K-Map Solver: Boolean Function Minimization

Final Project Report
EE1202 - Digital Circuits

Anant Maheshwary
CS24BTECH11006

September 2025

Contents

1 Abstract

Gate-level minimization refers to the design task of finding an optimal gate-level implementation of the Boolean functions describing a digital circuit. It can be thought of as minimizing the number of literals in a Boolean expression.
This minimization task becomes difficult to perform using manual methods when the logic has more than a few inputs.

This project addresses these limitations by developing a simple computer-based logical tool that can minimize a large set of inputs.

The project consists of an interactive web tool, built from scratch to help create **Truth Tables** and visualize **Karnaugh Maps** (for 2–5 variables).
It then solves the KMap to obtain minimized Boolean Functions (in **Sum-of-Products** and **Product-of-Sums** forms)
And finally generates **Verilog** code with an auto-**testbench**.

2 Project Structure/Workflow

1. `index.html` and `script.js` - **Frontend UI** for truth table and K-Map creation. It gathers the current truth table selections and sends a `POST` request to `http://localhost:8080/solve`, then displays **SOP** and **POS** returned by backend. It formats the minimized expressions into a Verilog module and an auto-generated test-bench.

2. `WebServer.java` - **Minimal Java Web Server** . It starts a lightweight `HTTP` server on port 8080 and exposes a `POST /solve` endpoint that the front-end calls when solving the K-Map. It parses the incoming `JSON` from the Truth Table and passes the description of the minterms/dont-care conditions to QM_Minimization.java class

3. `QM_Minimization.java` - **Java Backend** - Implements the Boolean Minimization using the **Quine–McCluskey Algorithm** to find prime implicants and compute the minimal rectangles that cover the Karnaugh Map.

3 Background and Motivation

The general problem of minimizing Boolean functions is **NP-hard**; finding the minimum circuit size for multi-output Boolean functions is computationally difficult and can become infeasible to solve exactly for large inputs. Boolean expressions may be simplified by algebraic means using the laws of Boolean algebra, but this is awkward and lacks a specific algorithm to predict each next step in the manipulative process. We can approach this problem with algorithms like Karnaugh maps and the Quine–McCluskey algorithm to simplify first, and then use heuristic methods for practical applications.

4 Karnaugh Map

Karnaugh Map: It is a diagram made of squares, with each square representing one minterm of the function to be minimized. The minterms are arranged in a sequence similar to Gray code, ensuring that any two adjacent cells differ by only one variable. The simplification power of the K-Map comes from the basic theorem:

$$XY + \overline{X}Y = Y \tag{1}$$

referred to as the **Logical Adjacency theorem**.
Groups of cells are formed in powers of two (single, pair, quad, octet, etc.), where each power of two corresponds to eliminating a variable in the Boolean term.

Figure 1: Insert a diagram of a four-variable K-Map here.

4.1 Don't-Care Conditions

Don't-Care Condition is a combination of variables whose logical value is not specified. The corresponding minterms are marked as **X**'s in the map, and each **X** can be taken as **1** or **0** depending on which choice gives the simplest expression.

5 SOP vs POS Hardware Realization

The simplified expressions produced by the K-Map are usually in one of two standard forms: **Sum of Products (SOP)** or **Product of Sums (POS)**.

- The **SOP** expression is implemented with a group of **AND** gates (one per product term); the outputs of the **AND** gates connect to the inputs of a single **OR** gate.

- The **POS** expression is implemented with a group of **OR** gates (one per sum term); the outputs of the **OR** gates connect to the inputs of a single **AND** gate.

6 Motivation behind the Quine–McCluskey Method

When choosing adjacent squares in a map, we must ensure that:

1. All minterms of the function are covered when combining squares.
2. The number of terms in the expression is minimized.
3. There are no redundant terms.

The procedure for combining squares in the map can be made more systematic using the following concepts:

6.1 Prime Implicants

A **Prime Implicant** is a product term obtained by combining the maximum possible number of adjacent squares in the map.
If a minterm is covered by only one prime implicant, that prime implicant is ***essential***.

7 Quine–McCluskey Algorithm

The Quine–McCluskey algorithm is functionally identical to Karnaugh mapping. The basic Q-M flow is:

1. Find all prime implicants of the function.
2. Construct the prime-implicant chart to find essential prime implicants, as well as other implicants necessary to cover the function.
3. Include all essential prime implicants in the minimal sum.
4. After deleting minterms covered by essential implicants, determine the dominating rows/-columns in the chart, delete the dominated rows/columns and find the new (secondary) essential prime implicants.
5. Repeat the steps 3 and 4 until a minimum cover is found.

7.1 Why is QM preferred over KMaps ?

The tabular form (representation of the prime implicant chart) makes it more efficient for use in computer algorithms. It also gives a deterministic way to check if the minimal form of a Boolean function has been reached.

8 Implementation