E-Z MATH

BOOLEAN LAWS and REDUCTION

BY LOUIS E. FRENZEL, JR.

If you know a few Boolean laws, you can greatly simplify any logic circuit.

In last month's column we introduced you to Boolean algebra and showed you how binary logic signals could be represented with mathematical expressions. We then showed you how to derive a Boolean expression from any logic circuit. You also saw how to draw the logic circuit corresponding to a given Boolean expression. Finally, we explained the value and use of truth tables. We showed you how to write a truth table for a logic circuit or derive the Boolean expression from a truth table. This month, we will provide you with more advanced information on Boolean algebra. Specifically, we will introduce you to the basic rules (laws) that will allow you to apply Boolean algebra to the design of logic circuits.

Boolean algebra is a system of mathematics that is used to express how digital-logic circuits operate. It is very much like conventional algebra and most of the standard algebra rules and procedures work with Boolean expressions. You just have to remember that the only numeric values in Boolean algebra are 0 and 1. When those rules are combined with the Boolean laws you will learn in this article, you will be able to analyze and minimize the most complex digital-logic circuits.

Why Use Boolean? The two main benefits of using Boolean algebra come from its ability to represent and simplify digital-logic circuits for the purpose of reducing the amount of circuitry required to implement a function. Expressing digital logic in mathematical terms provides us with a compact and convenient way of designing and analyzing digital circuits. The complete operation of a computer or any other digital circuit can be fully expressed with Boolean equations rather than large complex logic diagrams. While logic diagrams, of course, have their place, most original design and analysis is done mathematically.

Using Boolean algebra to alter Boolean expressions (and so the corresponding circuits) to reduce the number of gates and other components required to implement them, is its best application. Many times in designing a digital circuit, the truth table will first be developec based on the desired inputs and outputs. The Boolean equation is then derived from the truth table. At that point, the equation can be represented by logic gates. However, if the equation derived from the truth table is manipulated using Boolean expressions, it can be simplified (minimized). That simplification generally results in a smaller circuit with fewer parts. Any time that you can reduce the number of components or logic gates, considerable savings can be realized. That is particularly true of a complex digital system such as a computer. Reducing the number of logic gates reduces the production cost, size, and power consumption. The reliability is increased, and operating speed is usually improved. Keep those benefits in mind as you are learning the Boolean rules.

The rules of Boolean algebra are stated in terms of laws that were originally layed down by mathematician George Boole in his book "The Laws of Thought," published in 1847. Let's explain each law in detail.

Laws of Intersection. The laws of intersection relate to the use of AND gates. Remember the basic Boolean expression for an AND gate is:

$$C = AB$$

where A and B are the inputs and C is the output. A or B may be either 0 or 1. In an AND gate, the output is only 1 if both inputs are binary 1. The laws of intersection simply state what the output of an AND gate will be if one input is binary 0 while the other is A or if one input is binary 1 and the other is A.

The basic laws of intersection are:

$$A(0) = 0$$
$$A(1) = A$$

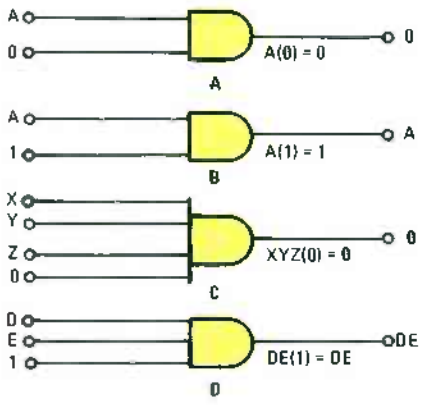


Fig. 1. Here we demonstrate the laws of intersection. In A, one input of the AND gate is 0, making the output 0. In B, a binary 1 is applied to an input, making the output simply A. Figures C and D demonstrate the same principle for multiple inputs.

We can illustrate those laws with AND-gate symbols as shown in Fig. 1. In Fig. 1A, one input of the AND gate is 0. The other input A can be either 0 or 1. Just remember that any time one input to an AND gate is 0, the output will always be 0 regardless of the state of A.

In Fig. 1B, a logic signal A is applied to one input of the AND gate while a binary 1 is applied to the other. Of course, A may be either 0 or 1. Under those conditions, the output is simply equal to the A input. If A is 0, the AND gate output will be 0. If A is 1, both inputs are 1, making the output 1. Use a truth table for the AND gate to verify both laws yourself.

What the laws of intersection are really stating are some special cases in the application of AND gates. If you know how an AND gate operates, those rules are pretty obvious. By remembering those special mathematical conditions, often a circuit can be simplified by applying them. Keep in mind that the rules also apply to AND gates with more than two inputs. Figures 1C and 1D are examples of that.

Laws of Union. The laws of union relate to the application of OR gates. Recall that the Boolean expression of an OR gate is:

$$C = A + B$$

Again, A and B are inputs that can be either 0 or 1. The laws of union simply state what the output will be if one input to an OR gate is either binary 0 or binary 1. The truth table in Fig. 2A illustrates this.

The laws of union are:

$$A + 1 = 1$$
$$A + 0 = A$$

In the circuit at Fig. 2A, one input to the OR gate is fixed at binary 1. The other input is A.

From your understanding of the operation of an OR gate, you can see that the output will always be binary 1 regardless of the state of A. The output of an OR gate is binary 1 if any one or all inputs to the OR gate are 1.

In Fig. 2B, one input to the OR gate is fixed at binary 0. As a result, the output will simply be a function of the input. If input A is 1, the output is 1. If input A is 0, the output will be 0. The laws of union are extremely easy to understand, particularly if you remember how an OR gate works. Again these basic expressions can often be applied to a Boolean expression for the purpose of

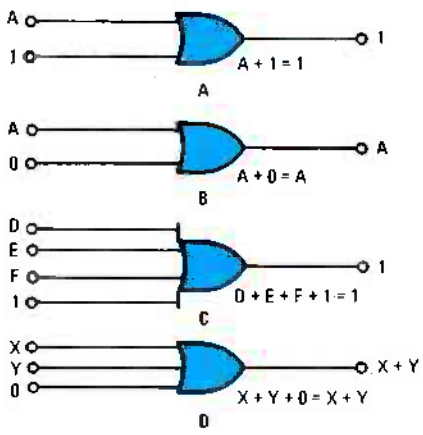


Fig. 2. In A, one input to the OR gate is fixed at binary 1, the other input is A, forcing the output to be binary 1. In B, one input is fixed at binary 0. As a result, the output will be a function of the input. The rule also applies to gates with more than two inputs, as in C and D.

changing its form or reducing its complexity. Don't forget that the rule also applies to OR gates with more than two inputs. Figures 2C and 2D are examples of that.

Laws of Tautology. The laws of tautology indicate the effect of redundant inputs on logic gates. There are two forms of the law, one for AND gates and one for OR gates. They are:

$$AA = A$$
$$A + A = A$$

The circuits for those expressions are given in Figs. 3A and 3C. The laws of

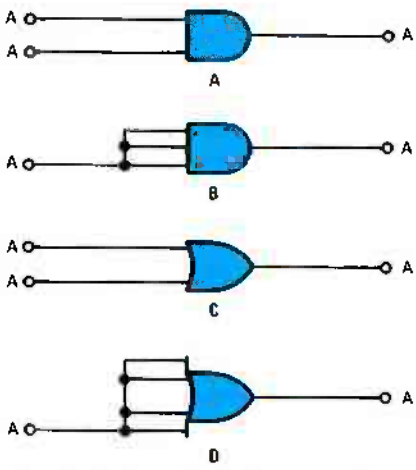


Fig. 3. The laws of tautology state that when all of the same inputs are applied to a non-inverting logic gate, the output will be the same as the input. The circuits for such expressions are given in A and C. For multiple-input gates, such as in B and D, the same holds true.

tautology state that when all of the same inputs are applied to a non-inverting logic gate, the output will simply be the same as the input. You can assume values of 0 and 1 for A on either circuit and determine that in every case the output will be A.

Suppose you had the Boolean expression:

$$Z = XXJ$$

you can generate that logic with the gate in Fig. 4A. By the laws of tautology, you can replace XX with X so the new expression is $Z = XJ$. Now you can implement the circuit with a 2-input AND gate as they are equivalent (see Fig. 4B).

Another example is shown in Fig. 4C for the relation:

$$M = D + D + R$$

Since $D + D = D$, the expression becomes $M = D + R$. So a 2-input gate can be used as Fig. 4D shows.

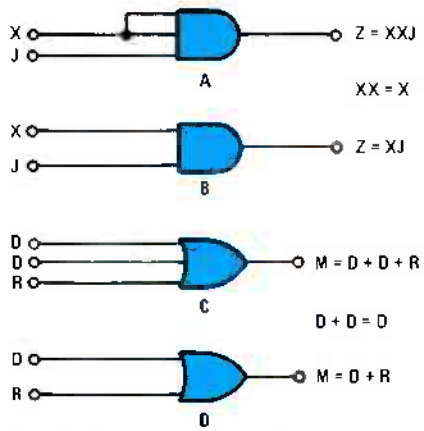


Fig. 4. By the laws of tautology, you can replace the XX in A with just X, so the new expression can be implemented with a 2-input gate as in B. The same is true for the OR gate in C, which is transformed to the one in D.

The Law of Complements. The law of complements tells what the output of a logic circuit is when both a logic signal and its complement are applied to a gate. There are two versions of the rule, one for AND gates and one for OR gates. Those are:

$$A\bar{A} = 0$$
$$A + \bar{A} = 1$$

Figure 5 illustrates those rules with gates. In Fig. 5A, you can see that if you apply a signal and its complement to the AND gate, one of those signals will always be a binary 0. Therefore, the output will always be 0.

The opposite condition occurs in an OR gate (see Fig. 5B). If a signal and its complement both are applied simultaneously, one input will always be a binary 1. That, of course, will cause the output to be binary 1. Again, those laws apply even if the gate has more inputs. See Figs. 5C and 5D for examples.

The Law of the Double Negative. The law of the double negative states that if a binary signal is inverted twice, its value will not be changed, as follows:

$$\overline{(\overline{A})} = A$$

If we invert or complement a logic signal, we reverse its value. A 0 will become a 1 and a 1 will become a 0 (see Figs. 6A and 6B. If we complement (invert) again, the first inversion is undone and we end up with a signal at the output that is the same as the input. That is illustrated in Figures 6C and 6D. One complement simply cancels the other.

The Laws of Commutation. The laws of commutation state that you can write the variables in a Boolean expression in any order and the meaning or effect will be the same. The two basic variations of the law are:

$$AB = BA$$
$$D + E = E + D$$

You can see that it doesn't matter what order the variables are in. What that means is that it does not matter

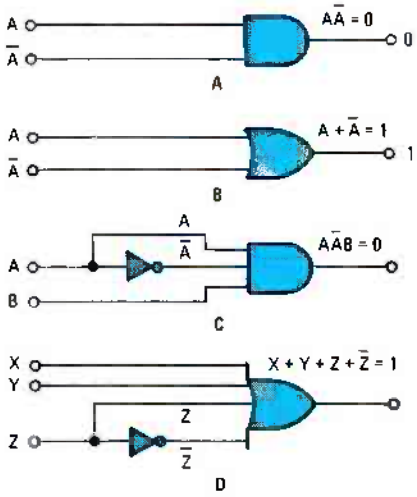


Fig. 5. In A you can see that if you apply a signal and its complement to the AND gate, one of those signals will be a binary 0, always making the output 0. The opposite occurs in an OR gate (B). Those laws apply even if the gate has more inputs as in C and D.

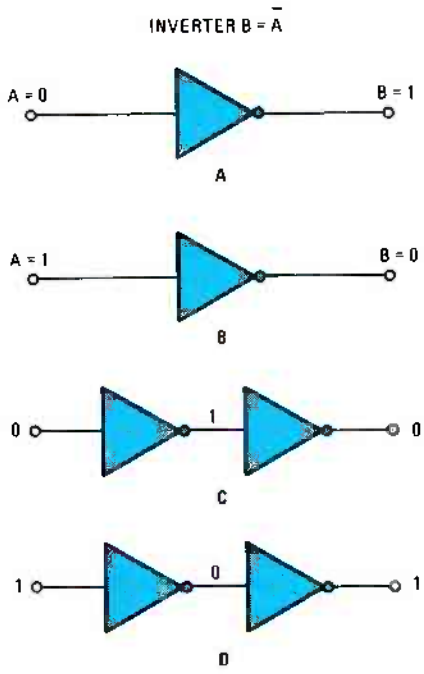


Fig. 6. If we invert or complement a logic signal as in A and B, we reverse its value. If we do it twice, the first inversion is undone and the output is the same as the input (see C and D).

which input of an AND gate or OR gate you send a variable's value to. The logical function is the same. As we have indicated before, that rule also applies to expressions containing three or more input variables. More typical examples are:

$$ABC = ACB = BAC = BCA = CAB = CBA$$
$$D + E + F = D + F + E = E + D + F =$$
$$E + F + D = F + D + E = F + E + D$$

The Laws of Distribution. The two forms of the law of distribution are:

$$A(B + C) = AB + AC$$
$$(A + B)(A + C) = A + BC$$

Those rules are similar to the standard algebra rules of factoring and expanding algebraic expressions. In the first version of the law of distribution, A is multiplied by the expression B + C. In ordinary algebra, we would multiply each term inside the parenthesis by A and add them together. The result being:

$$A(B + C) = AB + AC$$

You can also work the expression in the opposite direction. For example, starting with the expression:

$$AB + AC$$

you can see that A is common to both terms. Therefore, you could factor it out and simplify the expression. The result, of course, is simply:

$$A(B + C)$$

As you work with various Boolean expressions to simplify them or to change their form, you will regularly be either factoring or expanding them. Those two techniques will greatly speed up and simplify the process.

The other form of the laws of distribution is somewhat more complex. Take a look at this expression again:

$$(A + B)(A + C) \text{ Product-of-Sums} =$$
$$A + BC \text{ Sum-of-Products}$$

It says that a product-of-sums can be simplified into a sum-of-products. Fig. 7 shows those two expressions implemented with logic gates. You can see that the sum of products version is simpler as it uses one less gate.

To see how we arrived at that expression, let's use some standard al-

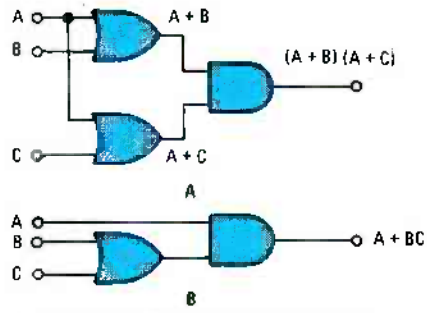


Fig. 7. A product-of-sums (A) can be simplified into a sum-of-products (B). You can see that the sum of products version (B) is much simpler.

gebra procedures. First, let's expand the expression on the left-hand side of the equation as we would a normal algebraic expression. We do that by multiplying each term inside the left-hand parentheses by each term in the right-hand parentheses, and adding all the terms together. The result of is:

$$(A + B)(A + C) = AA + AC + BA + BC$$

The expression to the right of the equals sign is correct algebraically and could be implemented directly with logic gates. However, it can be significantly reduced. We can use some of the previously described Boolean laws to reduce it.

First, take a look at the term AA on the right-hand side of the equation. Looking back, you should remember that AA = A as one of the laws of tautology. As a result, you can replace AA by A.

Next, notice that three of the terms

contain the letter A. You can factor out A, giving the expression:

$$A(1 + C + B) + BC$$

You can see that the term inside the parenthesis contains a binary 1 ORed with other variables. When you see a term such as that, you immediately know that you can apply one of the laws of union. That means that the entire expression inside the parenthesis can simply be replaced with a binary 1 giving:

$$A(1) + BC$$

Now you can use one of the laws of intersection to reduce the expression A(1) to A. The resulting expression then derived is:

$$A + BC$$

As you can see, the Boolean laws are in themselves extremely simple. Once you gain practice in recognizing their different uses inside longer, more complex expressions, you will be able to apply them and quickly reduce the expressions to something considerably simpler.

Laws of Association. The laws of association simply state that AND or OR expressions containing parentheses can be simplified by removing the parentheses. The two basic forms of this law are:

$$(AB)C = A(BC) = ABC$$
$$A + (B + C) = (A + B) + C = A + B + C$$

Normally, parentheses are used to set off two or more terms in groups to show that they are separate and distinct. However, when all the terms are ANDed (or multiplied as in algebra) or all of the terms are ORed (or added) together, then there is no need for the parentheses. They can be eliminated to simplify the final expression.

You can see the effect of those expressions when they are actually implemented with AND and OR gates. Figure 8A shows the AND expressions while Fig. 8B shows the OR expressions.

Laws of Absorption. There are four variations of the laws of absorption. Those are:

$$A(A + B) = A$$
$$A(\bar{A} + B) = AB$$
$$A\bar{B} + B = A + B$$
$$A\bar{B} + B = A + B$$

By looking at those expressions, you cannot really tell that they are actually

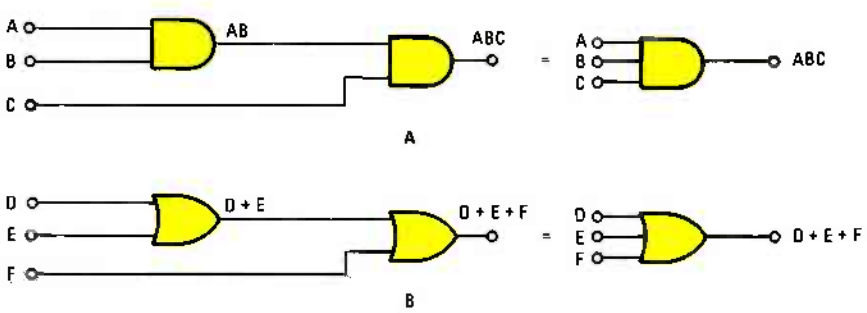


Fig. 8. Normally, parentheses are used to set off two or more terms in groups to show that they are distinct. When the same operation is being performed on all the terms then there is no need for the parentheses. When such expressions are implemented as in A and B without the parentheses, they are greatly simplified.

equivalent. However, you can see the expressions to the right of the equal signs are shorter and simpler than the expressions on the left.

As we did with the laws of distribution, we can use some Boolean laws to prove those expressions are equivalent to one another. Let's start with the first one:

$$A(A + B) = A$$

We will be working with the left-hand side of the expression to prove that it is indeed equal to A. The first thing we can do is to expand the expression by multiplying. That, of course, is the law of distribution. The result is:

$$A(A + B) = AA + AB$$

Of course, you can see that the term AA is redundant and it can be replaced by the letter A as the law of tautology indicates. Our new expression is simply:

$$A + AB$$

Again using the laws of distribution, we can factor an A out of both terms. That results in:

$$A + AB = A(1 + B)$$

The 1 + B indicates that the law of union applies. We can replace 1 + B by 1, giving the new expression:

$$A(1 + B) = A(1) = A$$

Of course, A(1) indicates that the law of intersection applies, and so the term A results. So, we proved the expression.

Now let's work on the second version of the laws of absorption:

$$A(\bar{A} + B) = AB$$

Again we will be working with the left-hand side of the equation trying to make it equal to AB. We can start by applying the laws of distribution and expanding the expression. The result is:

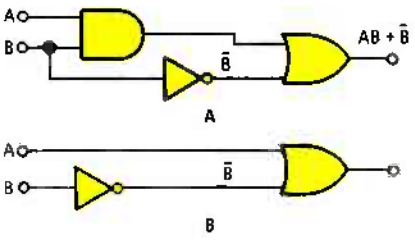
$$A(\bar{A} + B) = A\bar{A} + AB$$

The law of complements, of course, tells us that the expression $A\bar{A}$ is equal to 0. Therefore, our expression becomes:

$$A\bar{A} + AB = 0 + AB$$

The law of union applies here, therefore, the complete expression becomes equal to simply AB.

The last two forms of the law of absorption are relatively tricky to prove. There are ways to use algebraic techniques to do that, but we will avoid that here. Instead, we will show you another way to prove the equivalence of the



INPUTS		OUTPUTS		
A	B	AB	$\bar{B}$	AB + $\bar{B}$
0	0	0	1	1
0	1	0	0	0
1	0	0	1	1
1	1	1	0	1

AND THESE TO GET — OR THESE TO GET — THESE OUTPUTS ARE EQUAL

C

INPUTS		OUTPUTS	
A	B	$\bar{B}$	A + $\bar{B}$
0	0	1	1
0	1	0	0
1	0	1	1
1	1	0	1

OR THESE TO GET

D

Fig. 9. Obviously, the circuit in B is easier than the one in A. To prove their equivalence you can compare the outputs of their truth tables given in C and D.

two Boolean expressions by using truth tables. Let's use a truth table to prove each of the remaining two versions of the laws of absorption.

Let's start with the expression:

$$AB + \overline{B} = A + \overline{B}$$

First, you can see that if we implement each side of the equation with logic gates, we get the circuits shown in Fig. 9. Obviously, the circuit in Fig. 9B is a simpler and more desirable form. What we really don't know at this point is if the two circuits in Fig. 9 do actually produce the same logical output. Let's use a truth table to find out.

Since there are only two variables involved, then we have to develop a truth table containing all of the four different possible input combinations. Then we can develop a column in the truth table for each of the terms, such as AB. The table in Fig. 9C represents the circuit in Fig. 9A. Each column in the truth table is simply developed by observing the actual binary values of the expression and performing the logical operations.

The truth table for the circuit in Fig. 9B is given in Fig. 9D. You can see that the two columns representing the output expressions for the left- and right-hand sides of the equation are the same. Therefore, the simpler circuit does produce the same logical effect.

The truth table proving the last version of the laws of absorption is given in Fig. 10. Note that the columns for the left and right-hand sides of the equation are the same thereby proving equivalency.

The basic trick in using the laws of absorption is to become familiar with the different variations and recognizing them when they occur in larger expressions. That takes some practice, but after a while you will be able to spot them and immediately make the substitution. On occasion, when you do not recognize the different formats, it is possible that you may reduce the expression anyway using some of the other Boolean laws.

Reducing Logic Equations. Now let's apply the rules to simplifying Boolean expressions and the logic circuits they represent. Let's start with the circuit shown in Fig. 11A. The equation for that circuit is:

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

To solve a Boolean expression, the usual procedure is to first put it into a sum-of-products form. Remember that a sum of product format is two or more AND expressions ORed together. To do that, we must first expand the first expression by the law of distribution. The revised expression becomes:

$$D = AA + AB + \overline{C}$$

Next you should recognize that the laws of tautology allow you to replace AA by A. Doing that reduces the expression further.

$$D = A + AB + \overline{C}$$

You can now use the laws of distribution and factor out an A from the first two terms. That produces the revised equation:

$$D = A(1 + B) + \overline{C}$$

Next you will recognize that the 1 + B term can be replaced by 1 according to the laws of union. That gives us:

$$D = A(1) + \overline{C}$$

Finally you will recognize that the A(1) term can be replaced by the letter A by the law of intersection, giving the final expression:

$$D = A + \overline{C}$$

The revised circuit then is an OR gate as shown in Fig. 11B. What was previously a three-gate circuit is now a one gate circuit. Not only did you use fewer gates, but you also sped up the operation of the circuit. In Fig. 11A, three gates were used and the inputs must pass through each before reaching the output. Remembering that each gate has a finite amount of propagation delay, you can see that it will take longer for the output to be developed in Fig. 11A than it will in Fig. 11B. Assume the average gate propagation delay is 15 nanoseconds. With three gates or three levels of logic, the output would occur approximately:

$$3 \times 15 = 45$$

nanoseconds later. Using the revised circuit, the output would, of course, take only 15 nanoseconds to be developed.

Another example is shown in Fig. 12A. The Boolean expression representing that circuit is:

$$D = \overline{A}B\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C} + AB\overline{C}$$

As you can see, four 3-input AND gates and a 4-input OR gate are needed to implement the circuit. There is a good possibility that the circuit can be simplified.

To begin simplifying you should first scan the equation to see if you can recognize any common factors. You should keep the law of distribution in mind and attempt to locate terms that can be factored out to simplify the expression. You should first recognize that the first and fourth terms contain the expression $B\overline{C}$. You should also notice that the second and third terms contain the expression $A\overline{B}$. Those, of course, can be factored out. But first to simplify that task, the expression can be rearranged using the laws of com-

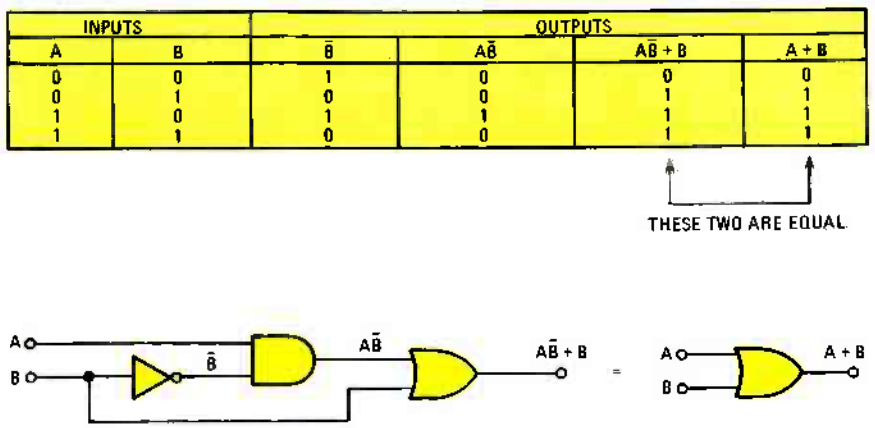


Fig. 10. Once again a truth table (A) shows the equivalence between two circuits (B) to prove another of the laws of absorption.

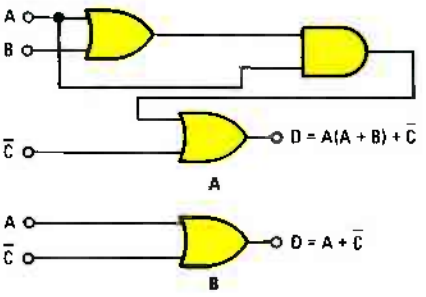


Fig. 11. By using the rules of Boolean algebra, the circuit in A can be converted into the circuit in B.

mutation, which means that the order in which the terms are listed is irrelevant. Therefore, let's rearrange the equation to group the last term with the first term since they both contain the same factor. The expression becomes:

$$D = \overline{A}B\overline{C} + AB\overline{C} + A\overline{B}C + A\overline{B}\,\overline{C}$$

Now you can use the law of distribution and factor out the common terms in the expressions. That results in:

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

Of course, you can see that the $(\overline{A} + A)$ and $(C + \overline{C})$ terms can be reduced to 1 by the law of complements. That produces the expression:

$$D = B\overline{C}(1) + A\overline{B}(1)$$

Finally, you can eliminate the 1's with the law of intersection producing the final equation:

$$D = B\overline{C} + A\overline{B}$$
$$= A\overline{B} + B\overline{C}$$

Of course, the circuit for the reduced

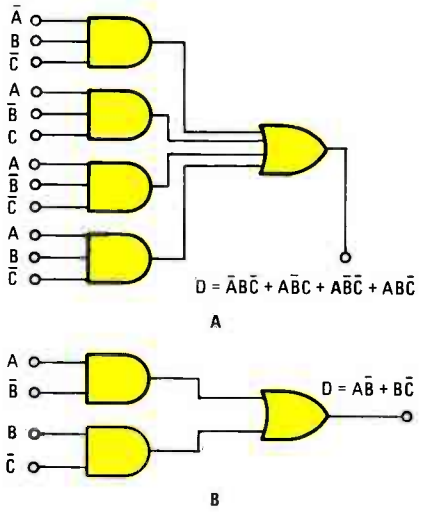


Fig. 12. As you can see, four 3-input AND gates and a 4-input OR gate are used for A. The circuit can be simplified to look like B.

equation is considerably simpler. It is illustrated in Figure 12B.

Here is another more complex example. Refer to the circuit in Fig. 13A and the equivalent Boolean expression:

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

When you get a product-of-sums expression like that, your first job should be to use the law of distribution and expand it into sum-of-products expression. You do that by ANDing each term in the left-hand expression with each term in the right-hand expression and

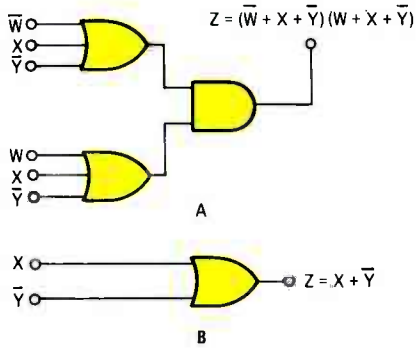


Fig. 13. It's hard to believe that circuits A and B are equivalent, but that shows you the value of Boolean algebra.

ORing the resulting terms together. By doing that you get:

$$Z = W\overline{W} + \overline{W}X + \overline{W}\,\overline{Y} + WX + X\overline{X} + X\overline{Y} + W\overline{Y} + X\overline{Y} + \overline{Y}\,\overline{Y}$$

In ANDing the expression yourself, you may get the term XW. I wrote WX as it is common practice to put all terms in alphabetical order. Just remember, XW = WX because of the laws of commutation.

The way to reduce a long expression like that is to be able to recognize all of the various identities that can result in some reduction. The first thing you can do is to look through the expression and take care of the obvious. For example, the first $W\overline{W}$ term can be replaced with 0 by the law of complements. The term XX can easily be replaced by a single X by the law of tautology. The same is true of:

$$\overline{Y}\,\overline{Y}$$

which can be replaced by:

$$\overline{Y}$$

and the two $X\overline{Y}$ terms can be reduced to one term. Based on the laws of union the first expression (equal to 0) drops out. The resulting expression is:

$$Z = \overline{W}X + \overline{W}\,\overline{Y} + WX + X + X\overline{Y} + W\overline{Y} + \overline{Y}$$

Next, scan through this equation looking for various ways to group the expressions for further reduction by the laws of distribution. The key is to locate as many common input terms as possible and group them together. For example, the second, fifth, sixth, and seventh terms contain $\overline{Y}$. The first, third, and fourth terms contain X. Therefore, we can rearrange the expression as follows according to the laws of commutation.

$$Z = \overline{W}\,\overline{Y} + X\overline{Y} + W\overline{Y} + \overline{Y} + \overline{W}X + WX + X$$

Now, factor out the common terms, $\overline{Y}$ from the first four terms and X from the last three terms. The result is:

$$Z = \overline{Y}(\overline{W} + X + W + 1) + X(\overline{W} + W + 1)$$

You can see that the expressions containing the 1s can readily be reduced to 1 by the laws of union. The result is:

$$Z = \overline{Y}(1) + X(1)$$

Now by the laws of intersection, that simply becomes:

$$Z = \overline{Y} + X$$

You could take that one step further and use the laws of commutation to get:

$$Z = X + \overline{Y}$$

Normally in an expression such as that the letters are written in alphabetical order, although, of course, it is not necessary. Note that the W term just drops out altogether. The resulting new circuit configuration is just an OR gate as shown in Fig. 13B.

Now let's take an actual design example. Usually a design starts as a truth table that defines all of the inputs and designates the desired output state for each set of inputs. Let's suppose that we want to design a circuit that will detect invalid four-bit BCD codes. If you will recall, BCD means binary coded decimal. That is a system for representing decimal numbers in binary form. We only wish to represent the decimal numbers 0 through 9 and that is done by using the 4-bit binary equivalents. However, with four bits you can represent a total of 16 different binary codes. ($2^4 = 16$). Those, as it turns out, represent the decimal numbers 0 through 15. They are illustrated in the truth table in Fig. 14. Only the first ten of the 4-bit codes are valid for BCD. What we would like to do is create a circuit that will recognize the invalid codes and tells us when they occur. The invalid codes are, of course, the last six codes in the table, or those representing the decimal numbers 10 through 15 (1010–1111).

We can now complete the design truth table. We want to develop an output X that is a binary 1 when one of the invalid codes is detected. Therefore, we want the output of the circuit to be 0 when a valid BCD code exists and binary 1 when an invalid code exists. See column X in Fig. 14.

As you learned in the previous article, you can now write the Boolean

expression from the truth table. You merely write down an AND expression of the terms corresponding to the inputs for each condition where a binary 1 appears in the output. As an example, a binary 1 output appears for the decimal number 10. That is the binary code 1010. In other words, at this time A is 1, B is 0, C is 1, and D is 0. To write a Boolean AND expression for this term, you write the input letter where a binary 1 occurs and the complement of the input letter where the 0 occurs. Therefore, the term for 1010 is:

	DECIMAL	BINARY A	B	C	D	X
VALID BCD CODES	0	0	0	0	0	0
	1	0	0	0	1	0
	2	0	0	1	0	0
	3	0	0	1	1	0
	4	0	1	0	0	0
	5	0	1	0	1	0
	6	0	1	1	0	0
	7	0	1	1	1	0
	8	1	0	0	0	0
	9	1	0	0	1	0
INVALID BCD CODES	10	1	0	1	0	1
	11	1	0	1	1	1
	12	1	1	0	0	1
	13	1	1	0	1	1
	14	1	1	1	0	1
	15	1	1	1	1	1

$X = A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$

Fig. 14. The first step in the design of any logic circuit is to draw the truth table and generate an equation.

$$A\bar{B}C\bar{D}$$

Using that technique, you can now write the complete expression. It is as follows:

$$X = A\bar{B}C\bar{D} + A\bar{B}CD + AB\bar{C}\bar{D} + AB\bar{C}D + ABC\bar{D} + ABCD$$

Note that the result is a standard sum of products Boolean expression which we derived from the truth table. Go through each of the terms in the expression to be sure that you see how they were obtained from the input states.

As usual, we have a Boolean expression which we now want to reduce to its minimum form. We could, of course, just implement it directly from the expression. The resulting circuit would appear as shown in Fig. 15A. It uses six 4-input AND gates and a 6-input OR gate. That would be a considerable nuisance to implement with integrated circuits, but it could be done although it would not be economical and it would take up considerable space. Based on what you have seen previously in this article, you can just about

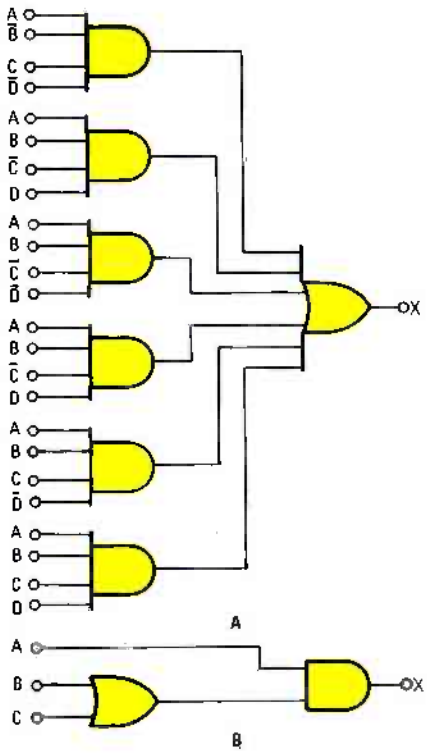


Fig. 15. The invalid-code detector generated directly from it's unsimplified equation is a monster in comparison to its Boolean-derived equivalent.

bet that the circuit will reduce to something simpler. All we have to do is reduce it using the same techniques used earlier.

The first thing that you should do is scan through the expression looking for common terms and expressions. You can see, for example, that the first two terms contain the common expression $A\bar{B}C$. The next two terms contain the common expression $AB\bar{C}$, while the last two terms contain the common expression ABC. As a result, you can factor those terms out producing this expression:

$$X = A\bar{B}C(\bar{D} + D) + AB\bar{C}(\bar{D} + D) + ABC(\bar{D} + D)$$

Of course, by using the law of complements and the law of intersection, the term $(\bar{D} + D)$ can be eliminated, leaving the expression:

$$X = A\bar{B}C + AB\bar{C} + ABC$$

You can now look through the expression for common terms again. The second and third terms contain AB, therefore, it can be factored out. The revised expression then becomes:

$$X = A\bar{B}C + AB(\bar{C} + C)$$

Again using the law of complements and intersection, you can reduce the parenthesis $(\bar{C} + C)$ expression. The result is:

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

Again you can look for common factors. A is common to both terms so let's factor it out next.

$$X = A(\bar{B}C + B)$$

If you have a sharp eye, you will recognize that the term $(\bar{B}C + B)$ can be reduced using the law of absorption to $B + C$. With that further simplification, the expression becomes:

$$X = A(B + C)$$

At this point, the expression is fairly simple. In fact, it can be implemented at this point with an AND gate and an OR gate as shown in Fig. 15B. That considerably simpler circuit will, of course, recognize the six invalid 4-bit codes, which are not BCD values.

A Design Example. Now let's take a look at how a designer would use Boolean algebra in creating some new digital product. As a simple example, let's assume that you wished to design a digital die.

You plan to use light-emitting diodes mounted in the standard die format to represent the numbers 1 through 6. That is illustrated in Fig. 16. Your job is to develop the digital circuit that will turn the lights on as desired.

The most obvious thing we know about the design is that there are six unique patterns that we wish to create. Therefore, the circuit must have six discrete states in which it can reside. We also know that we want the patterns

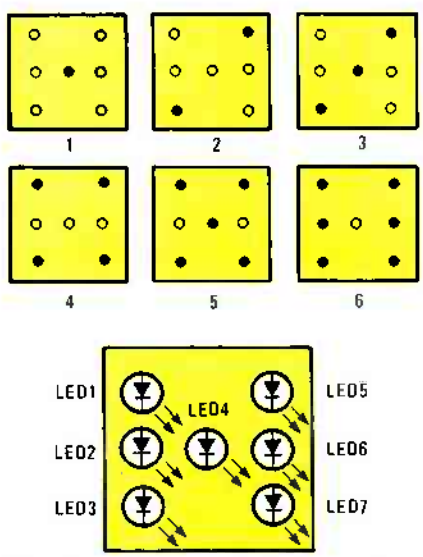


Fig. 16. An electronic die display can be easily made using six LED's.

shown in Fig. 16. The patterns define a total of seven different LED's (LED1–LED7). Therefore, we will need one signal to drive each LED.

The logic circuits driving the LED's will get their inputs from a signal source that defines the six desired states. The signal source will, in general, be some form of binary counter. We want a binary counter that has six states. In other words, it is a modulo six counter with six states and would also perform frequency division by 6. Naturally we can design such a circuit ourselves, but typically such circuits are already available. For example, we can use the 7492 TTL divide-by-12 counter. The device contains four flip-flops. One of them is independent while the other three are connected as a modulo 6 counter with the six binary states 000 through 101. Other counters are also available. Numerous CMOS divide-by-N counters are also available and are easily configured to produce a count of 6. For this example, we will assume the use of the 7492 counter with the six states described previously.

OK, at this point the problem is pretty well defined. The next step is to develop a truth table that fully defines every possible state. To start, let's write down the six counter states provided by the 7492 counter. We will use only the A, B, and C flip-flops which produce the binary count 000 through 101 as illustrated in Fig. 17. Note in the left-hand column we have recorded the decimal state that each of the binary counter states will represent. Those are the decimal numbers 1 through 6 that we want to represent in the form of an LED display. Just keep in mind that the binary value in this case does not equal the decimal state.

When N flip-flops are used for counting or frequency-division, it is possible to achieve a maximum count of 2^N states. That would give us a total number of:

$$2^3 = 8$$

With our 7492 counter, we are using only 6 of the 8 states. The two remaining states are 110 and 111. Since the counter will never go into those states, we do not care whether or not they exist. However, sometimes when we include the "don't care" states in expressions, we can more easily simplify the Boolean expressions we derive from the truth table. Therefore, we have recorded those two states in the truth table and labelled them "don't care."

DIE VALUE	COUNTER STATES	LED OUTPUTS						
	A B C	LED1	LED2	LED3	LED4	LED5	LED6	LED7
1	0 0 0	0	0	0	1	0	0	0
2	0 0 1	0	0	1	0	1	0	0
3	0 1 0	0	0	1	1	1	0	0
4	0 1 1	1	0	1	0	1	0	1
5	1 0 0	1	0	1	1	1	0	1
6	1 0 1	1	1	1	0	1	1	1
–	1 1 0	1	1	1	1	1	1	1
–	1 1 1	1	1	1	1	1	1	1

"DON'T CARE" STATES

Fig. 17. The truth table for the electronic die is shown here as the first design step.

We will use them in deriving the Boolean expressions as they will improve our ability to reduce them.

Next, let's develop the output states for the seven LED's. We have labelled the outputs as LED1 through LED7 in Fig. 17. Referring back to Fig. 16, you can see what LED's must be on to represent each digit. For example, to represent a decimal 1 on a die, LED4 will be on. For a die value of 1, the counter output states are 000. Therefore, we will record a binary 1 in the LED4 column and binary 0's in all other columns. To represent a 3, LED3, LED4, and LED5 are on, while the others are off. Look through the truth table and compare the LED entries for each of the six states. Finally note that we have recorded binary 1's in each column for the two "don't care" output states. Each LED value is entered in the table using the same reasoning until the table is complete.

The truth table now completely defines our design. At this point, we convert the truth table into the Boolean equations. We will develop a separate equation for each of the LED outputs. Let's start with LED1. Looking at the LED1 column, we note the input states where a binary 1 appears in the output. Then we write an AND expression involving inputs A, B, and C for each occurence of a binary 1 output. For a 1 input, we will write the input variable, while for a 0 input we'll write the complement of the input variable.

The equation is then written by ORing together each of the input terms. The input expression for LED1 is:

$$LED1 = \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

Work through the example yourself to be sure that you see how each of the terms was developed from the input code.

Now go through and derive all of the remaining Boolean expressions for LED2 through LED7. As shown here:

$$LED2 = A\bar{B}C + AB\bar{C} + ABC$$

$$LED3 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

$$LED4 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC$$

If you've examined the truth table closely enough, you should have discovered that the remaining three columns LED5, LED6, and LED7, are simply the same as columns LED1, LED2 and LED3. For example:

$$LED5 = LED3$$
$$LED6 = LED2$$
$$LED7 = LED1$$

Because the expressions are equal, it will certainly cut down on the amount of Boolean algebra we are going to have to do and the number of circuits we have to implement.

Now comes the fun part. Our job is to reduce the equations to the simplest possible form. Obviously, you can implement the equations just as they are written. That will require lots of 3-input AND gates and multiple-input OR gates. The resulting circuit will be very large and complex. By using Boolean algebra, we will be able to reduce the circuitry to a manageable level. Now let's use Boolean algebra to reduce each of the previous expressions to its simplest form. The reduction is done as follows and we provide you with the rule related to each step. Work through them yourself a step at a time to be sure that you understand them. Starting with the expression for LED1:

$$LED1 = \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

Rearrange by the laws of commutation:

$$LED1 = \bar{A}BC + ABC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C}$$

Factor out by laws of distribution:

$$LED1 = BC(\bar{A} + A) + A\bar{B}(\bar{C} + C) + AB\bar{C}$$

Reduce with the laws of complements:

$$LED1 = BC(1) + A\bar{B}(1) + AB\bar{C}$$

Reduce by laws of intersection:

$$LED1 = BC + A\bar{B} + AB\bar{C}$$

Factor out by law of distribution:

$$LED1 = BC + A(\bar{B} + B\bar{C})$$

Reduce by laws of absorption:

$$LED1 = BC + A(\bar{B} + \bar{C})$$

Expand by laws of distribution:

$$LED1 = BC + A\bar{B} + A\bar{C}$$

This could be rearranged by the law of commutation to:

$$LED1 = A\bar{B} + A\bar{C} + BC$$

Now let's look at LED2:

$$LED2 = A\bar{B}C + AB\bar{C} + ABC$$

Factor by laws of distribution:

$$LED2 = A\bar{B}C + AB(\bar{C} + C)$$

Reduce by law of complements:

$$LED2 = A\bar{B}C + AB(1)$$

Reduce by law of intersection:

$$LED2 = A\bar{B}C + AB$$

Factor with law of distribution:

$$LED2 = A(\bar{B}C + B)$$

Reduce by law of absorption:

$$LED2 = A(B + C)$$

Expand by law of distribution:

$$LED2 = AB + AC$$

Going on to LED3 next, we get:

$$LED3 = \bar{A}\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + A\bar{B}C + AB\bar{C} + ABC$$

Rearrange terms by law of commutation:

$$LED3 = \bar{A}\bar{B}C + A\bar{B}C + \bar{A}B\bar{C} + \bar{A}BC + A\bar{B}\bar{C} + AB\bar{C} + ABC$$

Factor by law of distribution:

$$LED3 = \bar{B}C(\bar{A} + A) + \bar{A}B(\bar{C} + C) + A\bar{B}\bar{C} + AB(\bar{C} + C)$$

Now reduce by law of complements:

$$LED3 = \bar{B}C(1) + \bar{A}B(1) + A\bar{B}\bar{C} + AB(1)$$

Reduce by law of intersection:

$$LED3 = \bar{B}C + \bar{A}B + A\bar{B}\bar{C} + AB$$

Factor by law of distribution:

$$LED3 = \bar{B}C + \bar{A}B + A(\bar{B}\bar{C} + B)$$

Reduce by law of absorption:

$$LED3 = \bar{B}C + \bar{A}B + A(\bar{C} + B)$$

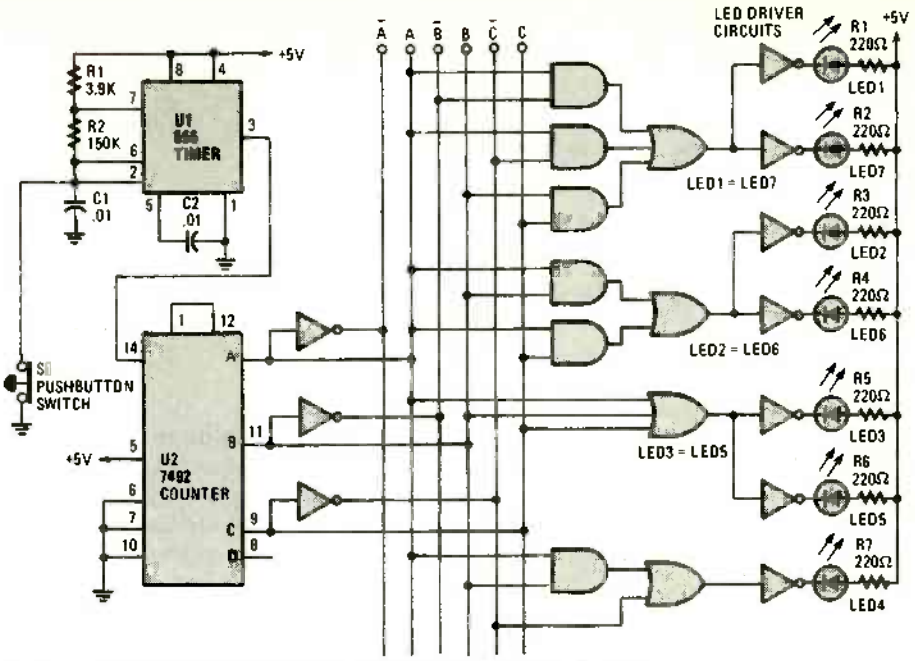


Fig. 18. At long last we arrive at the circuit for the electronic die.

Expand by law of distribution:

$$LED3 = \bar{B}C + \bar{A}B + A\bar{C} + AB$$

Rearrange by law of commutation:

$$LED3 = \bar{B}C + A\bar{C} + \bar{A}B + AB$$

Factor by law of distribution:

$$LED3 = \bar{B}C + A\bar{C} + B(\bar{A} + A)$$

Reduce by laws of complements and intersection:

$$LED3 = \bar{B}C + A\bar{C} + B$$

Rearrange by law of commutation:

$$LED3 = \bar{B}C + B + A\bar{C}$$

Reduce ($\bar{B}C + B$) by law of absorption:

$$LED3 = B + C + A\bar{C}$$

Again reduce ($C + A\bar{C}$) by law of absorption:

$$LED3 = B + A + C$$

Finally rearrange by law of commutation:

$$LED3 = A + B + C$$

Moving on to LED4:

$$LED4 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} + AB\bar{C} + ABC$$

Rearrange by law of commutation:

$$LED4 = \bar{A}\bar{B}\bar{C} + A\bar{B}\bar{C} + \bar{A}B\bar{C} + AB\bar{C} + ABC$$

Factor by law of distribution:

$$LED4 = \bar{B}\bar{C}(\bar{A} + A) + B\bar{C}(\bar{A} + A) + ABC$$

Reduce by laws of complements and intersection:

$$LED4 = \bar{B}\bar{C} + B\bar{C} + ABC$$

Factor by law of distribution:

$$LED4 = \bar{C}(\bar{B} + B) + ABC$$

Reduce by laws of complements and intersection:

$$LED4 = \bar{C} + ABC$$

Reduce by law of absorption:

$$LED4 = \bar{C} + AB$$

or rearranging by law of commutation:

$$LED4 = AB + \bar{C}$$

The remaining signals are redundant because:

$$LED5 = LED3$$
$$LED6 = LED2$$
$$LED7 = LED1$$

A summary of these equations:

$$LED1 = LED7 = A\bar{B} + A\bar{C} + BC$$
$$LED2 = LED6 = AB + AC$$
$$LED3 = LED5 = A + B + C$$
$$LED4 = AB + \bar{C}$$

At this point we have a set of minimal equations. We are now able to implement them with logic circuits. If you look through the set of equations, you will see that most consist of terms with two input variables ANDed together. Looking further, you probably noticed that some of the equations have the same term in them. For example, the term AB appears in both the expressions for LED2 and LED4. A single AND gate can be used to develop the expression AB which can then be used to implement both the LED2 and the LED4 outputs. That further reduces the

(Continued on page 105)