Logic Circuit Expressions

A logic circuit is any electrical circuit that is able to evaluate a logic relation, such as those defined by a core logic operator or derived operator: AND, OR, NOT, XOR, NAND, NOR, or complex logic expression. With at least two (2) inputs, a logic circuit can generate at least one (1) output.

But how is the logic expression determined solely from the input to the required output?

The answer is in defining the possible states that provide the necessary output (as "true" or "on" states) and those that do not (as "false" or "off" states). The resulting logic expression is the pattern of matching the inputs to outputs, as seen below.

The final, *optimal* (or *minimised)* expression is a collection of logic operators that directly relate all states of the inputs to the necessary states of the output(s). In practice, rather than use a multitude of AND, OR, and NOT logic gates, the cheapest gate to manufacture circuits is via the NAND gate (Not AND), since it has been shown that AND, OR, and NOT operations can be defined using only NAND (http://encyclopedia.thefreedictionary.com/Logical%20nand).

> **Note:** Although the creation of physical logic circuits themselves is fundamental to computer science, and just plain interesting, it is beyond the scope of this course, which is geared towards individuals in *computing* science, not *computer* science. The following discussion considers the aspects of logic circuits only to the point of deriving the logic expressions, leaving actual circuit to electrical and electronic engineers.

Logic Tables

A logic table maps *a set of inputs to a specific set of outputs* (unlike a *truth table* that evaluates an expression through all inputs to a final result). From the table, a logic expression is constructed that exactly describes the relation of input to output at the binary logic level—but no processing is described…yet!

Consider the logic table example describing a circuit that has three (3) inputs (A, B, C) and one (1) output (Z),

A	B	C	|	Z	
0	0	0	|	0	
0	0	1	|	0	
0	1	0	|	0	
0	1	1	|	1	<- "on"
1	0	0	|	0	
1	0	1	|	1	<- "on"
1	1	0	|	1	<- "on"
1	1	1	|	1	<- "on"

This table above describes a circuit that is true whenever at least two inputs are true (the circuit is "on" whenever two input are "on"); in all other cases the circuit is false. Observe that the circuit is "on" (or lets electricity pass) only 4 (four) cases; these are the ones of interest. The table below shows only these cases,

A	B	C	|	Z	
0	1	1	|	1	(1)
1	0	1	|	1	(2)
1	1	0	|	1	(3)
1	1	1	|	1	(4)

From this smaller table, an initial logic expression is constructed. The technique is to consider each row as having its variables **AND**ed together to result on **1**, complementing any variables as necessary; with the overall expression having the table's rows **OR**ed together.

$$Z = \overline{A}\cdot B\cdot C + A\cdot\overline{B}\cdot C + A\cdot B\cdot\overline{C} + A\cdot B\cdot C$$
$$\quad\ (1)\qquad\ (2)\qquad\ (3)\qquad\ (4)$$

For the three inputs (A,B,C), this logic expression will always "result in true (on) if at least two inputs are true (on); false otherwise." But is the expression as efficient as possible; can it be simplified?

At this point, the fundamental logic identities and theorems could be applied to the expression in hopes of simplification, but there is a more interesting method (although it may not be as intuitive).

Karnaugh Map (K-Map)

Rather than use a complex sequence of Boolean identities, theorems, and laws in simplifying a logic expression, a K-map offers a simpler, shorter, and compressed, graphical method for logic simplification.

K-Map, or Karnaugh Map, was invented by Maurice Karnaugh in 1953 (based on earlier work by E. W. Veitch). Karnaugh was a telecommunications engineer at Bell Labs studying how digital logic could be applied to telephone logic, and out of necessity (and from seeing the spatial relationships of Boolean sets) created a "short hand" or "short cut" method of simplifying logic expressions.

K-maps are a useful tool in graphically representing Boolean algebraic (logic) expressions. In simplifying the logic expression, visual 'pattern matching' is used to combine terms leading to a minimised or, hopefully, reduced expression.

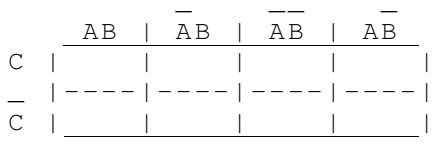
> **Note:** The following description of the K-map technique is modified slightly from the original definition. The modification allows for a direct representation of the logic expression rather than the Boolean patterns originally described by Karnaugh. Consider the modification a "short cut" to K-map—a "short cut of a short cut," if you will.

There a few rules to drawing a K-map (see examples for clarification),

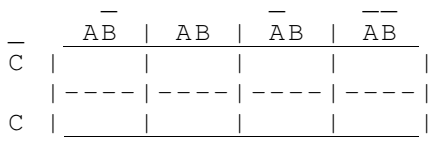
- the logic expression under consideration must be in "sum of product" (SoP) form; other forms can not be directly represented (consider using the laws and theorems to "convert" an expression to SoP
- in defining the columns and rows, the complement of each variable group must follow a sequence that cycles
- a dot is placed in each cell that corresponds to a term in the expression (one term may represent more than one dot!)
- in grouping "dots," group sizes can consist of only 1, 2, 4, 8, or 16 dots; and the goal is always to form the largest groups, and groups must always be rectangular in shape
- dots in one group can be "borrowed" to form a larger group with another set of dots, but this creates two groups, not a large, single group

In evaluating the simplified logic expression from the K-map, each group represents a term in the expression. A variable in a particular group are eliminated if the variable exists as both itself and its complement (hence, it plays no role in the term). Finally, all terms are **OR**ed together to form the final, simplified expression.

Consider the example above with three variables (A,B,C). The K-map *could* look like,



or with a small change to the complement sequence,



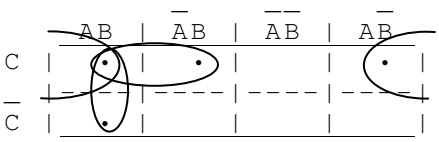
but in both the complement sequence cycles, in that, "there is only one change in complement between columns."

In using the K-map to simplify the expression from the example, the boxes are filled for every term that matches the state of the variables.

$$Z = \bar{A}\cdot B\cdot C + A\cdot\bar{B}\cdot C + A\cdot B\cdot\bar{C} + A\cdot B\cdot C$$

	$\bar{A}B$	$A\bar{B}$	$\bar{A}\bar{B}$	AB
C	•	•		•
$\bar{C}$	•			

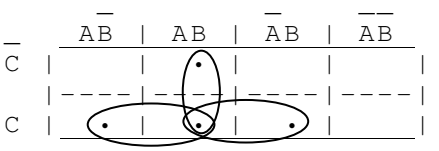
From the map, groups are made and an expression derived. If the grouping is correct, it is a simplified expression.



$= AC + AB + BC$ (minimised)

Since the K-map cycles, the alternate table could have been used, producing a slightly different map. The results, of course, are the same.

	$\bar{A}B$	AB	$\bar{A}\bar{B}$	$A\bar{B}$
C		•		
$\bar{C}$	•	•	•	



$= AC + AB + BC$ (minimised)

K-Map Practice

Applying the concepts of "logic table to logic expression" shown above, complete the following,

a)

A	B	C	D	Z
0	0	0	0	0
0	0	0	1	0
0	0	1	0	0
0	0	1	1	0
0	1	0	0	1
0	1	0	1	0
0	1	1	0	0
0	1	1	1	0
1	0	0	0	1
1	0	0	1	1
1	0	1	0	1
1	0	1	1	1
1	1	0	0	1
1	1	0	1	0
1	1	1	0	1
1	1	1	1	1

b) derive the expression, after shifting the column or row sequence, but keep the dots in the same relative cells):

	$\bar{A}B$	$A\bar{B}$	$\bar{A}\bar{B}$	AB
CD	•	•		•
$\bar{C}D$	•	•		
$\bar{C}\bar{D}$	•	•		
$C\bar{D}$	•	•	•	•

c) derive the final expressions for both Z_1 and Z_2,

"A circuit that takes in 4 inputs and turns on one circuit (Z_1) if the total number of inputs is less than 2 (0,1,2), and another circuit (Z_2) if the total number of inputs is 3 or 4."

Here is the header of the logic table,

A	B	C	D	Z_1	Z_2
0	0	0	0	1	0

K-Map References

The following can provide some more details on using K-map,

Math 162—Camosun College: http://ccins.camosun.bc.ca/~trushel/math162/logic/logic7/logic7.htm

CS 103A—Standford University: http://www-cse.stanford.edu/classes/cs103a/h9BooleanAlgebra.pdf