**Method of Developing Logical Circuits through Truth Tables**

**Overview**

Building logic circuits from truth tables with multiple outputs can be frustrating if not completed in an efficient method.

There are many factors to consider in the relationships of co-dependent inputs and outputs who have more of a direct relationship with some input over others. A circuit that is built through trial and error will most often be inefficient, and require numerous testing to ensure all conditions are satisfied.

To develop logical circuits with complex inputs and outputs, we will take influence from the **Karnaugh Map Method**.

**Karnaugh Map Method (K-map)**

A K-map is a visual truth table used to create algebraic expressions for Boolean functions.

f(A, B, C, D) = A̅BC̅D + ABC̅D̅ + ABC̅D + ABCD + ABCD̅ + AB̅CD + AB̅CD̅

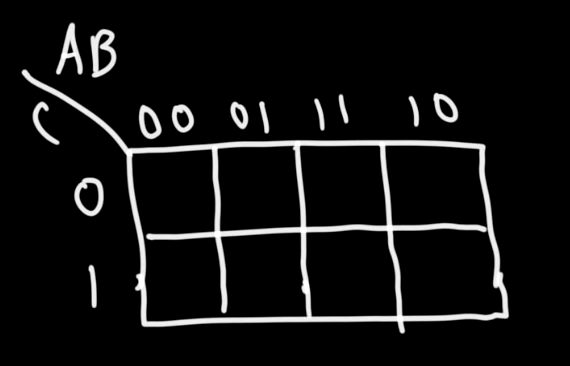
A̅ = the **not** symbol inverts the input value (1 **→** 0; vice versa)



AB = Two inputs multiplied denotes an **and** block



A + B = Two inputs denotes an **or** block

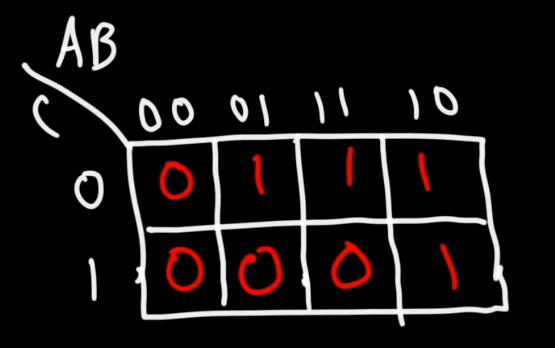
The K-map comes with multiple inputs (I) and expects one output (O) for each combination. The method is started with a basic truth table:

|  |  |  |  |
| --- | --- | --- | --- |
| A (I) | B (I) | C (I) | F (O) |
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 0 |

The inputs are split evenly (when odd number then as evenly as possible) and the K-map is drawn with the amount of possibilities that we have as cells.

Display all possibilities of the binary combinations for each grouping of the inputs. AB has four possible combinations, while C has two. Referencing the truth table, fill in the cells based on the inputs’ results.

The first cell would be the input 000 (AB \* C), while the one to the right of it would be 010.

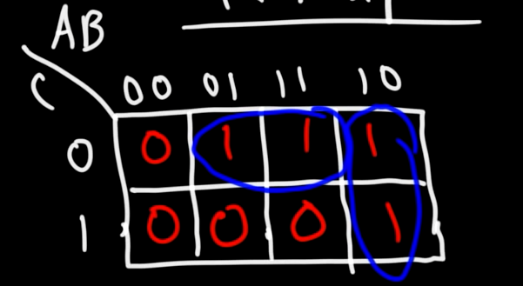


The goal now becomes to take the data, and turn it into a function that can be used to construct the logic circuits.

The next step is to begin grouping the 1’s in the K-map. The amount of 1’s you are able to group must be a number that is the result of any power of 2.

The number of 1’s you can circle:

20 = 1 You cannot circle 3 or 5 consecutive 1’s.

21 = 2

22 = 4

23 = 8

…

Looking at the first pair of horizontally circled 1’s, we must observe the relationship between the result as well as the input.

**B** is 1 in both scenarios, meaning that B is directly dependent on being 1 for those results.   
**C** is 0 in both cases, and therefore C̅ is dependent on being the inverse to get the results.   
**A** is different in both cases, so it is undefined in the first part of the function.

We get BC̅.

Looking at the vertical pair of 1’s, we can deduce that C will not be a part of the equation as it is differing.

**A** is always one independent of other inputs, so A will be included in the equation.  
**B** is always 0 for both cases, meaning that the inverse of B will be included.

We get AB̅.

The equation is **BC̅ + AB̅** for this output.

The equation can be tested by taking a row of inputs from the truth table and substituting them into their corresponding letters. If the result matches the according output on the table, the equation works for that specific instance.

Every combination must work under the equation for it to be valid for solving the output.

**Working with Multiple Outputs (Complex Truth Tables)**

Using the Karnaugh method above, find a separate equation for each of the outputs on the truth table. Once all equations have been found, it is time to start constructing the logic gate.

You can always use a Karnaugh map calculator online to quickly construct the equations.  
[Online Karnaugh map solver with circuit for up to 6 variables (32x8.com)](http://www.32x8.com/index.html)

Let’s say you have three equations…

BCD̅E B̅CDE AB̅CDE + B̅CE

Finding similarities between the equations is how you will make your circuit the most efficient.

**CE** are always positive and together. Grouping them in an **and** gate from the start to form one connection eliminates too many inputs in the long run.

CE(BD̅) CE(B̅D) CE(AB̅D + B̅)

**B̅CDE** is a sub equation of the third equation: ABCD̅E + B̅CE. Branching off from the result of the first equation and multiplying it with A forms one part of the equation.  
**A** is only used once as a positive. Keeping the A port away from the others could keep the circuit free of entanglements.

Then slowly but surely construct all of the equations! Starting from the smaller ones and working your way up will be helpful because logical combination results could become a sub equation.

