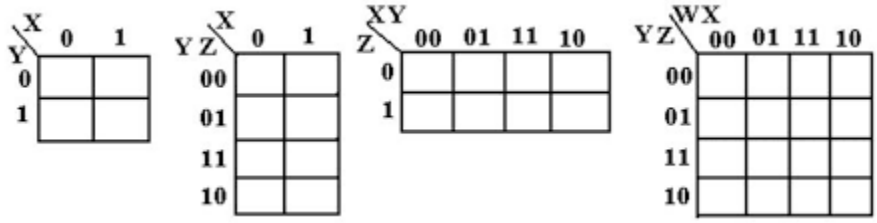
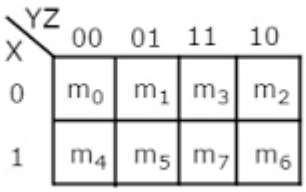
The K – Map simplification

Just like arithmetic expressions the logic expression may be simplified to reduce the cost of implementing the logic circuit. Let us take a common example.

F = ab + ab' + a'b' -- this expression tells that for a two variable (a, b) circuit the output is true in 3 out of possible 4 cases (note: for the case a'b the output is false and hence not considered). Now to implement the logic circuit for F we need i) 1 three input OR gate; ii) 3 two input AND gate and iii) 2 inverters. On the contrary F can be simplified as F = ab + (a+a')b' = ab + b' = (a+b')(b+b') = a + b'; indicating the same with 1 two input OR gate and 1 inverter. For, simple expressions involving a couple of input variables this circuit reduction is easy. However, a simpler graphical method is available to do the same and is known as the K (Karnaugh) – Map simplification technique. Here, a matrix is formed by dividing the variables as 1: 1 (for a two variable map, or 2:1 (or 1: 2) (for a three variable map) or 2: 2 (for a four variable map) as shown below.



Note the row and column numbering – it is following the Gray code (More on Gray code later; for now, keep in mind that in Gray code 00, 01, 11, 10 is the count sequence instead of 00, 01, 10 and 11)

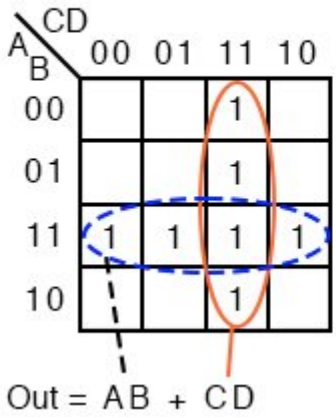


The above K-map for 3-variable is also showing the cell values – read it like XYZ = 000 (the top left cell) = 0; so, the cell is m0. Similarly, the bottom right is 110 (xyz) = 6 and the cell value is m6. Here, 'm', stands for the min-term (more on this later).

One of the characteristics of the Gray code is that between any two successive values only one variable change state. For example (YZ) 00 → 01 (only Z goes from 0 to 1) → 11 (Y changes from 0 to 1 and Z remains at 1) → 10 (Z changes to 0). Now if we combine any two adjacent cells (horizontally) , say cell mm1 and m3 we see it equivalent to F = m1 + m3 =X' Y'Z +X' YZ = X'Z (Y'+Y) = X'Z. In a nutshell, this is the technique for reduction. Note that for F = m1 + m4 [the cells are not adjacent – horizontally or vertically) and we cannot have any reduction. This simple idea may be extended from 2-adjacent cells to bigger cell covers (4-, 8- etc.) maintaining adjacency or folded adjacency (note: m0 and m2 are adjacent – if we fold the map from the middle– assume the map being folded (left to right) from the middle horizontally – then m0 is coming over m2; or vice versa; similarly, m1 is adjacent to m5 if we fold along the imaginary middle from top to bottom. In any case that if the

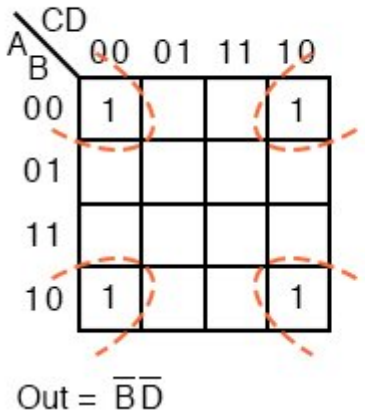
cells have a change in only one variable then they are adjacent. This is the reason for marking the rows and columns using Gray code. Let us take a few examples suing 4-variable maps.

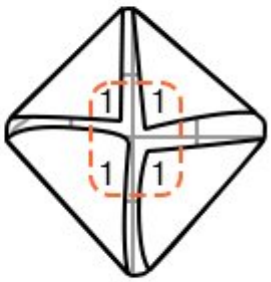
$$\text{Out} = \overline{A}\,\overline{B}CD + \overline{A}B\overline{C}D + ABCD + A\overline{B}CD + AB\overline{C}\,\overline{D} + AB\overline{C}D + ABC\overline{D}$$



Two cell-covers each covering 4-cells are shown in different colours Reduction can be carried out by inspection. For example, in the vertical (orange) cover CD is fixed to 11 but A and B both have changes from 0 to 1 (or 1 to 0). So, the resultant is CD. Similarly, for the blue cover the reduced expression is AB. Thus, the final output is AB + CD.

$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}C\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}C\overline{D}$$

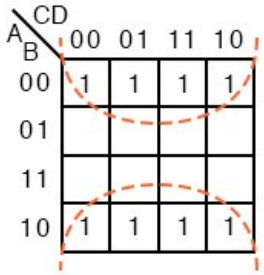




Here, the four corner cells may be taken together to form a 4-cell adjacent cover (shown on the right hand side image). The changes are, on the horizontal side C – goes from 0 to 1 (D remains at 0) and on the vertical side A goes from 0 to 1 while B remains at 0. So, the final output is B'D'.

Take another example;

$$\text{Out} = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}\overline{B}CD + \overline{A}\overline{B}C\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}\overline{C}D + A\overline{B}CD + A\overline{B}C\overline{D}$$

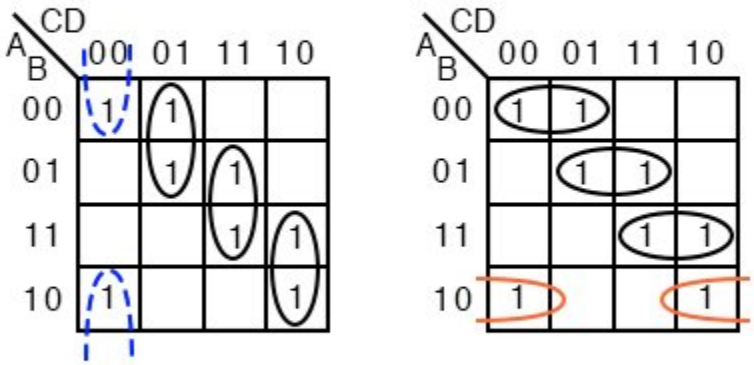


$\text{Out} = \overline{B}$

For a cover of 8 (2^3) you actually get rid of 3 variables (normal or complemented form). So, for a cover of 4 (2^2) you get rid of 2- variables (see the previous example).

The Map cover may have alternative and the reduction is optimal (if done correctly) but may not be unique. Here is one example

$$\text{Out} = \overline{A}\overline{B}\overline{C}\overline{D} + \overline{A}\overline{B}\overline{C}D + \overline{A}B\overline{C}D + \overline{A}BCD + ABCD + ABC\overline{D} + A\overline{B}\overline{C}\overline{D} + A\overline{B}C\overline{D}$$



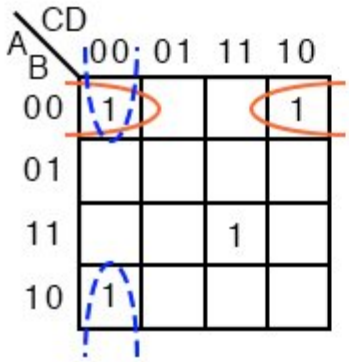
$$\text{Out} = \overline{B}\overline{C}\overline{D} + \overline{A}\overline{C}D + BCD + AC\overline{D}$$

$$\text{Out} = \overline{A}\overline{B}\overline{C} + \overline{A}BD + ABC + A\overline{B}\overline{D}$$

Here, both the solutions are correct and considering the GATE count – they are same. So, there may be different ways to get the covers. However, if you create a 4-cell cover where you have the possibility to go for an 8-cell cover then it is wrong (non-optimal reduction).

We will conclude by showing that there may be isolated cell which cannot be covered.

$$\text{Out} = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,C\,\overline{D} + A\,\overline{B}\,\overline{C}\,\overline{D} + ABCD$$



$$\text{Out} = \overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{D} + ABCD$$

A final point: i) A cover cannot fully enclose any other cover.

ii) For more than 4-variables – using K-map is complicated. And for this this reason another technique, namely, Quine-Mccluskey tabular technique is used (this again is complicated).

The GRAY code

It is a reflective binary code (RBC) or simply RC code suits encoding, particularly for electro-mechanical system, in many applications including K-map row/column numbering as only one digit changes between two successive numbers.

Consider a 4-bit Binary and the corresponding 4- bit number in Gray code

Value	b3	b2	b1	b0		g3	g2	g1	g0
0	0	0	0	0		0	0	0	0
1	0	0	0	1		0	0	0	1
2	0	0	1	0		0	0	1	1
3	0	0	1	1		0	0	1	0
4	0	1	0	0		0	1	1	0
5	0	1	0	1		0	1	1	1
6	0	1	1	0		0	1	0	1
7	0	1	1	1		0	1	0	0
8	1	0	0	0		1	1	0	0
9	1	0	0	1		1	1	0	1
10	1	0	1	0		1	1	1	1
11	1	0	1	1		1	1	1	0
12	1	1	0	0		1	0	1	0
13	1	1	0	1		1	0	1	0
14	1	1	1	0		1	0	0	1
15	1	1	1	1		1	0	0	0

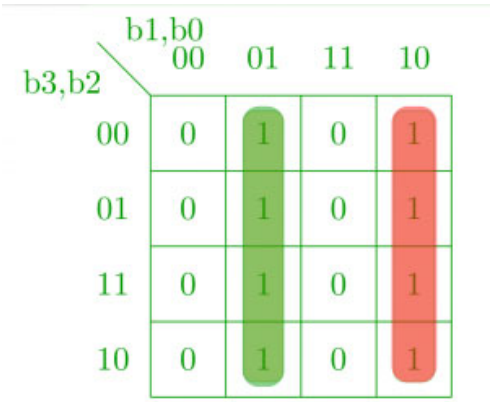
Carefully notice that if you assume an imaginary mid- line between the count value 7 and 8 you would find that g3 has changed from 0 to 1 for the next 8 (entries; i.e., 8 to 15) and g2, g1 and g0 is copied in reverse (reflected) order. See, 7 in Gray is 0 1 0 0 and 8 is 1 1 0 0; Similarly, 9 is 1 1 0 1 and its reflection 6 is 0 1 0 1. So, the horizontal folding applies here at any level. And for the same reason 15 is 1 0 0 0 and 0 is 0 0 0 0 (in Gray) indicating they are adjacent.

BINARY TO GRAY AND GRAY TO BINARY

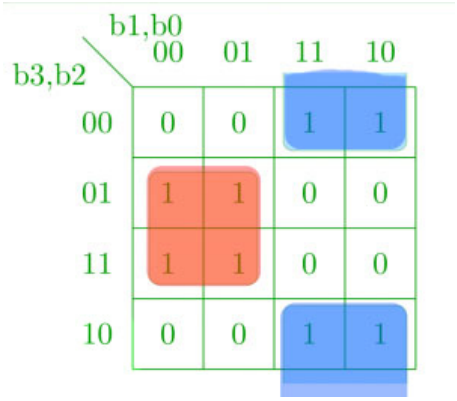
Say, you are designing a 4-bit Binary to Gray converter. So, it is primarily a black-box with four inputs (b3, b2, b1 and b0) and four outputs (g3, g2, g1 and g0).

In order to implement the logic circuitry the above table may be considered the trutj-table for conversion of Binary to Gray. Now, the corresponding K-maps are :

For g0:

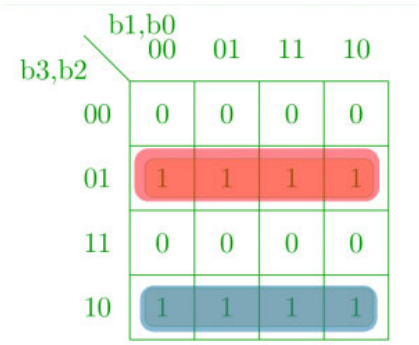


For g1

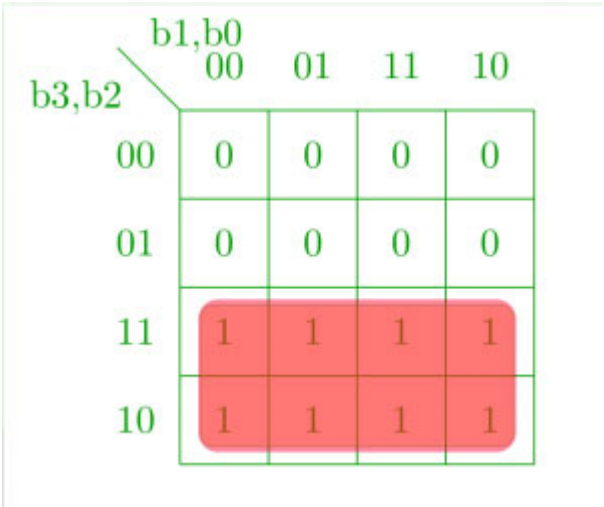


:

For g2:



For g3:



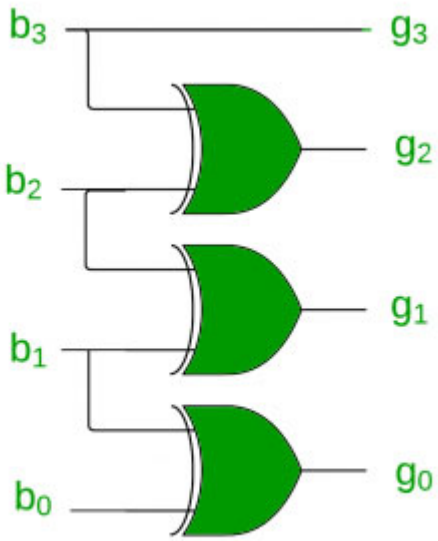
Corresponding minimized expressions are

g0 = b0b1' + b1bo' = b0 xor b1; and similarly;

g1 = b1 xor b2

g2 = b2 xor b3 --- and

g3 = b3 = 0 xor b3; So, we could implement it as shown:



In a very similar manner, the corresponding expressions (Simplified) are:

b3 = 0 xor g3 = g3;

b2 = (0 xor g3) xor g2 = g3 xor g2

b1 = ((0 xor g3) xor g2) xor g1 = g3 xor g2 xor g1

b0 = (= ((0 xor g3) xor g2) xor g1) xor g0 = g3 xor g2 xor g2 xor g1

And the implementation (using 2 – input XOR) is