K-Map 1

Lesson Objectives:
Even though **Boolean expressions** can be simplified by algebraic manipulation, such an approach lacks clear regular rules for each succeeding step and it is difficult to determine whether the ***simplest expression*** has been achieved.

In contrast, Karnaugh map (K-map) method provides a straightforward procedure for simplifying Boolean functions.

K-maps of up to 4 variables are very common to use. Maps of 5 and 6 variables can be made as well, but are more cumbersome to use.

Simplified expressions produced by K-maps are always either in the **SOP** or the **POS** form.

The map provides the same information contained in a Truth Table but in a different format.

The objectives of this lesson are to learn:
1. How to build a 2, 3, or 4 variables K-map.
2. How to obtain a minimized SOP function using K-maps.

Code Distance:
Let's first define the concept of Code Distance. The distance between two binary code-words is the number of bit positions in which the two code-words have different values.

For example, the distance between the code words **1001** and **0001** is **1** while the distance between the code-words **0011** and **0100** is **3**.

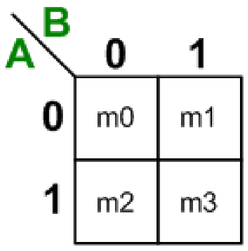
This definition of code distance is commonly known as the ***Hamming distance*** between two codes.

Two-Variable K-Maps:
The 2-variable map is a table of 2 rows by 2 columns. The 2 rows represent the two values of the first input variable A, while the two columns represent the two values of the second input variable B.

Thus, all entries (squares) in the first row correspond to input variable A=0, while entries (squares) of the second row correspond to A=1.

Likewise, all entries of the first column correspond to input variable B = 0, while entries of the second column correspond to B=1.

Thus, each map entry (or square) corresponds to a unique value for the input variables A and B.



For example, the top left square corresponds to input combination AB=00. In other words, this square represents minterm m_0.

Likewise, the top right square corresponds to input combination AB=01, or minterm m_1 and the bottom left square corresponds to input combination AB=10, or minterm m_2. Finally, the bottom right square corresponds to input combination AB=11, or minterm m_3.

In general, each map entry (or square) corresponds to a particular input combination (or minterm).

Since, Boolean functions of two-variables have four minterms, a 2-variable K-map can represent any 2-variable function by plugging the function value for each input combination in the corresponding square.

Definitions/Notations:

Two K-map squares are considered ***adjacent*** if the input codes they represent have a *Hamming distance of 1.*

A K-map square with a function value *of 1* will be referred to as a ***1-Square***.
A K-map square with a function value *of 0* will be referred to as a ***0-Square***.

The simplification procedure is summarized below:

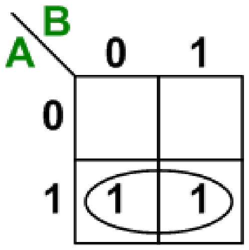
Step 1: Draw the map according to the number of input variables of the function.
Step 2: Fill “1’s” in the squares for which the function is true.
Step 3: Form as big group of ***adjacent 1-squares*** as possible. There are some rules for this which you will learn with bigger maps.
Step 4: Find the common literals for each group and write the simplified expression in SOP.

Example:

Consider the given truth table of two variable function. Obtain the simplified function using K-map.

A	B	F
0	0	0
0	1	0
1	0	1
1	1	1

First draw a 2-variable K-map. The function F is true when AB’ (m_2) is true and when AB (m_3) is true, so a 1 is placed inside the square that belongs to m_2 and a 1 is placed inside the square that belongs to m_3.



Since both of the 1-squares have different values for variable B but the same value for variable A, which is 1, i.e., wherever A = 1 then F = 1 thus F = A.

This simplification is justified by algebraic manipulation as follows:
$F = m_2 + m_3 = AB' + AB = A(B' + B) = A$

To understand how combining squares simplifies Boolean functions, the basic property possessed by the **adjacent squares** must be recognized.

In the above example, the two 1-squares are adjacent with the same value for variable A (A=1) but different values for variable **B** (one square has B=0, while the other has B=1).

This reduction is possible since both squares are adjacent and the net expression is that of the common variable (A).

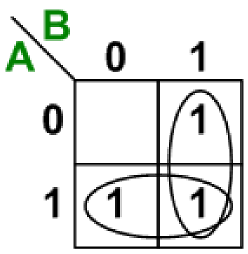
Generally, this is true for any 2 codes of Hamming distance 1 (adjacent). For an *n*-variable K-map, let the codes of two ***adjacent squares*** (distance of 1) have the same value for all variables except the *i*th variable. Thus,

Code of 1st Square: $X_1.X_2......X_{i-1}.\boldsymbol{X_i}.X_{i+1}......X_n$
Code of 2nd Square: $X_1.X_2......X_{i-1}.\overline{\boldsymbol{X_i}}.X_{i+1}......X_n$

Combining these two squares in a group will eliminate the different variable $\boldsymbol{X_i}$ and the combined expression will be
$X_1.X_2......X_{i-1}.X_{i+1}......X_n$
since:
$$\left(X_1.X_2......X_{i-1}.\boldsymbol{X_i}.X_{i+1}......X_n\right)+\left(X_1.X_2......X_{i-1}.\overline{\boldsymbol{X_i}}.X_{i+1}......X_n\right)$$
$$=\left(X_1.X_2......X_{i-1}.X_{i+1}......X_n\right)\left(\boldsymbol{X_i}+\overline{\boldsymbol{X_i}}\right)$$
$$=\left(X_1.X_2......X_{i-1}.X_{i+1}......X_n\right)$$
The variable in difference is dropped.

Another Example:

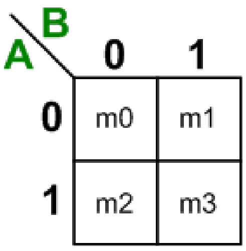
Simplify the given function using K-map method:
$F = \sum(1, 2, 3)$

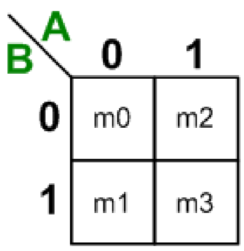


In this example:
$F = m_1 + m_2 + m_3 = m_1 + m_2 + (m_3 + m_3)$
$F = (m_1 + m_3) + (m_2 + m_3) = A + B$

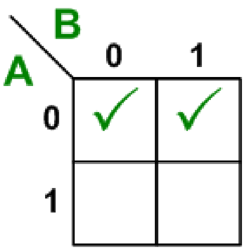
Rule: A 1-square can be member of more than one group.

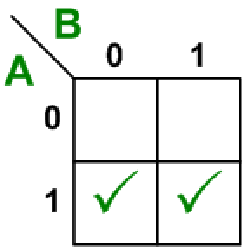
If we exchange the places of **A** and **B**, then minterm positions will also change. Thus, **m_1** and **m_2** will be exchanged as well.

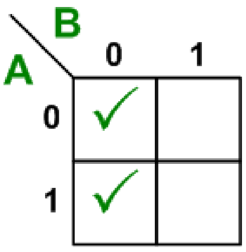


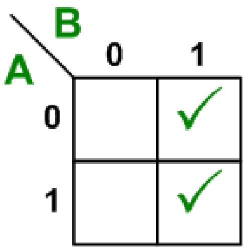


In an *n*-variable map each square is ***adjacent*** to "***n***" other squares, e.g., in a 2-variable map each square is adjacent to two other squares as shown below:

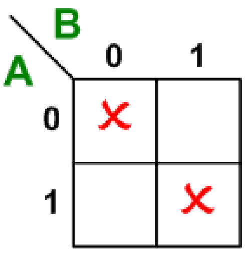


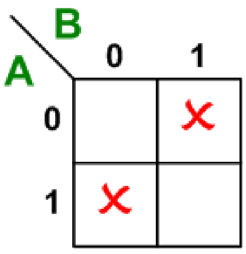






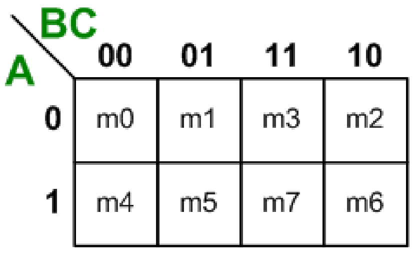
Examples of non-adjacent squares are shown below:





Three-Variable K-Maps:

There are eight minterms for a Boolean function with three-variables. Hence, a **three-variable** map consists of **8 squares**.



A \ BC	00	01	11	10
0	m0	m1	m3	m2
1	m4	m5	m7	m6

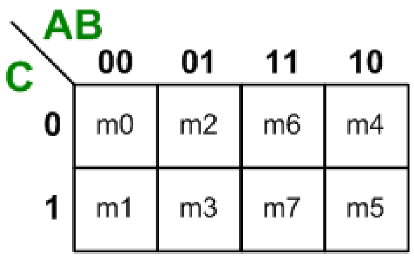
All entries (squares) in the first row correspond to input variable A=0, while entries (squares) of the second row correspond to A=1.

Likewise, all entries of the first column correspond to input variable B = 0, C = 0, all entries of the second column correspond to input variable B = 0, C = 1, all entries of the third column correspond to input variable B = 1, C = 1, while entries of the fourth column correspond to B=1, C = 0.

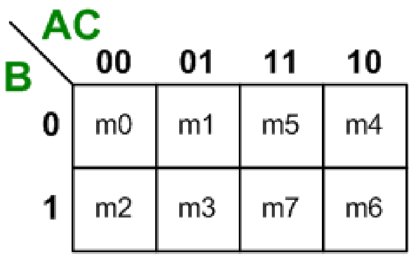
To maintain adjacent columns physically adjacent on the map, the column coordinates do not follow the binary count sequence. This choice yields unit distance between codes of one column to the next (00 – 01—11 – 10), like **Grey Code**.

Variations of Three-Variable Map:

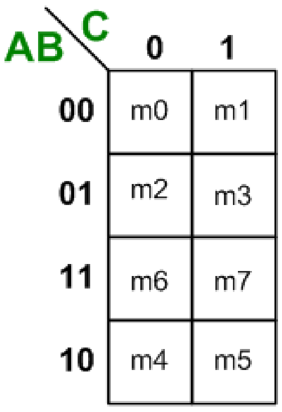
The figure shows variations of the three-variable map. Note that the minterm corresponding to each square can be obtained by substituting the values of variables **ABC** in order.



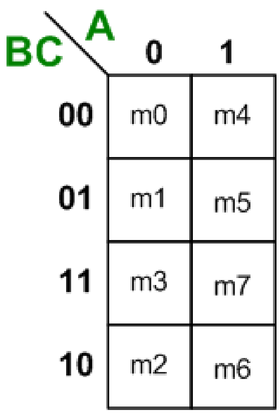
C \ AB	00	01	11	10
0	m0	m2	m6	m4
1	m1	m3	m7	m5



B \ AC	00	01	11	10
0	m0	m1	m5	m4
1	m2	m3	m7	m6



AB \ C	0	1
00	m0	m1
01	m2	m3
11	m6	m7
10	m4	m5



BC \ A	0	1
00	m0	m4
01	m1	m5
11	m3	m7
10	m2	m6

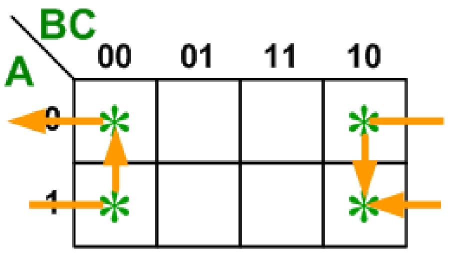
Examples: **(see authorware version)**

There are cases where two squares in the map are considered to be adjacent even though they do not physically touch each other.

In the figure of 3-variable map, m_0 is adjacent to m_2 and m_4 is adjacent to m_6 because the minterms differ by only one variable. This can be verified algebraically:

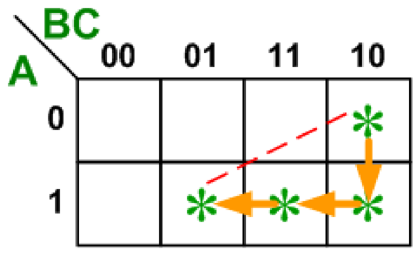
$m_0 + m_2 = A'B'C' + A'BC' = A'C' (B' + B) = A'C'$

$m_4 + m_6 = AB'C' + ABC' = AC' (B' + B) = AC'$



Rule: Groups may only consist of **2**, **4**, **8, 16,**… squares (always power of 2). For example, groups may not consist of 3, 6 or 12 squares.

Rule: Members of a group must have a closed loop adjacency, i.e., **L-Shaped 4** squares do not form a valid group.



Notes:

1. Each square is adjacent to 3 other squares.
2. One square is represented by a minterm (i.e. a product term containing all 3 literals).
3. A group of 2 adjacent squares is represented by a product term containing only 2 literals, i.e., 1 literal is dropped.
4. A group of 4 adjacent squares is represented by a product term containing only 1 literal, i.e., 2 literals are dropped.

Four-Variable K-Maps:

There are **16 minterms** for a Boolean function with **four-variables**. Hence, four-variable map consists of 16 squares.

AB \ CD	00	01	11	10
00	m0	m1	m3	m2
01	m4	m5	m7	m6
11	m12	m13	m15	m14
10	m8	m9	m11	m10

Notes:

1. Each square is **adjacent** to **4** other squares.
2. One square is represented by a minterm (a product of all 4-literals).
3. Combining **2** squares drops **1**-literal.
4. Combining **4** squares drops **2**-literals.
5. Combining **8** squares drops **3**-literals.

Examples: **(see authorware version)**

Rule: The combination of squares that can be chosen during the simplification process in the **n-variable** map are as follows:
A group of **2^n squares** produces a function that always equal to **logic 1**.
A group of **2^{n-1} squares** represents a product term of **one literal**.
A group of **2^{n-2} squares** represents a product term of **two literals** and so on.
One square represents a minterm of ***n* literals**.