E-Z MATH

BOOLEAN ALGEBRA and LOGIC CIRCUITS

BY LOUIS E. FRENZEL, JR.

Don't let logic gates bar your ability to experiment with circuits

If you have followed this math series, you know that we've spent a lot of time covering the math related to basic electrical principles and electronic fundamentals. In this month's installment, we'll head out in another direction for a change of pace. We will discuss a type of math used with digital-logic circuits. That math is known as *Boolean algebra*.

What's Boolean for? Boolean algebra is a collection of simple mathematical procedures used to represent and express the logical operations that go on in a digital circuit. Boolean algebra is very similar to standard algebra. The primary difference is that unlike standard algebra, in which variables can be any value, in Boolean algebra only the values 0 and 1 are recognized. Besides that, most of the basic rules of working with algebraic expressions apply.

The big benefit of Boolean algebra is that it provides a way to express digital-logic operations mathematically. Boolean equations can be written to precisely describe how a logic circuit operates, which can help you to design such circuits. Boolean algebra also provides a way to minimize the number of gates needed in a logic circuit to simplify circuit design. That lowers overall cost, and can help reduce power consumption.

Also, the equations can show at a glance what is going on in a logic circuit to aid you in troubleshooting.

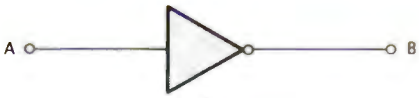
As I've said in previous articles, don't let terms like "Boolean," "equation," "mathematical expression," or "algebra" scare you. Once you learn the jargon and the few simple fundamentals presented here, even complex circuits will be easy for you. So, get ready for a digital-logic refresher, then we will have some fun writing the Boolean equations of a circuit and creating a circuit from the equations.

Review of Digital-Logic Circuits. At one time or another, you probably learned how basic logic circuits work. If not, the following brief summary will bring you up-to-date. The review is also for those of you who need a refresher.

The three basic logic gates are the inverter, AND gate, and OR gate. Two other widely used gates—the NAND and the NOR—are often derived from those basic gate circuits. All of the gate circuits process binary numbers made up of 0's and 1's. Binary 0 and binary 1 are represented by voltage levels. For example, a binary 0 may be indicated with zero volts (ground), while a binary 1 may be indicated by +5 volts.

The Inverter. An inverter is a logic element with a single input and a single output. As its name implies, it inverts an input signal. A binary-0 input produces a 1 output. A 1 input generates a 0 output. The inverter always produces an output that is the complement of the input. Complement here means opposite or reverse. You will also hear the inverter referred to as a NOT gate.

The logic symbol for an inverter is shown in Fig. 1. The triangle represents a buffer—a circuit that directly passes a binary digit onto the next circuit without changing the value. The circle at the output indicates inversion. So the digit passes through the buffer and is inverted at the gate's output. Note that the input and output are labelled with letters. All logic signals are given a name or designation. Here A is the input and B is the output.



A

INPUT	OUTPUT
A	B
0	1
1	0

B

Fig. 1. The simple inverter, A, is shown here with its little four-entry truth table.

Also shown in Fig. 1 is a table that shows all possible combinations of inputs and outputs. The input, A, can be either a 0 or 1. The table shows the state of the output, B, for each input state. Such a table is called a truth table. Truth tables are used to show what's going on inside a logic circuit.

AND Gate. An AND gate is a logic circuit with two or more inputs and a single output. The output is a binary 1 if all inputs are binary 1. Otherwise, the output is binary 0. The AND gate is often called a coincidence circuit because the output will be binary 1 only when all inputs are simultaneously all binary 1.

The logic symbol for a two-input AND

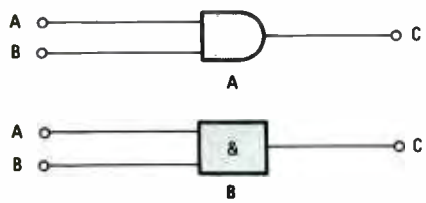
gate is shown in Fig. 2A. The inputs are A and B; the output is C. The shape of the symbol designates its function. An alternate symbol is given in Fig. 2B. The box designates the circuit while the ampersand (&) indicates the gate's function.

The truth table for a two-input AND gate is shown in Fig. 2C. There are always 2^N possible input combinations, where N is the number of inputs. With two inputs, there are:

$$2^2 = 4$$

different combinations. They are listed in the truth table along with the resulting outputs. Note that the only time the output (C) is 1, is when both inputs are 1.

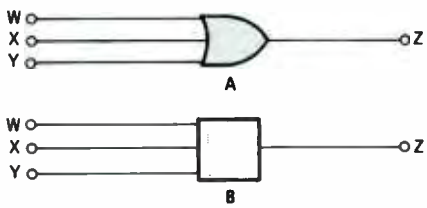
Keep in mind that an AND gate may have more than two inputs. Integrated-circuit AND gates typically have 2, 3, 4, 5, 8, or 13 inputs.



INPUTS		OUTPUT
A	B	C
0	0	0
0	1	0
1	0	0
1	1	1

C

Fig. 2. This two-input AND gate, A, can be drawn as shown in B. The truth table for all its possible states is shown in C.



INPUTS			OUTPUT
W	X	Y	Z
0	0	0	0
0	0	1	1
0	1	0	1
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	1
1	1	1	1

C

Fig. 3. For a change, this OR gate, A, is shown with three inputs instead of two. An alternative symbol is shown in B, while the elements truth table is shown in C.

OR Gate. An OR gate is also a logic circuit with two or more inputs and a single output. Its output is a binary 1 if at least one of its inputs is binary 1. Otherwise, the output is binary 0.

The logic symbols and truth table for an OR gate are given in Fig. 3. Note that the "equal to or greater than 1" designation means the OR function. The truth table shows the output Z with the inputs W, X, and Y. With three inputs, there are:

$$2^N = 2^3 = 8$$

possible input combinations. As with AND gates, IC OR gates typically come with 2, 3, 4, 5, 8, or 13 inputs.

A NAND Gate. A NAND gate is the combination of an AND gate and an inverter. It is often referred to as a NOT-AND circuit, and thus its name N-AND. The output is binary 0 only when all inputs are binary 1. For other input conditions, the output is binary 1.

A NAND can be drawn as an AND with an inverter (NOT) circuit, as Fig. 4A shows. However, the special symbol in Fig. 4B is normally used. The circle at the output indicates inversion. An alternate symbol is given in Fig. 4C. Here the triangle or half arrow on the output indicates inversion. The truth table indicates all possible inputs and the corresponding output states. Looking back at the truth table for the AND gate, you can see that a NAND output is its complement. NAND gates with 2, 3, 4, 5, 8, and 13 inputs are available in IC form.

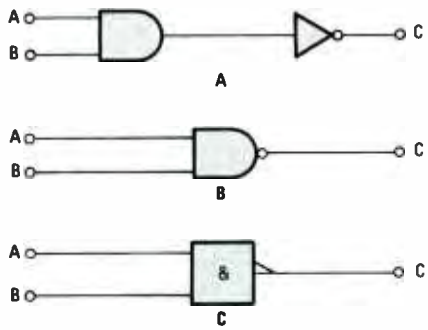
NOR Gate. The NOR gate or NOT-OR circuit is an OR gate followed by an inverter. The output is binary 0 if at least one of the inputs is binary 1. Otherwise, the output is binary 1.

The NOT-OR circuit, shown in Fig. 5A, clearly illustrates the circuit's function, but usually one of the symbols in Fig. 5B or 5C is more often used. The truth table shows the possible input and output states. IC NOR gates are available with 2, 3, 4, 5, 8, and 13 inputs.

Expressing Logic Mathematically. To begin using Boolean algebra, we need to find some way to express the basic logic operations using mathematical expressions. Let's take a look at ways of expressing inversion, AND, OR , NAND, and NOR operations.

As you learn the basic rules, keep in mind that the binary signals to be processed by the logic circuits are known as variables. Variables are signals that can change value. Binary variables can have one of two values; those values are 0 and 1.

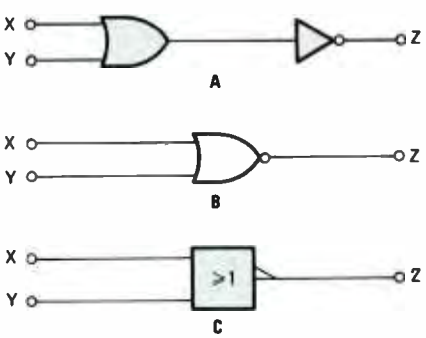
Variables are usually given names to distinguish them from one another. Letters of the alphabet are the most common, although numerous other alpha or alphanumeric names are also used. Usually signals are given some variable name (mnemonic) that is simply a shorthand way of referring to the



INPUTS		OUTPUT
A	B	C
0	0	1
0	1	1
1	0	1
1	1	0

D

Fig. 4. A NAND gate is nothing more than an inverted AND (B). Its output is the compliment of an AND gate's (C).



INPUTS		OUTPUT
X	Y	Z
0	0	1
0	1	0
1	0	0
1	1	0

D

Fig. 5. A NOR gate is nothing more than an inverted OR (B). Its output is the compliment of an OR gate's (C).

signal. An example is a binary signal called "clear," which might be represented by the mnemonic CLR. Many times binary signals are grouped together and related as in a binary number. For example, the bits in an 8-bit word might be given the names A0 through A7. In any case, you will see many different variations.

Inversion. Inversion is expressed mathematically by placing a bar over the variable. In Fig. 6, the input of the inverter is A while the output is B. Note that B is expressed in terms of A. That

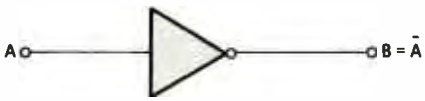


Fig. 6. The complement of a variable can be represented by placing a bar over that variable as shown here.

expression is read B is equal to NOT A. The NOT bar indicates that signal A has been inverted. Remember that A can be either a binary 0 or a binary 1. NOT A, of course, is the opposite, or complement.

Since it is difficult to type a bar over a letter as shown in Fig. 6, other simpler methods have been devised for representing inversion. Sometimes the inverted variable is indicated by an asterisk or a prime (similar to an accent). Using the variables in Fig. 6:

$$B = A^* \text{ or } B = A'$$

AND Function. The logical AND operation is indicated by placing a dot between the two variables to be ANDed. That is illustrated in Fig. 7. The two inputs to the AND gate are A and B

Fig. 7. ANDing of variables is indicated by using a dot between them.

while the output is designated C. Look at this expression for the output:

$$C = AB$$

In regular algebra AB would mean multiply A and B together. That's why the output of an AND gate is often called the product of the inputs. As in regular algebra, it is not necessary to show any symbol between the two variables (although sometimes a dot is used). Instead, they are simply just written adjacent to one another.

Figure 8 shows a four-input AND gate with different input variables. Many times you will see the output expression written with some variables separated by parentheses. Each input term appears within a set of parentheses to keep them visually separated to avoid confusion. But since each expression is written directly adjacent to the next, it

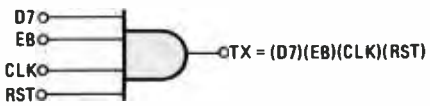


Fig. 8. The variables in Boolean algebra need not be one letter in length, but for clarity, separating them with parentheses becomes necessary.

means that the variables are ANDed together. In Fig. 8, we say that the output product is:

$$TX = (D7)(EB)(CLK)(RST)$$

OR Function. The logical OR is indicated by placing a plus sign between the variables. That is illustrated with the three-input OR gate shown in Fig. 9.

Fig. 9. ORing of variables is indicated with plus signs. Note the three inputs.

Often you will hear the output of an OR gate referred to as the sum of the input variables.

NAND Function. The NAND or NOT-AND function is simply the inverted product of the input variables. An example is shown in Fig. 10. The output expression is written just as it would be for an AND gate, but with a NOT indication given to the entire expression. That can be done by putting a bar over the entire expression as shown in Fig. 10. Alternately, the ANDed input terms can be put into parentheses and an asterisk or

Fig. 10. In a NAND expression, the result of all ANDing is simply inverted.

apostrophe used to indicate the NOT of the function. Note that the B term has a NOT bar over it.

The NOR Function. To produce the NOR function, we simply invert a basic OR output. Figure 11 shows a four-input NOR gate. The output expression is formed by simply writing the input variables separated by plus signs. Then, a bar is placed over the entire expression to invert it. Again note that one term, DZ, is inverted at the input.

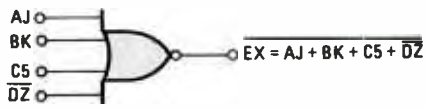


Fig. 11. Multiple-input NANDs do not need to have their variables separated by parentheses for clarity.

Now using those basic (Boolean) expressions for each of the logic gates, more complex circuits can be easily represented.

Deriving Boolean Expressions. Knowing the basic rules outlined in the previous section, you can now derive a complete Boolean expression for any larger, more complex logic circuit. The process is simply to work your way through the various logic gates starting with the inputs and building the equation a step at a time. A couple of examples will illustrate the process.

Refer to the circuit in Fig. 12. Note that the input variables are labelled. The output is designated G. Our job is to write the expression for G in terms of the input variables. It's really not as complicated as it sounds.

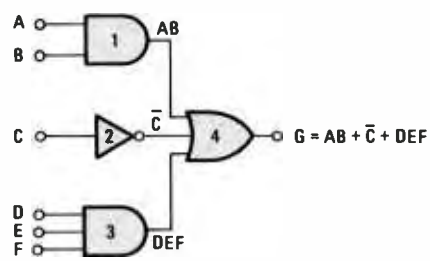


Fig. 12. You end up with a sum of products expression for this circuit after analysis.

To begin, you start with the variables at the inputs to each of the circuits on the left. Write the expression for the output of each circuit. For example, the output of AND-gate 1 is simply AB. The output of the inverter 2 is NOT C. The output of AND-gate 3 is DEF.

The outputs of gates 1 and 3, and inverter 2, form the inputs to OR-gate 4. To complete the expression, simply OR together each of the inputs to gate 4. The output expression G then becomes:

$$AB + \overline{C} + DEF$$

Take a look at the expression we just derived. You often hear an expression like that referred to as a sum of products. In this case, the products are the ANDed variables AB and DEF. The sum, of course, refers to the ORing together of each of the products.

A slightly more complex circuit is shown in Fig. 13. Still the evaluation process is the same. Work your way through the circuit from left to right writing the output expression for each gate. The output of gate 1 is A1(K) as shown. We use parentheses in this case to show the separation between the

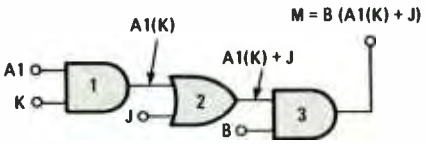


Fig. 13. The output of one gate becomes the input of the next in this circuit.

two variables, yet they are written adjacent to one another to indicate a product or AND function.

Next, the output of gate 1 is ORed with the input of J. The resulting output from gate 2 is:

$$A1(K) + J$$

That becomes one of the inputs to AND-gate 3. That expression is ANDed with input B to produce the final output expression:

$$M = B(A1(K) + J)$$

Again parentheses are used to keep the variables separated and to ensure the correct logical operation is expressed.

Take a look at the example in Fig. 14. Again, the procedure is to develop the output expressions of the input gates,

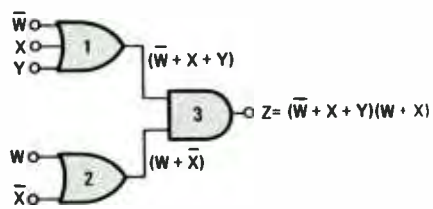


Fig. 14. You end up with a product of sums expression for this circuit after analysis.

then work your way from left to right to create the output. The output from gate 1 is:

$$(\overline{W} + X + Y)$$

The output of gate 2 is:

$$(W + \overline{X})$$

Those two outputs become the inputs to AND-gate 3. We create the final output expression, Z, by simply ANDing together the two expressions. The result is:

$$Z = (\overline{W} + X + Y)(W + \overline{X})$$

You might hear that kind of expression called a product of sums.

Generating a Circuit From Equations. Now let's consider the process of drawing the logic circuit corresponding to a given Boolean expression. Let's start with the simple expression below:

$$W = XY + \overline{Z}$$

The various logic functions implied by the equation are pretty easy to spot. The X and Y are written adjacent to one another indicating that the two signals are ANDed. Simply draw an AND gate with X and Y as the input. The output of that AND gate XY is then going to be ORed with another input called $\overline{Z}$. The plus sign tells us we need an OR gate to do that. If only the variable Z is available, an inverter is needed to produce $\overline{Z}$. The resulting circuit is shown in Fig. 15.

A slightly more complex example is given below:

$$X = (A + B + \overline{C})(\overline{D} + E)(F)$$

The parentheses tell you that you have three different groups of variables ANDed together to form the output, X. The variables in the groups are ORed

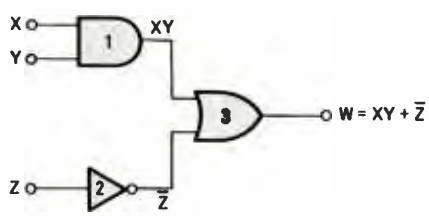


Fig. 15. By drawing the logic symbols that correspond to the Boolean expressions you'll arrive at the correct circuit.

together. You can start by creating the circuits for each group of variables. The plus signs inside the parentheses indicate an OR gate should be drawn. To start you can draw an OR gate with inputs A, B, and $\overline{C}$. Another expression is derived by ORing the input variables $\overline{D}$ and E. Simply draw an OR gate with the two variables as the inputs. The variable F inside parenthesis will be ANDed together with the other two expressions. Finally, to complete the circuit simply draw an AND gate with three inputs and connect them to the outputs of the two OR gates and a source of signal F. See Fig. 16.

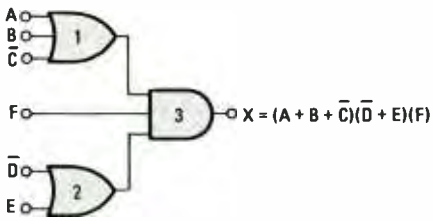


Fig. 16. The product of sums expression shown was used to generate this circuit.

Exercise problems. Here are a couple of problems for you to practice on.

1. Write the output expression of the circuit shown in Fig. 17.

2. Draw the logic diagram corresponding to the expression:

$$M = (\overline{F} + G + H)(J + \overline{K} + L)$$

Assume no inverted signals are available.

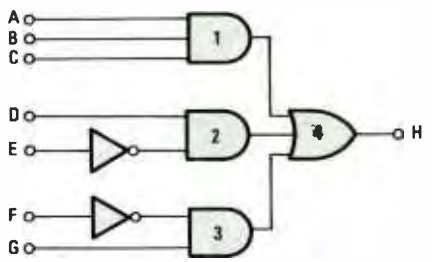
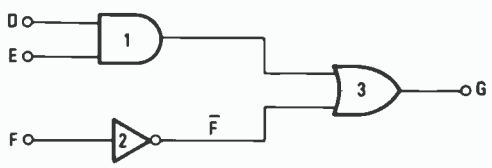


Fig. 17. Write the equation for the circuit.

Truth Tables. You have already seen how truth tables are used to define all possible combinations of inputs and outputs for the various logic elements. Truth tables, however, can also be used to describe larger, more complex logic circuits. The nice thing about a truth table is that it gives you a complete picture of what's going on in the circuit for any set of input states.

Developing a truth table for any logic circuit is relatively easy. All you have to do is write out all the possible input states, and for each one compute the output state for every gate in the circuit until the final output is derived. Let's take a couple of simple examples to show how you can evaluate the output state for a given set of inputs.

Take a look at the circuit shown in Fig. 18A. Where N is the number of in-



A

INPUTS			OUTPUTS		
D	E	F	GATE 1 DE	INVERTER 2 F̄	GATE 3 G
0	0	0	0	1	1
0	0	1	0	0	0
0	1	0	0	1	1
0	1	1	0	0	0
1	0	0	0	1	1
1	0	1	0	0	0
1	1	0	1	1	1
1	1	1	1	0	1

B

Fig. 18. The possible outputs for circuit A can be displayed in a truth table like B.

puts, the total number of different input states is 2^N. The circuit shown has three inputs, so with three inputs, there are:

$$8 = 2^3$$

Those eight possible combinations are the binary numbers 000 (decimal 0) through 111 (decimal 7). Therefore, we will make a truth table with eight possible input states as shown in Fig. 18B.

The remainder of the truth table will contain the outputs at each element in the circuit. For example, note that we have the output of AND gate 1, the output from inverter 2, and the output from OR gate 3. Knowing how each of the logic gates work, you can then determine the output of each gate given the various combinations of inputs, and record those values in the table. For example, the input to gate 1 is D and E. Since it is an AND gate, the only time it will produce a binary-1 output is when both D and E are binary 1's. Simply locate those states in the inputs and record binary 1's beside them. All of the other entries in the DE column will be binary 0. The F̄ column is created by simply inverting the F column.

You now know both inputs to OR-gate 3. The DE and F̄ columns can then be ORed together to produce the final output, G. Again, remembering that an OR gate produces a binary-1 output if either or both of its inputs are binary 1, you can complete the G column.

Be sure you go through the circuit and the truth table carefully so that you understand exactly what is going on in each column.

Let's take one more example to be sure you know how to develop the truth table from a given logic circuit. Refer to Fig. 19A. That circuit has four different inputs, therefore, it will have:

$$2^4 = 16$$

possible input combinations. Those are the four-bit binary numbers 0000 (decimal 0) through 1111 (decimal 15). They are illustrated in the truth table shown in Fig. 19B.

The remaining columns in the truth table are the output of gate 1 (A + B); the output of gate 2 (C + D); and the final output, F. Again, using the input states, develop the output for gate 1 and then gate 2. Those are OR gates, and so produce a binary-1 output when either or both inputs are binary 1. For gates 1 and 2 simply search through the table for those rows where binary 1's occur at the inputs of the gates and record binary 1's in the corresponding output column. Once you have done that for both gates, you will have the inputs to gate 3. Gate 3 is an AND gate, so its output is 1 when the output columns for gates 1 and 2 are both binary 1. Again look through all of the columns in the truth table to be sure you understand how they apply to the circuit.

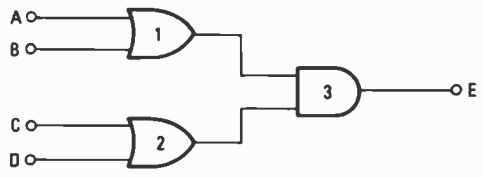
Exercise Problem. To see if you can do this yourself, try the following problem.

3. Draw the circuit for the Boolean expression:

$$Z = Y(VW + \bar{X} + \bar{V}X)$$

Assume only the inputs V, W, X and Y are available. Develop the truth table showing the outputs for all inverters and gates.

Writing from a Truth Table. In many cases, you will start with a truth table and develop the Boolean expression from it. That is what usually happens when you are designing a digital circuit. Typically, you will define a desired



A

INPUTS				OUTPUTS		
A	B	C	D	GATE 1 (A + B)	GATE 2 (C + D)	GATE 3 (E)
0	0	0	0	0	0	0
0	0	0	1	0	1	0
0	0	1	0	0	1	0
0	0	1	1	0	1	0
0	1	0	0	1	0	0
0	1	0	1	1	1	1
0	1	1	0	1	1	1
0	1	1	1	1	1	1
1	0	0	0	1	0	0
1	0	0	1	1	1	1
1	0	1	0	1	1	1
1	0	1	1	1	1	1
1	1	0	0	1	0	0
1	1	0	1	1	1	1
1	1	1	0	1	1	1
1	1	1	1	1	1	1

B

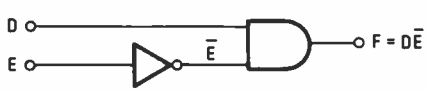
Fig. 19. You must use all possible input combinations for the circuit A for the table, B.

output condition that is generated when specific input states occur. To develop your design, you build a truth table filling in the columns with the desired output states for the given inputs. Then, the truth table can be used to help write the Boolean equation, and the logic circuit itself, can be deduced from the equation. Once the logic circuit is drawn, it can be implemented with ICs or other components.

A simple example of that is a design where we have two inputs and want a specific output to occur. For example, perhaps you want the output F to be binary 1 when input D is equal to 1 and input E is equal to 0. For all other input states, we want the output to be binary 0. That set of conditions can be drawn in a truth table as shown in Fig. 20A. With two inputs, there are four possible input combinations. We want the output to be a binary 1 when D is equal to 1

INPUTS		OUTPUT
D	E	F
0	0	0
0	1	0
1	0	1
1	1	0

A



B

Fig. 20. A truth table (A) must be generated from a circuit (B) before deriving the Boolean equation.

and E is equal to 0. All other input states produce a binary 0 output. The truth table shows that set of conditions.

Now to derive the Boolean expression from the truth table, we look at the output column F and note where binary 1's occur. Next, we look at the input states that produce that output. Then we write an expression that is the product of the input variables. For example, in the truth table of Fig. 20A, the equation becomes:

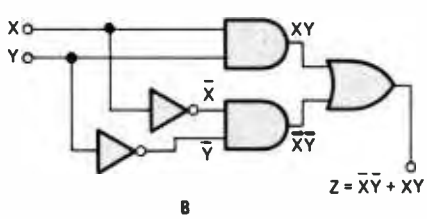
$$F = D\bar{E}$$

We write the D because a binary 1 appears in the D column. We write $\bar{E}$ because a zero exists in the E column. That simple equation, of course, can be implemented with a single two input AND gate. An inverter is be needed to produce $\bar{E}$ if only the E input is available. The resulting circuit appears in Fig. 20B.

Now let's take a more complex example. Suppose that we want to develop a simple circuit for comparing two bits. We would like the output of the circuit to be binary 1 when the two bits are equal, and binary 0 when they are different. That is described in the truth table shown in Fig. 21A. The two inputs are X and Y, therefore, the four possible input combinations are listed. We want the output Z to be binary 1 when the bits are alike. So we write a binary 1 when both bits are 0 and when both bits are 1. The remaining input states produce a binary 0 output.

INPUTS		OUTPUT	
X	Y	Z	
0	0	1	← $\bar{X}\bar{Y}$
0	1	0	
1	0	0	
1	1	1	← XY

A



B

Fig. 21. The truth table, A, generates a sum of products equation for circuit B.

Now we can write the equation for the circuit. We look at the output column and note the places where the binary 1's occur. Then we write an AND ed expression using the inputs. The first binary 1 output occurs if X = 0 and Y = 0. Therefore, the equation for that state is:

$$Z = \bar{X}\bar{Y}$$

The other binary 1 output occurs when X = 1 and Y = 1. Therefore, the input expression is:

$$Z = XY$$

To complete the Boolean expression, we simply OR the two AND expressions together. That is because the output becomes binary 1 under either condition. The resulting output expression:

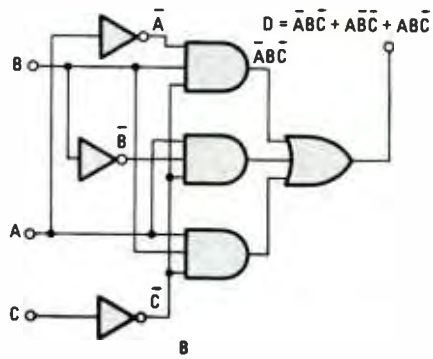
$$Z = \bar{X}\bar{Y} + XY$$

The resulting circuit is illustrated in Fig. 21B.

Let's take it one step further, and develop a more complex circuit. Suppose we have three inputs and the desired outputs are indicated by the binary 1's in the truth table of Fig. 22A. To develop the output expression for

INPUTS			OUTPUT	
A	B	C	D	
0	0	0	0	
0	0	1	0	
0	1	0	1	← $\bar{A}B\bar{C}$
0	1	1	0	
1	0	0	1	← $A\bar{B}\bar{C}$
1	0	1	0	
1	1	0	1	← $AB\bar{C}$
1	1	1	0	

A



B

Fig. 22. The conditions for a binary 1 output (A) must be ORed together to produce the Boolean equation (B).

the truth table, write an AND expression using the input variable for each place where a binary 1 appears in the output. The first AND expression is $\bar{A}B\bar{C}$. The variable with the NOT sign is used when a binary 0 appears at the input, and the variable itself is used when a binary-1 state occurs.

The other two conditions that produce a binary-1 output are $A\bar{B}\bar{C}$ and $AB\bar{C}$. Finally the output expression is built by ORing together the three input conditions that cause a binary 1 to appear:

$$D = \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C}$$

The corresponding circuit is shown in Fig. 22B.

That procedure works regardless of the number of inputs used. As the number of inputs increases, the Boolean expressions become far more complex. As it turns out, most of the larger more complex networks can be simplified by the use of Boolean rules. In the next installment, we will introduce the Boolean rules and show you ways to turn complex circuits into simpler ones.

But first, another exercise problem can be found on page 94. Why not turn there now to check your understanding. The answers to all of problems in this month's installment can be found there. ■

(Continued on page 94)

E-Z MATH

(Continued from page 79)

Exercise Problem. Try this yourself to be sure you understand the concepts presented.

4. A logic circuit has four inputs A, B, C, and D. Binary outputs occur when *any* three inputs are simultaneously binary 1, but not when all inputs are 1. Write the truth table, develop the Boolean output equation F, and draw the resulting circuit.

INPUTS				OUTPUT	
A	B	C	D	F	
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	0	
0	1	1	0	0	
0	1	1	1	1	$\bar{A}BCD$
1	0	0	0	0	
1	0	0	1	0	
1	0	1	0	0	
1	0	1	1	1	$A\bar{B}CD$
1	1	0	0	0	
1	1	0	1	1	$AB\bar{C}D$
1	1	1	0	1	$ABC\bar{D}$
1	1	1	1	0	

$F = \bar{A}BCD + A\bar{B}CD + AB\bar{C}D + ABC\bar{D}$

Answers to Exercise Problems

1. $H = ABC + D\bar{E} + \bar{F}G$

2. See Fig. 23.

3. See Fig. 24.

4. See Fig. 25.

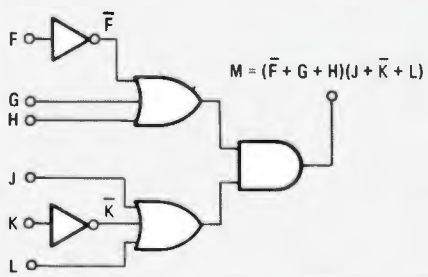


Fig. 23. Your solution to problem 2 should look like this, if not recheck your logic.

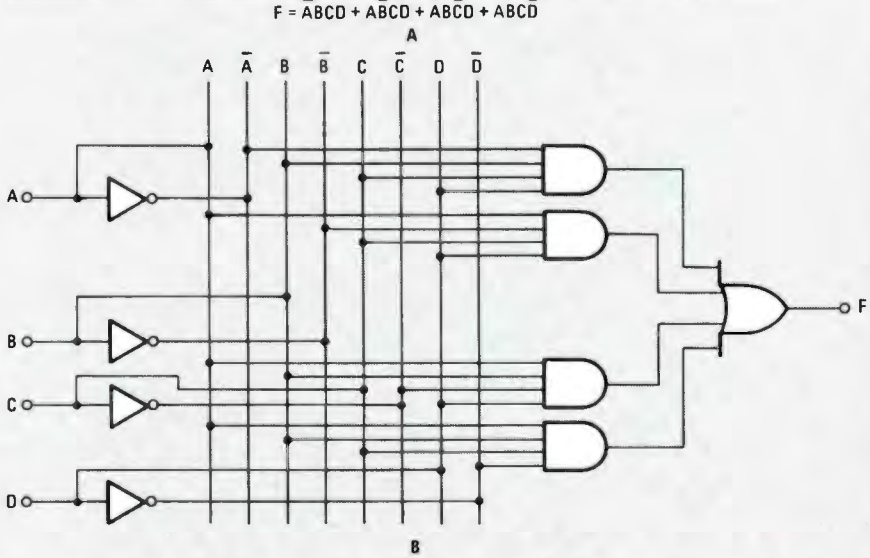
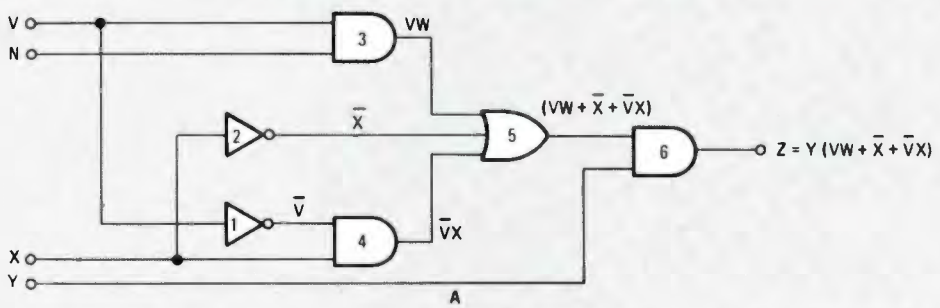


Fig. 25. When solving problem 4, you should've started with a truth table, generated an equation (A), and drawn the final circuit (B) as shown.



INPUTS				OUTPUTS					
V	W	X	Y	INVERTER 1 $\bar{V}$	INVERTER 2 $\bar{X}$	GATE 3 VW	GATE 4 $\bar{V}X$	GATE 5 $(VW + \bar{X} + \bar{V}X)$	GATE 6 Z
0	0	0	0	1	1	0	0	1	0
0	0	0	1	1	1	0	0	1	1
0	0	1	0	1	0	0	1	1	0
0	0	1	1	1	0	0	1	1	1
0	1	0	0	1	1	0	0	1	0
0	1	0	1	1	1	0	0	1	1
0	1	1	0	1	0	0	1	1	0
0	1	1	1	1	0	0	1	1	1
1	0	0	0	0	1	0	0	1	0
1	0	0	1	0	1	0	0	1	1
1	0	1	0	0	0	0	0	0	0
1	0	1	1	0	0	0	0	0	0
1	1	0	0	0	1	1	0	1	0
1	1	0	1	0	1	1	0	1	1
1	1	1	0	0	0	1	0	1	0
1	1	1	1	0	0	1	0	1	1

B

Fig. 24. Problem 3 should've tested your ability to generate the circuit in A and the table in B from the equation.