

EC262 Topic 5: The Three-Input Karnaugh Map

In Chapter 3 you learned how to simplify digital logic circuits by simplifying the Boolean expressions that represented these circuits. We simplified the Boolean expressions using Boolean algebra, and, particularly, the Table of Fun located on page 5 of Chapter 3.

Even though simplifying Boolean expressions using Boolean algebra provides hours of fun for bored midshipmen, it has some drawbacks. First, there is no clear algorithm to follow; becoming good at simplifying expressions requires intuition that is gained only through experience in solving many problems. Second, it is very easy to make a simple copying mistake (e.g., forgetting to include a complement sign on one of the literals as we go from step 17 to step 18 of our simplification process). Third, we often can't be sure we have simplified an expression as much as possible (which is why, on homework, quizzes and exams, you are told, for example, to "reduce the expression to three terms with eight literals").

Fortunately, we have much more convenient approach to simplifying digital logic circuits.

Three-Input Karnaugh Maps

A digital logic circuit with three separate inputs (let's call them x , y and z) has $2^3 = 8$ possible input combinations: 000, 001, 010, 011, 100, 101, 110, 111. In other words, the truth table for this circuit will have eight lines. Each of these possible input combinations represents a minterm. For instance, the input combination 000 represents the minterm _____ and the input combination 101 represents the minterm _____. There will, of course, be an output for each of the eight possible input minterm combinations.

A very helpful device for minimizing digital logic circuits is the *Karnaugh map* (*K-map*, for short). The K-map is a table where the rows and columns (together) account for all possible inputs to logic circuit. The outputs are displayed in a three-input K-map that is organized as follows:

	xy	00	01	11	10
z	0				
z	1				

(Note that the top row is arranged in the order from left to right: **00 01 11 10**. Take a moment to memorize this order. Note that the order is NOT 00 01 10 11.)

The entry in each table cell corresponds to the output for the inputs specified by the row and column. Within each of the eight squares of the K-map, we place the output (0 or 1) that corresponds to the input minterm.

AB	$A'B'$	$A'B$	AB	AB'
C	00	01	11	10
$C' 0$	$A'B'C'$	$A'BC'$	ABC'	$AB'C'$
$C 1$	$A'B'C$	$A'BC$	ABC	$AB'C$

From *Introduction to Logic Design*, Alan Marcovitz, McGraw Hill, 2010

Example. Suppose a digital logic circuit with three inputs (x , y , z) has the following Boolean expression:

$$x'y'z' + x'y'z + x'yz' + xyz'$$

What is the K-map for this circuit?

	xy	00	01	11	10
z	0				
z	1				

So...what do we do with this?

We look for groups of adjacent 1's, where the grouping can be in the same row or the same column. When looking for adjacent 1's, we consider that the table rows and columns wrap around.

Find all groupings in the table above.

z \ xy	00	01	11	10
0	1	1	1	0
1	1	0	0	0

Now, the idea is to **select the fewest groups that cover all the cells that have a 1.**

Look at the left red circle. It corresponds to the terms $x'y'z' + x'y'z$. But this can be rewritten as:

Thus, these two terms together can be expressed as

What can the red circle on the right be rewritten as?

What can the green circle be rewritten as?

Do we need to include this last term?

What is a simpler way of implementing $x'y'z' + x'y'z + x'yz' + xyz'$?

Example. Suppose a digital logic circuit with three inputs (x, y, z) has the following Boolean expression:

$$x'y'z' + xy'z' + x'yz + xyz$$

Simplify this expression using a K-map.

First, write the K-map:

Now, the idea is to find pairs of adjacent 1's in such a way that we select the fewest groups that cover all the cells that have a 1. Also, remember that when looking for adjacent 1's, we consider that the table's rows and columns wrap around.

Thus, the original expression can be reduced to

Note that we can do this simplification mechanically without even thinking of the Boolean algebra: When we group two adjacent cells, we can simply drop the variable that changes its value between the two cells. So, when we pair the 1 in the top-left corner corresponding to $x'y'z'$ with the 1 in the top-right corner corresponding to $xy'z'$, we note that the variable that changes between these two terms is x ...so we can just drop that literal and reduce these two cells to $y'z'$.

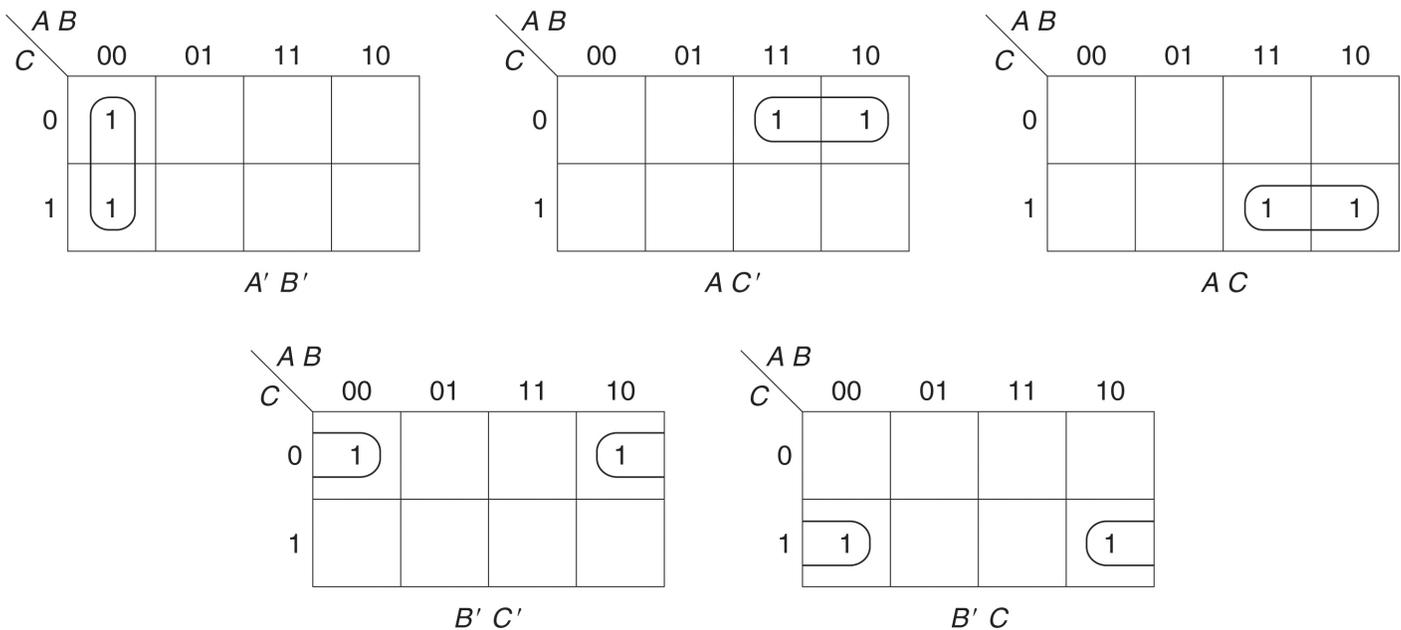
Example. Suppose that a logic circuit has the following truth table:

x	y	z	Output
0	0	0	0
0	0	1	0
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	1
1	1	1	1

Construct the K-map for this circuit and use the K-map to minimize this circuit.

A simpler circuit is:

The picture below emphasizes some of the various options for grouping pairs of 1's in a three-input K-map.



From *Introduction to Logic Design*, Alan Marcovitz, McGraw Hill, 2010

So, again, remember that a pair of adjacent 1's will correspond to a product term with one variable missing...the variable that changed value between the two terms.

Example. Suppose that a logic circuit has the following truth table:

x	y	z	Output
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	0
1	0	1	1
1	1	0	0
1	1	1	1

Construct the K-map for this circuit and use the K-map to simplify the circuit.

If we group

If we group

So, our simplified circuit is _____

Yes... and we can modify the K-map technique to do this:

So, for the above K-map, we could immediately group as:

This immediately reduces to just:

When we group four 1's together, we will obtain a product term with...

Which variables?

Example. We want to design a 3-input digital logic circuit that will output a 1 if inputs x and y are both 1 or if input x is 1 and inputs y and z are both 0. Show the truth table and the K-map, and simplify the Boolean expression using the K-map.

The truth table is

What is the Boolean expression expressed in canonical SOP form?

Suppose you could use gates that support any number of inputs. How many logic gates would be needed to directly implement this Boolean expression?

Now, show the K-map.

What simpler Boolean expression results from the K-map?

Draw the logic circuit.

Note that this simpler circuit only requires

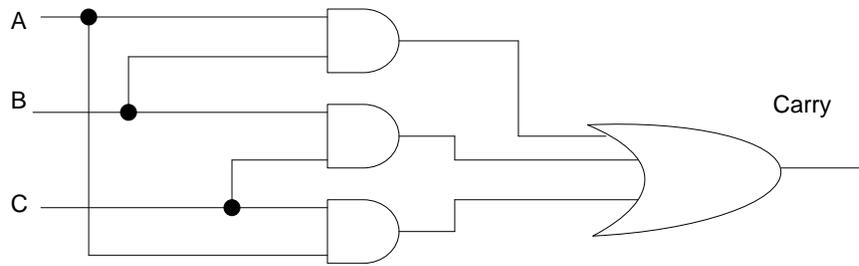
Example. Full Adder Consider a digital logic circuit to add three bits together. Calling the three inputs A , B and C , the truth table is

A	B	C	Carry	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Show the K-map for the Carry bit, and simplify the Boolean expression to derive a simpler implementation.

This simplifies to:

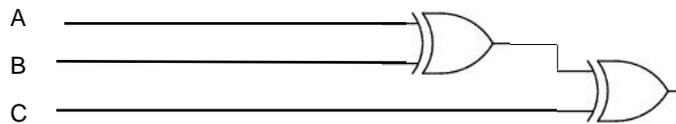
The logic circuit for the Carry bit is:



What is the K-map for the Sum bit?

Can this be simplified?

You should convince yourself that the Sum bit can be implemented by two XORs.



Example. Your friend is a math major who is taking a logic course. In the study of logic, the implication operator, denoted \rightarrow , is often used. The truth table for $(p \rightarrow q) \rightarrow r$ is shown below.

p	q	r	$(p \rightarrow q) \rightarrow r$
0	0	0	0
0	0	1	1
0	1	0	0
0	1	1	1
1	0	0	1
1	0	1	1
1	1	0	0
1	1	1	1

Determine the canonical SOP Boolean expression for this truth table.

Draw the Karnaugh map for this truth table.

Using your Karnaugh map, determine the simplest Booleans expression that can represent this truth table.

Sketch the logic circuit for the Boolean expression you developed. Use only NOT, AND and OR gates.

Suppose you only have NOT and AND gates available (i.e., you have no OR gates). Redesign you logic circuit from part (d) using only NOT and AND gates.

Example (from Marcovitz) Draw the K-Map that corresponds to the Boolean expression:

$$F = AB' + AC + A'BC'$$

Start with the empty K-map:

Terminology. Now that we have gained a heuristic appreciation for the use of K-maps, let's formalize the terminology.

An *implicant* of a Boolean function is any product term that can be used in an SOP expression for the function.

In a K-map, an implicant is any group of one, two, four or eight 1's that form a rectangle. Our goal is to choose the minimum number of implicants such that all 1's of the K-map are included (covered by) at least one of the implicants.

Example. List all the implicants in the K-map below.

C \ AB	00	01	11	10
0	1	1	0	1
1	0	1	0	0

A *prime implicant* is an implicant that is not completely contained in any other implicant.

What are the prime implicants in the K-map shown above?

A *essential prime implicant* is a prime implicant that includes at least one 1 that is not included in any other prime implicant.

What are the essential prime implicants in the K-map shown above?

Restated using this terminology, our procedure to use a K-map to simplify a Boolean expression is:

- Find all essential prime implicants. Circle these on the K-map. These will be in the final minimal SOP expression
- Find enough prime implicants to sweep up all the remaining 1's in the K-map. Accomplish this by attempting to first choose prime implicants that include as many 1's as possible, while avoiding leaving isolated 1's.

Using this procedure, what is the minimum SOP expression for the K-map above?

Four-Input Karnaugh Maps

The extension of three-input K-maps to four-input K-maps is quite straightforward.

A digital logic circuit with four separate inputs (let's call them A , B , C and D) has $2^4 = 16$ possible input combinations: 0000, 0001, ..., 1110, 1111. In other words, the truth table for this circuit will have 16 lines. Each of these possible input combinations represents a minterm. For instance, the input combination 0101 represents the minterm :

There will, of course, be an output for each of the sixteen possible input minterm combinations.

The K-map for a four-input problem is a table where the rows and columns (together) account for all possible inputs to logic circuit. The outputs are displayed in a four-input K-map that is organized as follows:

AB \ CD	00	01	11	10
00				
01				
11				
10				

Note that the rows and columns are arranged in the order from left to right (and from top to bottom) as:

00 01 11 10.

Take a moment to memorize this order. Note that the order is NOT 00 01 10 11 .

Exactly as with the three-input K-map, the entry in each table cell of the four-input K-map corresponds to the output for the inputs specified by the row and column. Within each of the sixteen squares of the K-map, we place the output (0 or 1) that corresponds to the input minterm.

AB \ CD	00	01	11	10
00	$A'B'C'D'$	$A'BC'D'$	$ABC'D'$	$AB'C'D'$
01	$A'B'CD$	$A'BCD$	$ABC'D$	$AB'C'D$
11	$A'B'CD$	$A'BCD$	$ABCD$	$AB'CD$
10	$A'B'C'D$	$A'BC'D$	$ABCD'$	$AB'C'D$

From *Introduction to Logic Design*, Alan Marcovitz, McGraw Hill, 2010

Example. Suppose a digital logic circuit with three inputs (x, y, z) has the following Boolean expression:

$$A'B'C'D' + A'B'CD + A'BCD + ABC'D' + ABC'D + ABCD + AB'CD$$

What is the K-map for this circuit?

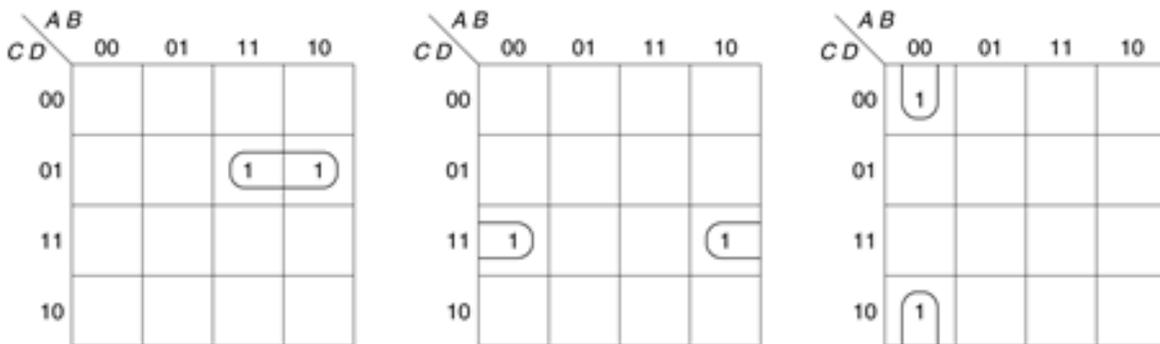
So...what's the plan?

We look for groups of adjacent 1's, where the grouping can be in the same row or the same column. When looking for adjacent 1's, we consider that the table rows and columns wrap around.

Informally, we want to

When looking for adjacent 1's,

Example. For each of the K-maps below, write the product term that results from combining groups of adjacent 1's.



From *Introduction to Logic Design*, Alan Marcovitz, McGraw Hill, 2010

Example. For each of the K-maps below, write the simplest Boolean expression that results from analyzing the K-map.

CD \ AB	00	01	11	10
00	0	0	0	0
01	1	0	0	1
11	1	0	0	1
10	0	0	0	0

CD \ AB	00	01	11	10
00	1	0	0	1
01	0	1	1	0
11	0	1	1	0
10	1	0	0	1

Example. For each of the K-maps below, write the simplest Boolean expression that results from analyzing the K-map.

	AB	00	01	11	10
CD		00	01	11	10
	00	1	1	0	0
	01	1	1	0	0
	11	1	1	0	0
	10	1	1	0	0

	AB	00	01	11	10
CD		00	01	11	10
	00	1	1	1	1
	01	0	0	0	0
	11	0	0	0	0
	10	1	1	1	1

We have to remember that the idea is to **select the fewest groups that cover all the cells that have a 1.**

Example. Write the simplest Boolean expression that results from analyzing the K-map below.

	AB	00	01	11	10
CD		00	01	11	10
	00	1	0	0	0
	01	1	0	1	1
	11	1	0	1	1
	10	1	0	0	0

	AB	00	01	11	10
CD		00	01	11	10
	00	1	0	0	0
	01	1	0	1	1
	11	1	0	1	1
	10	1	0	0	0

But what about this:

	AB	00	01	11	10
CD		00	01	11	10
	00	1	0	0	0
	01	1	0	1	1
	11	1	0	1	1
	10	1	0	0	0

What if we had inadvertently included this term... what would our (wrong!) answer have been for the “simplest Boolean expression”?

Could this have been simplified to the right answer using the Table of Fun?

The point: Midshipmen love K-maps because

Now that we are comfortable with four-input K-maps, let's recall our procedure to use a K-map to simplify a Boolean expression:

- Find all essential prime implicants. Circle these on the K-map. These will be in the final minimal SOP expression
- Find enough prime implicants to sweep up all the remaining 1's in the K-map. Accomplish this by attempting to first choose prime implicants that include as many 1's as possible, while avoiding leaving isolated 1's.

Example. Recall from page 2 that the Boolean expression:

$$A'B'C'D' + A'B'CD + A'BCD + ABC'D' + ABC'D + ABCD + AB'CD$$

has the K-map:

CD \ AB	00	01	11	10
00	1	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	0	0

List all the prime implicants:

What are the essential prime implicants?

Circle the essential prime implicants on the K-map:

CD \ AB	00	01	11	10
00	1	0	1	0
01	0	0	1	0
11	1	1	1	1
10	0	0	0	0

Have we found enough prime implicants to sweep up all the remaining 1's in the K-map?

What is a simpler way of implementing:

$$A'B'C'D' + A'B'CD + A'BCD + ABC'D' + ABC'D + ABCD + AB'CD$$

Now that we are (hopefully!) becoming more comfortable with the means for minimizing K-maps, you may have noticed that there is a benefit to focusing on “outliers”—1's on the K-map that have either no neighboring 1's or just a single neighboring 1.

Look at the top-left box in the example above. This 1 has no neighboring 1's so it must be an essential prime implicant and must be in the minimal solution.

Similarly, look at the one at the top of the third column. It has a single neighbor. So, this cell and its neighbor (the cell below it) must participate in an essential prime implicant.

And again: Note that we can do simplification mechanically without even thinking of the Boolean algebra: When we group 2 adjacent cells, we can simply drop the variable that changes its value between the 2 cells.

When we group four cells we drop the _____ variables that change their values among the cells.

When we group eight adjacent cells we drop the _____ variables that change their value among the cells.

Example. Minimize the following K-map:

wx \ yz	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

wx \ yz	00	01	11	10
00	1	0	0	1
01	0	0	0	0
11	0	0	0	0
10	1	1	1	1

Simpler expression:

Example. (From text) Derive a minimum Boolean expression for a function that has the K-map below.

YZ \ WX	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	1	1
10	0	0	0	0

First...

What's special about this guy?

Do we have any say in what this will be?

So, we have this:

YZ \ WX	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	1	1
10	0	0	0	0

What are the other prime implicants?

YZ \ WX	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	1	1
10	0	0	0	0

Which of these additional prime implicants are essential (i.e., contain at least one 1 that is not included in any other prime implicant)?

So, since essential prime implicants must be in the minimal solution, so far we have:

YZ \ WX	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	1	1
10	0	0	0	0

We have no other essential prime implicants. What do we do now?

So, the final answer is

YZ \ WX	00	01	11	10
00	1	1	1	1
01	0	1	0	0
11	0	1	1	1
10	0	0	0	0

and the minimal Boolean expression is

We interrupt this lecture for a subliminal message regarding service assignment.



Back to the lecture!

Example. (from Marcovitz text) Suppose that a logic circuit has the following truth table:

<i>a</i>	<i>b</i>	<i>c</i>	<i>d</i>	Output
0	0	0	0	1
0	0	0	1	0
0	0	1	0	1
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	1
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	0

Construct the K-map for this circuit and use the K-map to simplify the circuit.

	ab	00	01	11	10
cd	00				
01					
11					
10					

	ab	00	01	11	10
cd	00				
01					
11					
10					

	ab	00	01	11	10
cd	00				
01					
11					
10					

So, now we have:

cd \ ab	00	01	11	10
00				
01				
11				
10				

All we have is

So the final answer is:

Example. (from text) Derive a minimum Boolean expression for a function that has the K-map below.

CD \ AB	00	01	11	10
00	0	1	0	0
01	0	1	1	1
11	1	1	1	0
10	0	0	1	0

Example. (from text) Derive a minimum Boolean expression for a function that has the K-map below.

CD \ AB	00	01	11	10
00	1	1	0	1
01	0	0	1	1
11	1	1	0	1
10	1	0	1	1