CIRCUITOUS ROUTES

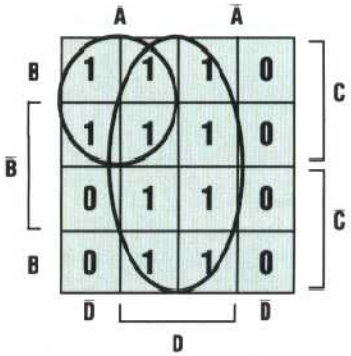
Having investigated the use of Karnaugh maps in the simplification of Boolean algebra expressions using two and three variables (see page 92), we now look at the more complex cases involving four variables. We also give examples of how k-maps are used to simplify the process of designing electronic circuitry.

Four Variables: In the cases of Boolean expressions involving four variables, the k-maps—and the expressions themselves — can seem formidably difficult. But by applying the simple ideas established when we looked at two and three variable k-maps, they soon become extremely familiar and easy to manipulate.

For example, suppose we are asked to simplify this expression:

$$AB\bar{C}\bar{D} + ABCD + \bar{A}BCD + A\bar{B}CD + \bar{B}CD + A\bar{B}\bar{C}D + \bar{A}\bar{B}CD + B\bar{C}D$$

We can see that a four variable k-map is required, but, although there are eight components of this expression, we will actually need to fill in 10 ones in the k-map (both the $\bar{B}CD$ and $B\bar{C}D$ terms represent two cases each). Thus the k-map is:



From the k-map we can see that the central group of eight ones represent every possible combination involving D. The group of four ones in the top left corner includes all the possible cases involving A AND C. So, the expression simplifies to A AND C OR D (expressed as: A.C + D).

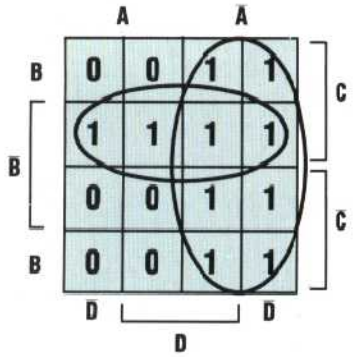
Sometimes an expression may need some initial manipulation to arrange it into a form suitable for representation on a k-map, as in the following example:

$$\overline{A + B + C} + \bar{A}.B + \overline{\bar{B} + C}$$

Here we must first apply de Morgan's law (see page 46) to the expression before it is possible to draw up a k-map. It is reformulated as:

$$\bar{A}.B.\bar{C} + \bar{A}.B + \bar{B}.C$$

and the k-map this expression produces is:



From the k-map this expression is seen to simplify to: $\bar{A} + \bar{B}.C$. Using de Morgan's law again, the expression is finally simplified to:

$$\overline{A.(B + \bar{C})}$$

CIRCUIT DESIGN

Example 1: Thirty Days

You probably know the rhyme: 'Thirty days hath September, April, June and November...'. Let us suppose that each month of the year is encoded into a four bit binary code—0001 for January through to 1100 for December. Our task is to design a circuit that will accept the four bit code as an input, and output a 1 if the month input has thirty days.

The truth table for such a circuit is:

MONTH	INPUTS				OUTPUT
	A	B	C	D	S
	0	0	0	0	X
JAN	0	0	0	1	0
FEB	0	0	1	0	0
MAR	0	0	1	1	0
APR	0	1	0	0	1
MAY	0	1	0	1	0
JUN	0	1	1	0	1
JUL	0	1	1	1	0
AUG	1	0	0	0	0
SEP	1	0	0	1	1
OCT	1	0	1	0	0
NOV	1	0	1	1	1
DEC	1	1	0	0	0
	1	1	0	1	X
	1	1	1	0	X
	1	1	1	1	X

The output X in the truth table signifies an invalid input. We shall assume that the circuit will *not* receive such signals. From the truth table, wherever S = 1 we can form the following Boolean expression from the binary bits of the input: