω) is the symbol for resistance. Thus, our flash-light bulb has a resistance of 100 Ω and the current, $I = 1.5 / 100 = 1.5\text{x}10^{-2}$ A, or 15 mA when expressed in engineering units.

Ohm's Law is typically expressed in a slightly different form: $V = I\text{x}R$. This tells us that the voltage drop across a circuit component is equal to the product of the current flowing through that component multiplied by the resistance of the component.

As you might suspect, not all resistors are flashlight bulbs. There are probably tens of thousands of different size, shapes and values of resistors. You can find resistors with resistance of milliohms to resistors with resistances of hundreds of megaohms. You can also find resistors so small that you can barely see them with the naked eye and resistors so large that you would only be able to lift them with help from a friend. Figure 2.22 shows a generic resistor that you might find in any electronic circuit.

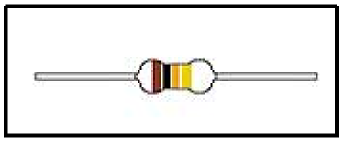


Figure 2.22 A typical resistor

Resistors are rated in terms of how much power you can dissipate inside of them. Clearly a light bulb gets hot when it is turned on, so heat must be generated by the current passing through the resistor. In fact, the heat being dissipated in a resistor is given by the equation:

$$\text{Power (in watts)} = I^2\text{x}R.$$

Using Ohm's Law we can substitute for I and find an alternative representation for Ohm's Law:

$$\text{Power (in watts)} = V^2/ R.$$

So, if you are an EE student designing a circuit, you would decide on how big a resistor you need based upon how much power you expect to be dissipated in the resistor during normal circuit operation. The resistor shown in figure 2.22 is approximately life-size and can dissipate about 1/4 of a watt.

The colored bands on the resistor represent a code that gives us the resistance of the resistor. If you are interested you can find plenty of references on deciphering the color codes of resistors.

Resistors are used in circuits to establish voltage levels and to limit the amount of current that can flow. Let's see how they might do that. Consider the simple circuit in figure 2.23

Here the same battery from figure 2.21. Note, you better read this quickly before the battery runs out of "juice." The current, I, flows from the positive terminal of the battery (A) through the two resistors, R_1 and R_2 , then back into the negative terminal of the battery (B). As an aside, the symbol used to represent resistors in schematic diagrams is a jagged line. There are different interpretations for why that is. Some say it represents a difficult path for the current to travel along, hence, resistance. Others (me included), are in the camp that believe it looks like the way resistance wire is strung inside space heaters and toaster. You are free to develop your own theory.

All of the current that exits the positive terminal must return to the negative terminal, closing the loop. Only voltage is lost in the process, since the charge carriers initially have 1.5 volts when they enter R_1 and have 0 volts when they exit R_2.

We might ask two questions about this circuit:

1. What is the value of the current, I, flowing in the circuit?
2. What is the voltage at point C?

Again, we might not ask these questions but please stay with me here. Hopefully, it is obvious to you that the total resistance that the current must flow through is comprised of R_1 and R_2 so it is reasonable to assume that the total resistance, $R_T = R_1 + R_2$.

Therefore, $I = V / (R_1 + R_2)$.

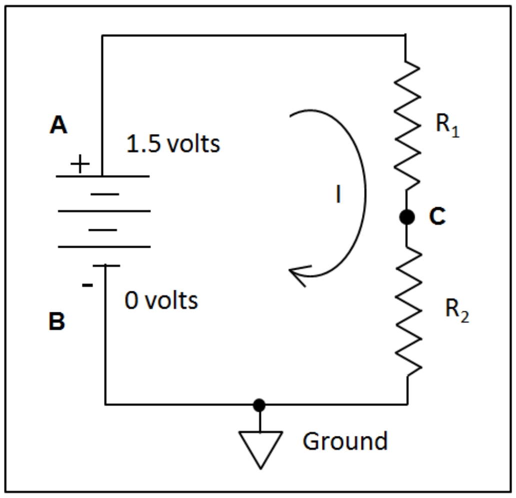


Figure 2.23 A circuit with two resistors representing a voltage divider.

Answering the second question, we might make an educated guess that the voltage at point C has got to be somewhere between 0 and 1.5 volts, since it can't be greater than the battery voltage or less than 0. Let's use ground as our zero voltage reference point and figure out the voltage drop across resistor R_2. Ohm's Law tells us that it is simply $I x R_2$, or $R_2 \times V / (R_1 + R_2)$.

Thus, we can express the value of the voltage at point C as the value of the battery voltage, 1.5 volts, multiplied by the fractional value of $R_2 / (R_1 + R_2)$. We can see that if $R_1 = R_2$ then the voltage at point C,V_C , equals 3/4 of a volt. We call this circuit a voltage divider because the output of the circuit is the input to the circuit divided by a value determined by the values of the resistors. Also note that once we've derived the equation for the voltage divider, we no longer have to go through the intermediate step of calculating the total current in the circuit.

OK, let's get back to CMOS logic. Too much EE stuff makes the EE union nervous. Consider Figure 2.23 with a new wrinkle. Suppose that R_1 and R_2 are not fixed resistance values, but rather a special kind of resistor whose resistance can change depending upon a third voltage value, V_X . We'll call R_1 a P-type resistor and we'll call R_2 an N-type resistor. The resistance of the two types of resistors can be expressed on the graph in figure 2.24.

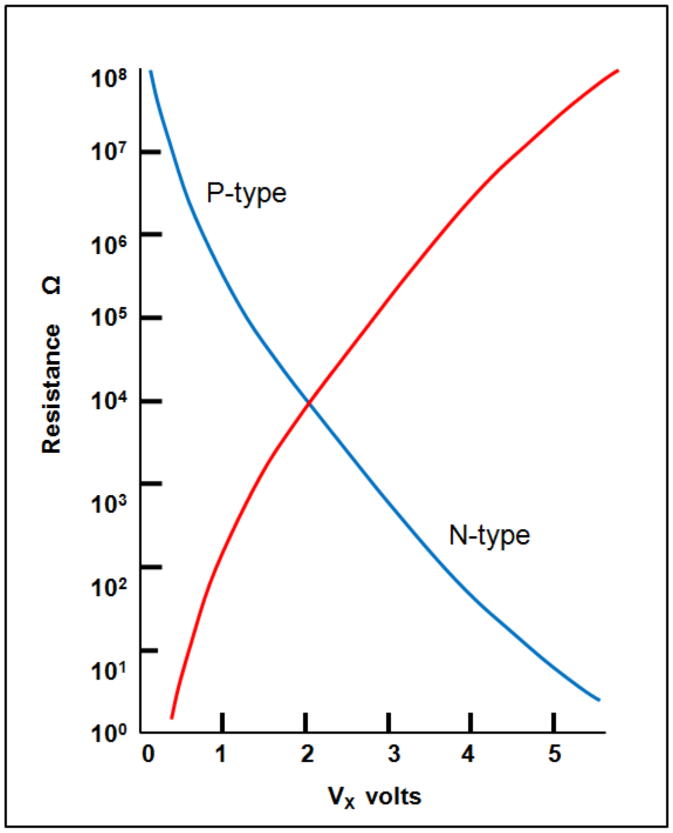


Figure 2.24 Plot of resistance versus the voltage, V_X for the two resistors, N-type and P-type.

Before we have our big "Aha!" moment, let's take this one step further. Referring to figure 2.24, we can see that the change in resistance as a function of V_X is a pretty dramatic exponential. Notice that the graph is semi-logarithmic, with the resistance changing over 8 orders of magnitude while V_X changes from 0 to 5 volts.

In order to bring this part of the discussion to a close, let's calculate the voltage at point C as a function of V_X. Also, for convenience, let's make the battery voltage 5 volts.

After roughly estimating the resistance values from figure 2.24, we can calculate the value of V_C for several values of V_X . The table shown at the right shows how the voltage at point C, V_C , varies as a function of this strange controlling voltage, V_X . Notice how dramatically the voltage changes from 5 volts to zero volts as V_X is swept from 0 to 5 volts.

V_X	R_2 (N-type)	R_1(P-type)	V_C
0	100,000,000	0	5
1	500,000	200	4.99
2	10,000	10,000	2.5
3	900	200,000	.02
4	80	500,000	.001
5	8	30,000,000	0

Table 1 Voltage at point C versus control voltage, V_X

This is exactly the behavior that we desire. The output voltage changes very rapidly as the input voltage sweeps through its range of values. Now, finally, we can bring our discussion back to the CMOS gate design.

Consider the mysterious resistor, R_2 , that we are calling an "N-type resistor." Actually, it is an N-type MOSFET transistor (I bet that you knew this all the time.) Let's look at this transistor in a bit more detail. Figure 2.25 shows a schematic representation of the N-type MOSFET transistor along with a plot of its behavior.

MOSFET transistors are called ***three-terminal devices***. Referring to figure 2.25 you can see the MOSFET has three terminals: a source (s), a gate (g), and a drain (d). In a three-terminal device, one of the terminals typically controls the behavior of the other two. In the MOSFET, the resistance between the drain and the source is controlled by the voltage on the gate, measured relative to the voltage on the source. In figure 2.25 this is V_{gs} .

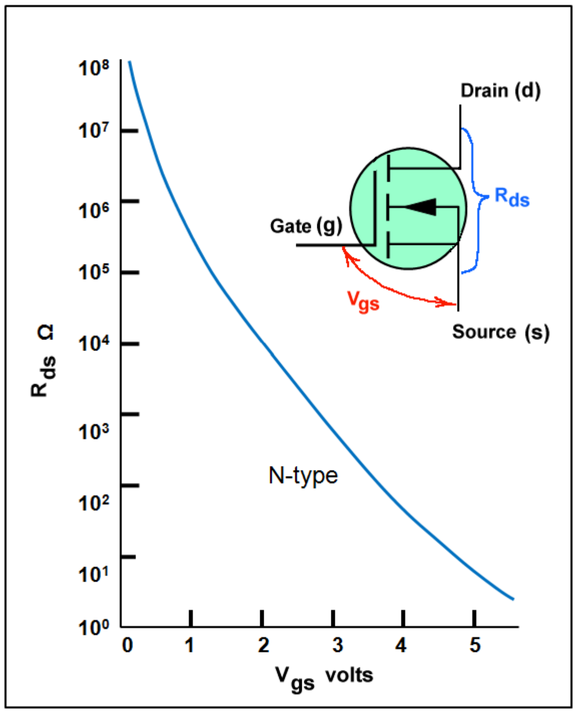


Figure 2.25 Characteristics of an N-type MOSFET transistor

For example, suppose an N-type MOSFET is in a circuit of some kind. Suppose we measure the voltage of the source relative to ground and the source voltage is 4.0 V above ground. Next, we measure the voltage of the gate relative to ground and this voltage is 5.0 volts. In this example,

$$V_{gs} = V_g - V_s = V_{gs} = 1.0 \text{ V}$$

If the graph of figure 2.25 looks familiar to you, it's because it is the same curve for the N-type resistor of figure 2.24, except I changed the legends on the X and Y axes because we want convert our example of special resistors to real circuit elements. Thus, we can now see that as we raise the voltage of the gate relative to the voltage of the source, the resistance between the source and the drain, R_{ds} , varies from over 100 MΩ to under 10 Ω as V_{gs} goes from 0 to 5 volts.

What about the P-type MOSFET? The behavior of the P-type is complementary to that of the N-type, but it is not the same. In this context, complementary means that the behavior is very similar, except that the polarity of the voltages are reversed. To see this, let's consider figure 2.26.

Here' what we reversed for the P-type MOSFET transistor. Comparing figures 2.25 and 2.26 we can see that the P-type MOSFET is "upside down" when compared to the N-type MOSFET. For the N-type, we connect the drain to the more positive voltage relative to the drain. For the P-type, we connect the source to a less positive voltage than the source. However, we measure V_{gs} in exactly the same way.

Let's do exactly the same sample calculation for V_{gs} that we did just above. However, now the source is connected to 5 volts relative to ground. We measure the voltages in exactly the same way. Thus,

$$V_{gs} = V_g - V_s = 4.0 - 5.0 = -1.0 \text{ V}$$

We can see this behavior on our graph in figure 2.26. As the voltage on the gate, V_{gs} , becomes more negative relative to the voltage on the source, V_s , the resistance that we measure between the source and the drain, R_{ds} , goes down. Note that resistance is a positive quantity, it does not become negative when we flip the polarities.

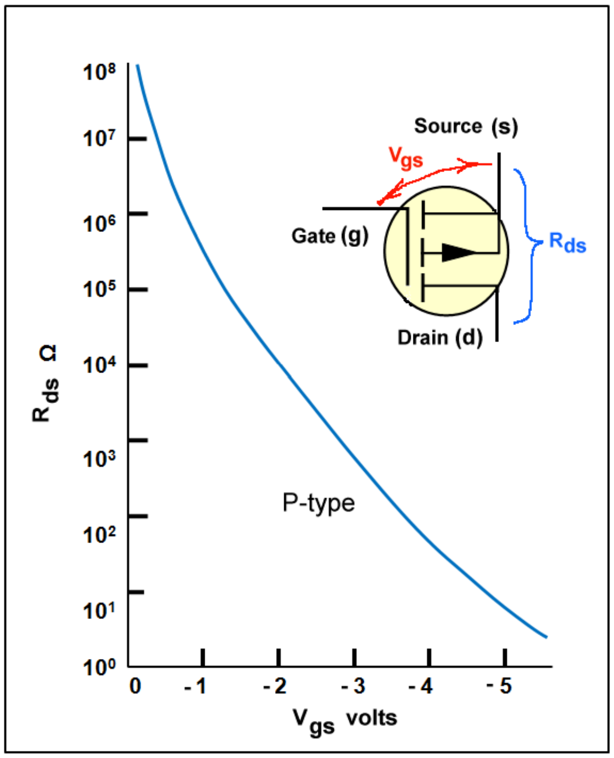


Figure 2.26 Characteristics of a P-type MOSFET transistor

Now (finally!), turn back the pages and re-examine figure 2.13. You can see how the gates of both MOSFETs are tied together so both transistors see exactly the same gate voltage. However, the behavior of the transistors are complementary. After all, they are CMOS gates. As the gate voltage rises (becomes more positive) the resistance of the N-type transistor goes down and the resistance of the P-type transistor goes up. According to our voltage divider in figure 2.23 and from the data of Table 1, we see that the voltage on the output of the gate will go down.

If we start with the gate voltage close to 5 volts and lower it towards 0 volts, the resistance of the P-type MOSFET goes down and the resistance of the N-type goes up. Once again, according to figure 2.23 and Table 1, we see that the output voltage goes up.

We can summarize the behavior in Table 2. We have the logical behavior of the NOT gate, but now expressed in terms of the actual voltages rather than the logic levels 1 and 0.

Gate voltage	Output voltage
0V	5V
5V	0V

Table 2 CMOS NOT gate

Let's take all of this newfound knowledge of MOSFET transistors and CMOS gates and put it to the test. We'll design a 3-input NOR gate!

Recall that the OR function produces a TRUE output if any input is TRUE. The NOR gate should just give the opposite result. The output should be FALSE, if any input is TRUE. That tells us that we need to make sure if any input goes HIGH (+5V) that we need a low resistance path to ground. In other words, the N-type transistors should all be connected in parallel so any one of them can provide the desired effect.

Figure 2.27 is a schematic representation of a CMOS NOR gate. Transistor pairs N1- P1, N2-P2 and N3-P3 are the complementary pairs. Let's see how it works.

If inputs A,B and C are all at 0 volts, then P1, P2 and P3 will all be in the low resistance state (figure 2.26). N1, N2 and N3 will all be in the high resistance state (figure 2.25). The three low resistance are one arm of the voltage divider and the three high resistances are the other. The output is therefore 5 volts, or logic level 1.

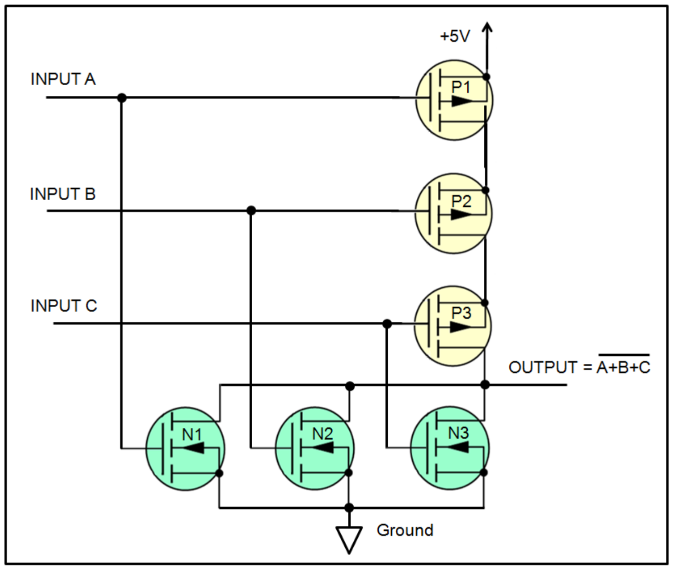


Figure 2.27 CMOS NOR gate

Now, if any of the inputs goes to logic 1, for example, input C, then N3 will go to the low resistance state and P3 will go to the high resistance state. The very low resistance of N3 will essentially be a short circuit around N1 and N2, so they really don't matter anymore. Likewise, the very high resistance of P3 will dominate P1 and P2 and that group will look like a high resistance. Therefore, the output will be very close to 0 volts, or logic 0.

I'll leave it to you to think about how we might change this design slightly to get a three-input OR gate. As a hint, think about what you might add to this circuit.

Before we leave this supplemental section I wanted to call your attention back to Table 1. Notice that when V_X = 2 volts, both R_1 and R_2 have a resistance of about 10,000 Ω . If we were to hold V_X at 2 volts, then a current of 5/20,000 = 250 microamperes (μA) would flow from the 5 volt power supply to ground. For the other values of V_X the sum of R_1 and R_2 is much greater, so far less current will flow between +5 volts and ground.

In fact, when V_X is either at 0 volts or +5 volts, the sum of R_1 and R_2 is tens to hundreds of megaohms, so virtually no current flows between +5V and ground, but it is in this narrow range around V_X = 2 volts that some small, but measurable current can flow from +5 volts to ground.

From our previous discussion, the power dissipated in R_1 and R_2 when 250 μA is flowing between +5 volts and ground is I^2R, or $(250^{-6})^2 * 20{,}000 = 1.25^{-3}$ watts, or 1.25 milliwatts (mW). Now, 1.25 mW may not seem like a lot of power to you, and frankly, it isn't. However, let's consider a modern microprocessor, such as the Pentium®. Its hundreds of millions of CMOS transistor pairs are busy switching billions of times each second and during that very narrow window of time when the CMOS pairs are switching between high resistance and low resistance, there is a fraction of time when tiny amount of current can flow and a small amount of power is dissipated in the transistor pair.

Consider figure 2.28. This exotic contraption is called a ***heat sink*** and it is designed to remove the 80 or so watts of heat generated by your Pentium dissipate it into the air. It functions in exactly the same way as the radiator in a car. Most heat sinks also have an integral fan to blow cooling air through the fins. The heat sink mounts to the top of the processor and thick *heat pipes* carry the heat from the processor to the radiator.

Were it not for the heat sink, your Pentium or Athlon would self-destruct in milliseconds. So, even though just milliwatts of power flow through the CMOS pairs for only picoseconds at a time, each of these little current bursts add up and we need significant mechanical cooling devices to keep the processors operating normally.

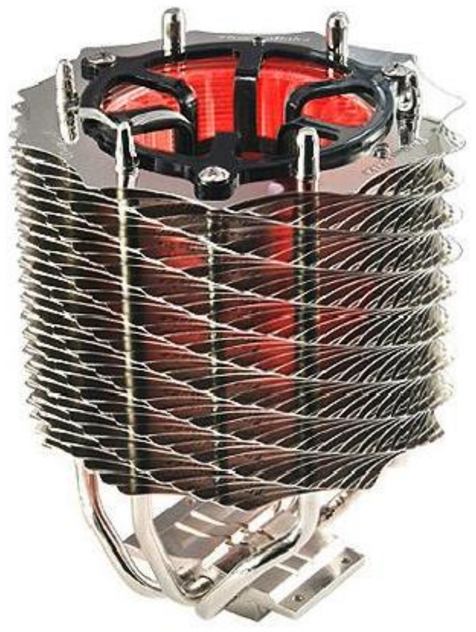


Figure 2.28 Heatsink for a modern Pentium® or Athlon® microprocessor.
Source: *http://www.xoxide.com/thermaltake-spinqvt-cpucooler.html*

Conversely, if we were to slow our processors way, way down so that there aren't too many times a second when both transistors are turned on, then we could easily run our computer with a few small batteries. However, don't expect to boot your operating system in less than a day or two.

Summary of Chapter Two

Here's what we've accomplished:

- Learned the basic logic gates, AND, OR and NOT and saw how more complex gates, such as NAND, NOR and XOR could be derived from them
- Learned that logic values that are dynamic, or change with time, may be represented on a graph of logic level or voltage versus time. This is called a waveform.
- Learned how CMOS logic gates are derived from electronic switching elements, MOSFET transistors.
- Learned how to describe the logical behavior of a gate or digital circuit as a ***Truth Table.***

Bibliography and References

1- http://www.dnaftb.org/dnaftb/20/concept/index.html
2- J. Watson, An Introduction to Field Effect Transistors, published by *Siliconix, Inc.*, Santa Clara, CA, 1970, pg. 91.
3- www.tomshardware.com
4-Fairchild Semiconductor Corp. Application Note 77, *CMOS, the ideal logic family***,** January, 1983

Exercises for Chapter 2

1- Consider the simple AND circuit of figure 1.14 and the OR circuit of figure 2.5. Change the sense of the logic so that an open switch (no current flowing) is TRUE and a closed switch is FALSE. Also, change the sense of the light bulb so that the output is TRUE if the light is not shining and FALSE if it is turned on. Under these new conditions of negative logic, what logic function does each circuit represent?

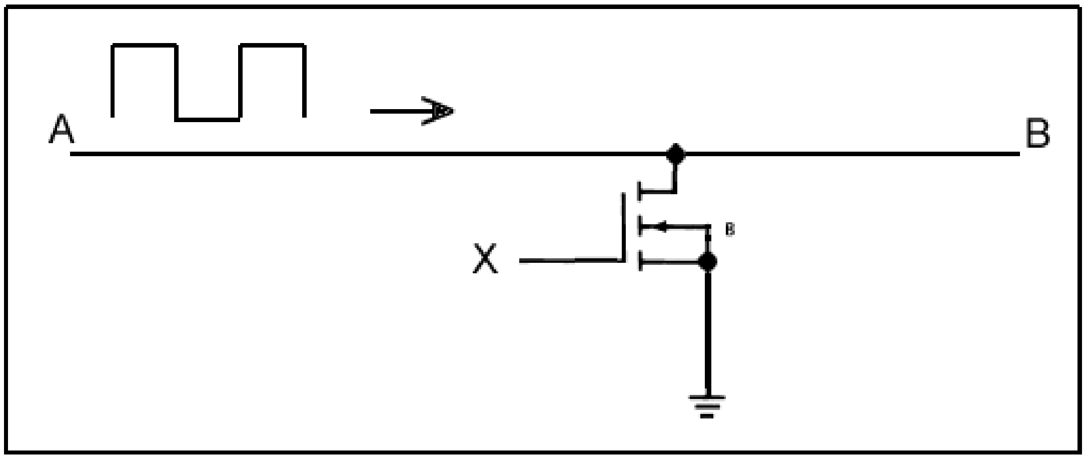
2- Draw the circuit for a 2-input AND gate using CMOS transistors. Hint: use the diagram in figure 2.16 as a starting point.

3- Construct the truth table for the following logical equations:

a- F = a*~b*~c + b*~a

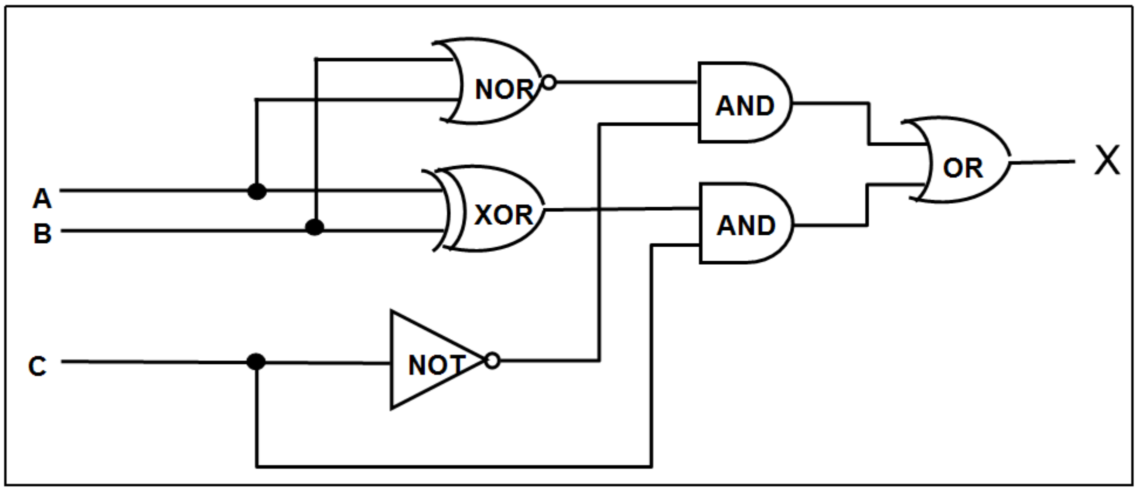
b- F = a*b + ~a*c + b*c

4- What is the effect on the signal between points A and B when the voltage at point X is raised to logic level 1?



5- Construct a truth table for a ***parity detection circuit***. The circuit takes as its input 4 variables, **a through d**, and has one output, **X**. Input **d** is a control input. If **d** = 1, then the circuit measures odd parity; if **d** = 0, the circuit measures even parity. Parity is measured on inputs **a, b and c.** Parity is odd if there are an odd number of 1's and parity is even if there is an even number of 1's. For example, if **d =** 1 and **(a,b,c)** = (1,0,0), then **X =** 1 because the parity is odd.

6- Draw the truth table that corresponds to the logic gate circuit shown below:



7- The gate circuit for the XOR function, equation **a ⊕ b = X**, is shown in figure 2.8. Given that you can also express the XOR function as:

a ⊕ b = ~[~ (~a * b) * ~(a * ~b)].

Redesign this circuit using only NAND gates.

8- Consider the circuit of figure 2.3. Suppose that we replace the AND gate shown in the figure with (a) an OR gate and (b) an XOR gate. What would the output waveform look like for the case where input A = 0 and the case where input A = 1?

9- Assume that you have three variables, A,B, and C defined as follows:

```
bool    A,B,C ;
```

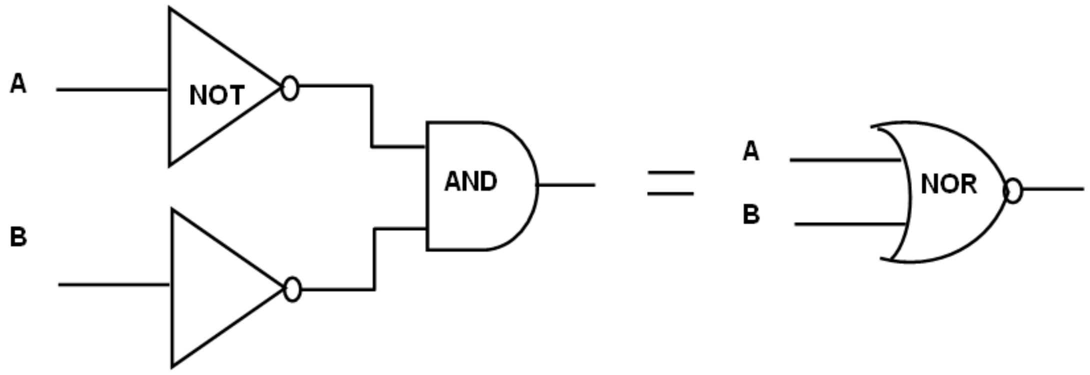
Consider the C++ control statement,

```
if ( A == false )
    C = B ;
else
    C = !B;
```

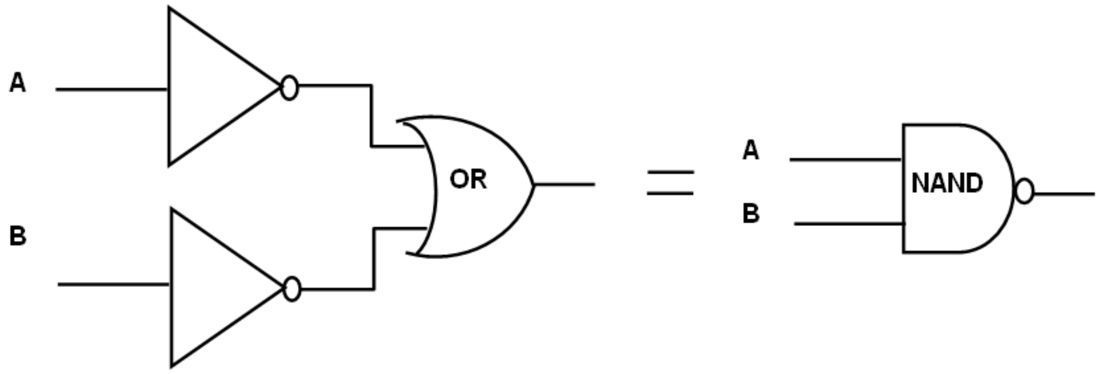
Draw the gate equivalent circuit for this C++ construct.

10- For the following two cases, prove that the two logic circuits shown are equivalent:

CASE1



CASE 2


A
B
OR
=
A
B
NAND