

EE334
Karnaugh Maps and Logic Design

A **Karnaugh map (K-map)** is graphical representation of a truth table. The usefulness of the Karnaugh comes from the human mind's ability to recognize patterns. Patterns that we can see in the Karnaugh map will enable us to write simplified, but maybe not optimal, equations for the truth table represented pictorially in the K-maps.

Drawing K-maps

To draw a K-map, you start with the logic function or truth table and then...

1. Create a rectangular grid with 2^n cells, where n = number of variables. Examples
 - a. If there are 2 variables, make either a 1x4 (1 row by 4 columns) or a 2x2 grid.
 - b. If there are 3 variables, make a 2x4 grid.
 - c. For 4 variables, create a 4x4 grid. Four variables is the most we'll see in this course.
2. Label the rows and columns of the grid to account for all possible combinations of the logic values of the variables.
 - a. It doesn't matter which variables go where.
 - b. The labeled logic values (0s and 1s) of neighboring rows (columns) should only change by one bit at a time, for reasons that will become apparent later.
3. Place 1s in the grid cells where the variable values of the corresponding truth table or logic function produce a 1. You do not need to plot 0s (they add clutter).

1. Fill in the following K-map for the AND function. The AND truth table is shown for your convenience.

A	B	Out
0	0	0
0	1	0
1	0	0
1	1	1

		B	
		0	1
A	0		
	1		

2. Fill in the K-map below for the OR function shown in the truth table. Pay attention to the labeling of the variable values on the columns.

A	B	Out
0	0	0
0	1	1
1	0	1
1	1	1

				AB			
00		01		11		10	

Karnaugh Maps and Logic Design

3. Complete the K-map for $f(A,B,C) = \sum m(0,2,3,7)$. $f(A,B,C) = \sum m(0,2,3,7)$ is short hand for which rows of a truth table evaluate to 1. In this case the rows where the triplet (A,B,C) has the pattern 000, 010, 011, and 111 are the rows that equate to 1. If it helps, draw the truth table before filling out the K-map.

		BC			
		00	01	11	10
A	0				
	1				

Writing Optimized Equations Using K-maps

As mentioned earlier, K-maps allow us to easily detect patterns in logic functions and truth tables. We can use those patterns to write optimized, but maybe not always optimal, equations for the logic function or truth table the K-map represents.

Although writing equations from K-maps is a bit of an art form, a general algorithm for writing equations from a K-map is:

1. Circle a grouping of 1s in the grid according to the following rules.
 - a. The number of 1s in the group must be a power of 2. (e.g., 1, 2, 4, 8, etc.)
 - b. The group must be rectangular.
 - c. The group must be as large as possible.
 - d. The group must have at least one 1 in it that is not already in some other group.
 - e. Place a priority on including 1s that cannot be grouped any other way.
 - f. You may wrap around the left and right edges of the K-map to form a group.
 - g. You may wrap around the top and bottom edges of the K-map to form a group.
2. Write an AND equation for the group using variable states that are consistent across the group. This takes practice.
3. Repeat steps #1 and #2 until all 1s are a member of some group.
4. OR together all the AND equations you created in step #2. This creates an SOP expression.

Note: This method of creating equations from K-maps results in pretty good equations – equations with a low number of variables and logic operations. *However, the method does not guarantee the best equation.* If asked to simplify a logic function, a K-map is a good first step. To obtain the best possible equation, you may have to manipulate the resulting K-map equation using Boolean algebra of to get the best equation.

EE 334
Karnaugh Maps and Logic Design

4. Write an SOP expression for the following K-map.

		BC			
		00	01	11	10
A	0	1			
	1			1	1

5. Write an SOP expression for the following K-map.

		YZ			
		00	01	11	10
WX	00				
	01	1			1
	11		1	1	
	10		1	1	

6. There are at least two different equations you can create from K-map below. See if you can come up with two different ones. Which grouping is better? The K-map has been reproduced twice for your convenience.

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

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

Don't Care Conditions and K-maps

Sometimes in logic we find ourselves in situations where we don't care if the outcome of an operation is a 0 or a 1. When or why does this occur? It could happen when we are designing a logic function based on some requirements and the requirements don't say exactly what is to happen in each situation. Another way it can happen is that a particular combination of inputs will never occur. If the input can't happen, we don't care what the output will be.

The interesting thing about “**don't care**” outcomes is that we can use them as 1s in our K-maps if it helps us to make groups of 1s. If they don't help us, we can ignore them (treat them as 0s).

Let's work through a full example of how don't cares can come to be and how we can use them to our advantage in creating equations from K-maps.

7. There are 10 decimal digits (0 through 9). A binary coded decimal (BCD) digit is a binary number that has the same value as one of the ten decimal digits. For example the BCD digit for decimal 6 is 0110. Since we have to include decimal 8 and 9 (1000 and 1001, resp.), a BCD digit is always four bits long.

|
Now suppose we want to create a function that signals when a 1 when a BCD digit is evenly divisible by 3. Create and complete a truth table for this function. Label the bits of the BCD digit as ABCD with A being the MSB (most significant bit) and D being the LSB. Be on the lookout for opportunities to include don't care outcomes. Mark the “don't care” outcomes with an “X”.

