****

**Digital Design 1**

**Dr. Mohamed Shalan**

**Project 1 Report**

**Logic Optimization using Quine-McCluskey Algorithm**

**Name: Ahmed Mohamed Ahmed**

**ID: 900141806**

**Name: Lotfy Hussein Lotfy**

**ID: 900143801**

**11/2/2016**

**Objectives:**

A C++ program that:  
1. Read in (and validate) a Boolean function using its minterms and don’t care   
terms (as decimal numbers).  
2. Generate and print all prime implicants using the Quine-McCluskey tabulation method.  
3. Using the prime implicants generated in part 2, obtain and print all the   
essential prime implicants.

**Tools:**

C++ programming language & IDE. also: iostream, vector, cmath, string, and iomanip libraries.

Quine-McCluskey tabulation algorithm.

**Inputs:**

Number of variables, minterms, and don’t cares.

Decimal representation of minterms and don’t care terms.

**Outputs:**

Minterms.

Prime implicants (in their binary and alphabetical representations) and their number.

Essential prime implicants (in their binary and alphabetical representations) and their number.

Coverage table.

**Procedures:**

1. Reading the requirements and rubrics.
2. Determining the basic skeleton of the project and the data structures to be used.
3. Inputs’ validation.
4. Determining the prime implicants from the implication table and printing them.
5. Determining the essential prime implicants from the coverage chart and printing them.
6. Debugging the code.

**Skeleton Design:**

The main skeleton of the implicant, truth table, implication table, and the coverage table:

* Implicant: a struct is used to store the data of the implicant:

1. An integer element stores the minterms or don't care terms for the first column of the implication table.
2. A vector of decimal minterms/don’t cares for implicants starting from the second column of the implication table.
3. A string to store the binary representation of the implicant as bits and dashes.
4. An integer equal 1 if the implicant ticked, and 0 if not.

* Truth table: minterms and don’t care terms needed to be stored in some data structure. Since they are both treated equally in the implication table and the minterms only are used in the coverage table, minterms and don’t cares are stored as implicants in the implication table vector and minterms are also stored in another 1D dynamic array of decimal minterms.
* Implication table: we first thought about making the table as a 3D dynamic array making a table of columns of groups according to number of 1’s and these groups consist of rows. But after reconsidering it, we made a 1D vector of implicants, that is used to find matches between implicants in the same column. We then get the prime implicants left in this column and reuse the vector for the next column and so on.
* Coverage table: a 2D vector or a vector of vectors of integers determining whether there is a match between each prime implicant’s row and minterm’s column by 0’s and 1’s.

**Inputs’ Validation:**

Inputs:

* Number of variables: for the length of the binary representation of the implicants. Validation: should not exceed 16 as one of the project’s constraints and not less than or equal zero.
* Number of minterms: for input’s loop and length of minterms’ dynamic array. Validation: should not exceed pow(2.0, varNo) or be negative.
* Number of don’t care terms: for input’s loop. Validation: should not exceed pow(2.0, varNo) - minNo or be negative.
* Minterms: by a loop on the number of minterms. Validation: a minterm cannot be negative, cannot exceed pow(2.0, varNo), and cannot be already submitted to the program or repeated.
* Don’t care terms: by a loop on the number of don’t care terms. Validation: a don’t care cannot be negative, cannot exceed pow(2.0, varNo), and cannot be already submitted to the program as a minterm or a don’t care.

**Decimal Integer to Binary String:**

A recursive function, that deals with the least significant bit. Checks if least significant binary digit is 0 or 1, and accordingly appends 0 or 1 in the least significant bit in the string, and recurs on the other bits.

**Alphabetical output:**

Loops on the length of the input vector of implicants to print the implicant. Starting from the most significant bit, which is an A for all functions, prints the alphabetical equal (A, B, …) if the string contains a 1 for the corresponding bit, prints the alphabetical equal (A, B, …) and a " ' " if the corresponding bit equal zero, and prints nothing if the corresponding bit is a dash.

**Prime Implicants:**

The function uses the implication table’s first column as a vector of implicants of minterms/don’t cares, dynamic array of decimal minterms, and returns vector of implicants representing the prime implicants. It first translates decimal terms into binary strings, then pushes each term into the vector of decimal terms in this implicant, and then loops tell it reaches an empty column.

Then, the function counts the difference of bits between each two binary strings for all implicants in the current column. If the difference is equal to one, it ticks the two implicants, pushes the new binary string with a dash in a temporary implicant, combines the two vectors of decimal terms and makes it the vector of the temporary implicant. Checks if this temporary implicant was already pushed into next column, and if not, pushes it.

Last, gets all unticked implicants and checks whether they cover only don’t cares. If not, pushes them into prime implicants vector of implicants.

Then re-loop on next column.

**Essential Prime Implicants:**

The function uses the prime implicants’ vector of implicants, dynamic array of decimal minterms, and returns 1D vector of implicants representing essential prime implicants and 2D vector of integers representing the coverage chart. It first initializes the coverage chart to zeroes. Then, loops on all implicants and minterms as pairs and places a 1 in their intersection if they are matched. Thus, constructing the chart.

Next, the function counts the number of 1’s in each column (minterm). If it is equal to 1, and if the prime implicant it occurs in is not already into the essential prime implicants vector, the functions pushes the prime implicant this minterm occurs in into the essential prime implicants vector.

Last, it prints the essential prime implicants and the coverage table.

**Main function:**

Declares variables, vectors, and dynamic arrays and initializes the needed of them. If there are no minterms, therefore the function is always zero or a contradiction and exits. If not, prints the minterms. If the truth table’s result is always one, that is, minterms number == pow(2, varNo), then the function is a Tautology. If not, gets prime implicants. If there are no prime implicants, prints no primes or essentials and exits. If not, gets essential prime implicants and prints them.

**Code’s Description:**

Language: C++, using Microsoft Visual Studio.

The code is modular by using functions and fully commented.