Logic Minimization Project

Contributors: Michael Henein, Moaaz Hafez, Shaza Ali

Dr. Mohamed Shalan

Digital design 1 - Fall 2023

Table of contents

How to run the code:

Use gcc to compile the code and then run it.

```
Gcc main.cpp -o main.out

 ./main.out
```

How is the input given:

The input is inserted into the main function as an array of strings.

The string represents a function that uses variables, * and + operators and ‘ for inversion.

Operators are not to be inputted at the beginning of a function.

Spaces are removed from the string at the beginning of execution.

Data Types used:

To complete this project many data types were used, such as:

Vectors: vectors were the main datatype used in the program. This is due to the fact of their adaptable size and built in insertion functions.

Maps: they were used to map the variables to their exact location in the truth table, therefore translating terms such as 101 to ab’c.

Sets: sets were also used to store variable and count the number of variables in the input as sets do not allow multiple occurrences of an input so that every variable is counter once;

Approach:

The following steps are followed to solve the problem using the Quin McCluskey algorithm;

1) (Michael) First, the input is validated by checking for multiple properties regarding the string, such as if the brackets are all set correctly and closed correctly and that every opened bracket is later closed. and the ‘*’ operator is inserted in place on many occasions where it is implicitly inserted, such as

in between a closing bracket and an opening bracket after it, or when to letter variables (operands) are placed directly after each other. Another check that is utilized, is to check whether the character is valid or not as the only operators permitted are the and (*) and the or (+) and the inversion (‘).

2) (Michael) Then the input is checked to see whether it is considered SOP or POS by passing through the string sequentially and checking one character by the other whether the string constitutes one of the above forms.

3) (Michael) The string is then represented in the prefix form for it to be easier to solve later on, this is done by first reversing the string then applying to it a postfix algorithm, then reversing the postfix representation of the infix.

4) (Moaz) Implemented a calculator of any prefix expression using the stack. The calculator loops over on the expression from right to left and pushes elements to the stack until it faces an operator. Then it pops the last two elements, does the calculation, and pushes the result.

5) (Moaz) Implemented multiple variations of DecimaltoBinary functions, dividing by 2 and storing the remainder.

6) (Moaz) Mapped all the input variables to the index that will be used later to identify their column in the truth table. For example, for the boolean expression c’da, c will be mapped to col[0] in the truth table, followed by col [1] = d then col [2] =a.

7) (Moaz) Implemented a search function for the map to inverse the process of mapping. So that when we have, for example, 101 is translated to cd’a.

8) (Moaz) Printed the truth table by mapping the variables, then using the row index to become binary, and storing the binary value of the index of the row to become the row. So, row[3] for 4 variables will contain
0 1 1 1 as their elements. Afterwards, the values are followed by the evaluated function that is provided from the test cases, for example, 0 1 1 1 | 0 means that 0 1 1 1 is evaluated to false.

9) (Moaz) Given the values of the f evaluated in the truth table, implemented functions to display the canonical POS and SOP, therefore identifying the minterms for f and preparing the program to enter to the main implementation of the Quine McClusky method.

10) (Moaz) Implemented a function that counts the number of 1’s in a string so that it is used directly in another function that groups the minterms according to their number of 1’s.

11) (Moaz) Implemented function isAdjacent, which checks if the number of differences between two binary strings = 1. Also, added a function to replace don't cares with '-'. And added a marked map, which maps every minterm to its status whether it is combined or not.

12) (Moaz) Each group is now directly compared with the next group only, and when the function isAdjacent is evaluated to be true, the function recursively reduce() reduces the minterms and replaces the dontcare terms and marks the minterms combined. In the process, it creates maps that map the groups of vectors of the new binary strings (the ones that are replaced with dontcares) to their number of 1's, creating more classified groups to be compared again with each other. When there's nothing isAdjacent anymore, there's no more groups created, then the base case for the recursive function reduce is when the minterm groups is empty.

13) (Moaz) This way, the prime implicants are generated by looping all over a map that stored all minterms generated in the reduce(), then exports the minterms that are not marked yet, i.e., they cannot be combined further and are PI.

14) (Moaz) Printed the PI's binary representation and implemented a vector that stores as well the minterms that it covers.

15) (Moaz) Debugged the code against logical errors, ensuring that the program is smooth and running as intended.

16) Shaza(srtingtominterms) function is made to convert the minterms string of every Prime implicant to vector of strings(minterms).,

17) Shaza(EPI)The existence of every minterm in the prime implicants is checked by looping over all minterms and then over all prime implicants. The essential prime implicant is pushed to the map(kay) when the minterm exsist only in it.The other minterms that exsist in the same essential prime implicant are stored in the map to be used later.

18) Shaza(NotfoundMT)Other minterms that do not exist in the essential prime implicants are produced by traversing all the minterms and checking their existence in the essential prime implicant.

19) Shaza(miniexpress)A table of minterms that are not covered by essential prime implicants is constructed using the previous step. The dominated rows vector is produced by checking which rows are subsets of others. The dominating columns vector is produced by checking which columns are supersets of the others. Dominated rows and columns are then erased from the PI table. Then the prime implicants are sorted by the number of the minterms they have. The prime implicants are included in the solution by their arrangement until all the minterms are covered.Then the essential prime implicants are added to the solution.

20) (Michael) Lastly, file handling is utilized to edit an html file so that Wavedrom can be used to Implement the circuit.