# K- Map Minimizer Project

Digital Design 1

Amer Elsheikh 900196010

Under the supervision of Dr. Mona Farouk

## Program Description:

The program achieves 3 main functionalities:

A) Getting and validating the input
B) Printing the K-map
C) Getting the simplified function

Using many modular and generic methods as I will describe:

### A) Getting and validating the input

The user is asked to enter three inputs:

1- The number of variables which should be between 2 and 4 inclusive. If he entered a wrong number, we keep asking him again until he enters a correct one.
2- The number of minterms which should be between 0 and $2^{number\ of\ variables}$ inclusive. If he entered a wrong number, we keep asking him again until he enters a correct one.
3- The minterms themselves where each minterm should be between 0 and $2^{number\ of\ variables} - 1$ inclusive. If the user entered an out-of-range minterm, we keep asking him to enter it again until it is correct. Moreover, if the user entered the same minterm as a previous one, we notify him and ask him to enter the correct minterm again. (That is done through using a set)

All of this is done inside the function "takeInput".

### B) Printing the K-map:

This is done inside the function "printKmap" which prints the proper K-map according to the number of the variables (2, 3, or 4) and then each cell it compares the binary representation of that cell with the minterms given from the user. If that cell corresponds to a given minterm, then we put 1. Otherwise, we put 0.

### C) Getting the simplified function:

This is done using the ideas of Quine–McCluskey algorithm by following these four steps:

**1) Calculating all prime implicants of the map:**

This is done inside the function "calcPrimeImplicants". To do so, I firstly convert all minterms to their binary representation using the helper function "getBinaryRep" which converts. Then, I sort them according to the number of setbits and combine all those with the same number of setbits into groups from the smallest number to the biggest number. Then, I check all elements of every two consecutive groups, and if two implicants differ in only one bit, I use the absorption law to combine those

implicants by replacing the differed bit by "-", then I will have a new set of groups, and I keep simplifying the implicants using the helper function "simplify" until I cannot do more simplification. Then, I will have only the prime implicants.

2) **Getting the essential prime implicants:**
   This is done in the first part of the function "simplifyPrimesAndAnswer". At first, I map every prime implicant I got to the mintems it represents, then I loop through every term and check if it occurred in only one prime implicant, then this prime implicant is essential. If so, I am sure it will be part of the simplified function, so I store it. Moreover, I remove all the minterms it covers from the other implicants as they are already covered, and I then continue searching for other essential prime implicants.
3) **Choosing out of the remaining non-essential prime implicants:**
   This is done in the second part of the function "simplifyPrimesAndAnswer". As I said, I already chose the essential implicants and removed the minterms they cover. Sometimes, there are no more minterms to cover, and we are done. However, sometimes, we still have some minterms to cover through the remaining non-essentail prime implicants. I do so by using trail and error which means I start by choosing one of the remining implicants and see if it covered the remining minterms. If no single one does that, I choose two, then three and so on keeping in mind that I check all possible combinations using C++ stl "next_permutation" function. And, the first time I found all minterms are covered, I will stop iterating, and I will add those implicants to the essential implicants I chose in step 2
4) **Printing the simplified function:**
   Now, that we have all the chosen Implicants, I will just print the simplified function in the last part of "simplifyPrimesAndAnswer" by looping through these implicants and mapping them to the right letters keeping in mind the cases when the function evaluates to 0 or 1.

## Problems in the program:

I covered all cases for the number of variables 2 to 4 inclusive, and I tested the program a lot with no problems at all.

## Instruction for using the program:

- To build the program, you just need to run the cpp file and then use it.
- You can also directly use the provided .exe file which wil take you to the command line interface of the program directly.
- The interface of the program is very easy to use. You just enter the number of variables, then the number of minterms, then those minterms as per the instructions appearing.

Here is an example of the program running: