Minimization of Switching Functions using Quine-McCluskey Method

Vladislav Manojlović
Faculty of Technical Sciences, Kneza Miloša 7,
38220 Kosovska Mitrovica, Serbia

ABSTRACT

The minimization of switching functions is important to reduce the original number of logic gates required to implement digital logic circuits. Quine-McCluskey algorithm is classical method for simplifying these functions which can handle any number of variables. This paper presents Quine-McCluskey algorithm for minimizing switching functions, with additional specific elements, such as starting part (that is decoding DNF form) and cost of circuit. An example of implementation of the algorithm is given too.

Keywords

switching functions, DNF form, cubes, minimization, Quine-McCluskey algorithm

1. INTRODUCTION

One of the aims of synthesis is to obtain a digital circuit which is optimal in relation to certain criteria. In other words, one of the steps of the synthesis consists of determining the simplest algebraic expression which can represents a switching function. In doing this the simplest expression is usually sought in the classes of disjunctive and conjunctive normal forms. The typical criteria which must have the least value in the simplest, so called minimal expression are:

- Number of literals (total number of inputs to all logic gates in the circuit),
- Number of operations (number of logic gates).

These two criteria minimize the cost of circuit. A great number of methods has been developed to simplify the function in order to obtain its minimal normal form. Among them are used algebraic, graphical and tabular methods.

Algebraic method of minimization is slow and error-prone. For these reasons some others procedures have been developed. Karnaugh map provides the ordinary method for simplifying switching functions, although it is limited to the problems with five or less input variables.

When the number of input variables is greater than 5, the tabular method for simplifying switching functions developed by Quine and McCluskey is used. This techniques is suitable also for problems with more than one output. Besides, the Quine-McCluskey method is easier to be implemented as a computer program.

Quine (1952) and McCluskey (1956) have suggested the above method of simplification which is considered the most useful tabular procedure and described in most books for logical minimization. In recent years some modified (simplified) algorithms with higher speed of execution have been observed too.

This paper first gives some basic terms for logical minimization, and then shows the Quine-McCluskey algorithm for simplifying switching functions. Since it starts from DNF form and not from 0-cubes, an algorithm has been developed for automatic decoding DNF. Besides, the costs are calculated for starting and minimal circuit. An example of application of the procedure developed is demonstrated.

Literature has been used for basic terms [1], for Quine-McCluskey algorithm [2] and for realization of the algorithm [2] and [3]. In relation to [3] a different denotation of cube elements has been suggested here.

2. SOME BASIC TERMS

2.1 Prime Implicants

A product term is implicant of a function if the function has the vaule 1 for all minterms of the product term. For function of n variables, implicants may contain n or less literals. The most basic implicants are the minterms. Each minterm of a switching function represents the implicant of that function which covers it on only one vector.

For example, function of two variable $Z = B + \overline{A}\cdot\overline{B}$ is shown in the following truth table.

A	B	Z
0	0	1
0	1	1
1	0	0
1	1	1

There are 5 possible implicants for that function. These are: three minterms, $\overline{A}\overline{B}$, $\overline{A}B$ and AB, and two implicants, which correspond to all posible pairs of minterms that can be joined, that is, $\overline{A}$ and B.

Prime implicant is the implicant which cannot be joined with another implicant to have less literals. In the previous example, $\overline{A}$ and B are simple implicants. Other implicants are covered by one of the simple implicants. Essential prime implicant is prime implicant which covers the implicant not covered by another simple implicant. This means that $\overline{A}$ and B essential simple implicants.

A set of implicants which cover all the the values for which the given function equals 1 is called a cover of that function. Numerous different covers exist for most functions. The examples of covers for the given function are:

- the set of all minterms,
- the set of all prime implicants,

- all the essential prime implicants plus other prime implicants needed to include remaining minterms not included in the essential prime implicants.

For the previous example, $Z = \overline{A} + B$ is minimal cover since it leads to realization with minimum cost.

2.2 Cost of a Circuit

The cost of a logic circuit can be usually expressed as a number of gates plus the total number of inputs to all gates in the circuit. Here it is implied that input variables are available in the right and complementary form. For example, let us determine the cost of a logic circuit which realizes the following function:

$$f = \overline{\overline{x_1\overline{x}_2 + x_3}(\overline{x}_4 + \overline{x}_5)} + x_5x_6$$

According to the formula

Cost of circuit = number of gates + total number of inputs to all gates in the circuit, function f is realized by using three AND gates, three OR gates and two NOT gates, with 14 inputs. Hence, cost of circuit is 3 + 3 + 2 + 14 = 22.

Or, for example, let us determine the cost of logic circuit which realizes the following function:

$$g = \overline{(x_2 + x_3)}(\overline{x}_3 + \overline{x}_4)(x_1 + x_2 + x_3 + x_4)$$

Function g is realized by using three OR gates, one AND gate and one NOT gate. Two of the OR gates have two inputs, and the third has four inputs; AND gate has three inputs. Therefore, cost of circuit is 3 +1 + 1 + 2 * 2 + 4 + 3 + 1 = 5 + 12 = 17.

2.3 Cubical Representation

Each logic function of n variables can be represeted as a subset of 2n vertices of a n-dimensional cube. Each vertex of cubes corresponds to a complete product. The vertices corresponding to complete products are called 0-cubes (null-cubes). If the vertices at which the function has the value 0 or 1 by are separately denoted, for example, 0 by an empty little circle and 1 by a full circle, then we have the possibility of setting the logic function geometrically.

Two null-cubes form 1-cube (line) if they differ only by one variable (coordinate). In a similar manner, a set of four 0-cubes forms 2-cube (square) if its coordinates differ only in two variables, etc.

For example, let us consider the switching function

$$Y = \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + A\overline{B}\,\overline{C}\,\overline{D} + AB\overline{C}D + ABCD$$

Through the set of cubes this function can be represented in the following way:

Y(1) = {0000, 1000, 1101, 1111}

The function can simplified as

Y(1) = {x000, 11x1}

The cube x000 is corresponding for $\overline{B}\,\overline{C}\,\overline{D}$. Here 1 represents the variable, 0 the complement to the variable, and x the absence of the variable. The cube 11x1 corresponds to the member ABD, cube 1111 corresponds to the member ABCD, etc.

So, the minimal solution is Y(1) = {x000, 11x1} that is $Y = \overline{B}\,\overline{C}\,\overline{D} + ABD$.

This expression has less products and less literals than the starting expression.

In this way the cubical representation of switching functions can be used for simplifying switching functions.

3. QUINE-McCLUSKEY ALGORITHM

The starting point for Quine-McCluskey method is truth table, or equivalently, the list of binary or decimal indexes of the function. If the function is given differently, it must be first transferred into the form required. For example, DNF expression $f(A,B,C,D) = \overline{A}\,\overline{B} + ABC$ is transferred into its canonical form

$$\begin{aligned} f(A,B,C,D) = {} & \overline{A}\,\overline{B}(C+\overline{C})(D+\overline{D}) + ABC(D+\overline{D}) = \\ & \overline{A}\,\overline{B}\,\overline{C}\,\overline{D} + \overline{A}\,\overline{B}\,\overline{C}D + \overline{A}\,\overline{B}C\overline{D} + \\ & \overline{A}\,\overline{B}CD + ABC\overline{D} + ABCD \end{aligned}$$

that is

f(1) = {0000, 0001, 0010, 0011, 1110, 1111}

Quine-McCluskey method is based on the procedure of grouping applied to every two minterms which differ only by the value of one variable. For example, $ABCD + ABC\overline{D}$ is reduced to ABC. Since ABCD and $ABC\overline{D}$ are represented by 1111 and 1110, the simplified form can be represented by 111*, where * denotes the missing variable. Thus, in this way the number of terms and number of variables are reduced.

Two main parts in the Quine-McCluskey algorithm are:

- Finding all prime implicants of the function.
- Use those prime implicants in a prime implicant table to find the essential prime implicants of the function and other prime implicants that provide the coverage of the function with minimum cost.

More precisely, Quine-McCluskey algorithm has the following steps:

I. Input:

Step 1: Enter input of switching function as DNF form.

Stop 2: Form table to enter 0-cubes for minterms from input data.

Step 3: Calculate cost of starting circuit as number of gates plus the total number of inputs to all gates in the circuit.

II. Finding the Prime Implicants:

Step 1: Arrange 0-cubes in increasing sequence of number of units. Divide the table intro classes of cubes so that each class contains the cubes with the same number of units, and separate the classes by a horizontal line.

Step 2: From the starting table form a new table by grouping 0-cubes from adjacent classes differing only an one bit. In the new table the grouped cubes on the differing bit has the symbol "*"", and in the previous table the grouped cubes are denoted by the symbol "√ ".

Step 3: Using step 2 form a new table and repeat the procedure as long as the pairing is possible. All non-paired cubes correspond to simple implicants. If in grouping the same reduced terms appear several times, they are entered into the new table only once.

III. Finding the Essential Prime Implicants:

Step 1: Form the coverage table where the simple implicants found enter the lines and the minterms enter the columns. Every minterm covered by a given simple implicant is denoted by a unit in the corresponding position.

Step 2: Identify essential simple implicants in the table for the columns containing only one unit.

Step 3: If after the previous step all the minterms are not covered, form a new coverage table from remaining simple implicants and minterms not covered yet. Choose such a simple implicant to cover most of the remaining minterms.

Step 4: Repeat step 3 until all the minterms are covered.

IV. Display:

Step 1: Display all separated simple implicants in all iterations of the reduction. In doing this, a number of minimal sets of simple implicants can be formed, which depends on the way of coverage and then one of them is accepted as a solution.

Step 2: Calculate the cost of minimum circuit.

4. IMPLEMENTATION

The algorithm is realized by using Fortran PowerStation language. With an adequate program, it is possible to achieve minimization of switching functions with maximum 10 input variables and maximum 50 product terms.

In order to check it for exact operation, the program has been tested on minimization of several switching functions, each including a different number of input variables. The results tested were then checked by hand. In all cases good results were obtained.

The program requires a textual file which defines DNF for function. The sign in the first column denotes the type of data: “c“ - comments, “y” – switching function in algebraic form and “e” – end of program.

Independent variables are denoted form by a, b, ... , j. Operators are conjunction (·), disjunction (+) and negation (^).

In addition to the above cited correct symbols for independent variables and operators, the following table shows some other correct symbols for input records.

correct special symbol	meaning
,	end of record
,,	continuation for the next record
blank	not to be considered

DNF expression is read in and decoded, and cube table determined for the vectors at which y = 1.

Each term in the DNF expression is represented by cube. The cube of n variables (n is up to 10) is represented by data structure with n elements, each of them having three possible values:

1	non-complementary variable
0	complementary variable
*	variable does not occur

DNF expression being decoded consists of symbols of which each can be a variable, operator or special symbol. The algorithm for decoding DNF expression consists of the following steps:

1. Set I = 3.
2. Test Ith symbol.
3. If disjunction, check if the product term number is greater than 50, if it is not go on to the new symbol. If it is yes, algorithm is stopped.
4. If conjunction, go on to the new symbol.
5. If complement, then IZN = ’0’ and go on to the new symbol.
6. If comma, go on to the new record.
7. If the variable is incorrect, call subprogram POGR to enter into it the correct variable and then test the symbol entered.
8. If the variable is correct, then IMATR(I, J) = IZN, IZN = ‘1’ and go on the new symbol.
9. If Ith symbol = ‘,’, algorithm is finished; if not, then I=I+1 and go to step 2.

Before beginning step 1, the matrix is deleted, that is, IMATR(I, J) = ‘*’ and IZN = ‘1’. The algorithm given does not consider continuation for the next record.

Figure 1 shows input data for DNF expression for the example from Section 5. Figure 2 shows the corresponding cube tables.

```
c
y = ^a^c.d+^a.b.c^d+^a.b.c.d+a^b.c.d
     +a.b^c^d+a.b.d,
end
```

Figure 1. Input data for the given example

CUBE TABLE

a	b	c	d
0	*	0	1
0	1	1	0
0	1	1	1
1	0	1	1
1	1	0	0
1	1	*	1

Figure 2. Cube table for the given switching function

0-CUBE TABLE

a	b	c	d
0	0	0	1
0	1	0	1
0	1	1	0
0	1	1	1
1	0	1	1
1	1	0	0
1	1	0	1
1	1	1	1

Figure 2. 0-cube table for the given switching function (continuation)

5. EXAMPLE

A simple example with four variables will be considered here. It can be written in SDNF form as follows:

Y = ^a^b^c.d+^a.b^c.d+^a.b.c^d+^a.b.c.d+a^b.c.d+

a.b^c^d+a.b^c.d +a.b.c.d

Applying Quine-McCluskey algorithm the following results are obtained:

I. Input

1. Enter input in the form of given expression.
2. Binary indexes (0-cubes) which correspond to minterms are 0001, 0101, 0110, 0111, 1011, 1100, 1101, and 1111.
3. Cost of circuit is:

   eight AND gates + one OR gate + (8 x 4) inputs in AND gates + eight inputs in OR gate = 9 gates + 40 inputs in gates = 49

II.

1. Arranged 0-cubes in increasing sequence of number of units are shown in table 1.

Table 1

i	P_i	
1	0001	√
5	0101	√
6	0110	√
12	1100	√
7	0111	√
11	1011	√
13	1101	√
15	1111	√

2. 1-cubes obtained by pairing the cubes of 0th order are shown in table 2. In the starting table all 0-cubes are denoted as paired, which means that there are no simple implicants among them.

Table 2

I,j	$P_{i,j}$	
1,5	0*01	A
5,7	01*1	√
5,13	*101	√
6,7	011*	B
12,13	110*	C
7,15	*111	√
11,15	1*11	D
13,15	11*1	√

3. 2-cubes obtained by pairing the cubes of 1th order are shown in the table 3.

Table 3

i,j,k,l	$P_{i,j,k,l}$	
5,7,13,15	*1*1	E
5,13,7,15	*1*1	F

There is no possibility of further pairing. All non-paired implicants (here denoted by E, A, B, C and D) form a set of simple implicants. So, the simple implicants are *1*1, 0*01, 011*, 110* and 1*11.

III.

1. Coverage table is shown in table 4.

Table 4

	0 0 0 1 0 1 1 1
	0 1 1 1 1 0 1 1
	0 0 1 0 1 1 0 1
	1 1 0 0 1 1 1 1

E	0 1 0 0 1 0 1 1
A	1 1 0 0 0 0 0 0
B	0 0 1 0 1 0 0 0
C	0 0 0 1 0 0 1 0
D	0 0 0 0 0 1 0 1

2. In this step are found essential prime implicants (here denoted only by a single unit in the column). These are simple implicants which are not covered by other simple implicants In this example, the essential simple implicants are A, B, C and D, that is, 0*01, 011*, 110* and 1*11.
3. After the previous step here are no minterms which are not covered.

IV.

In the beginning, the given switching function had the form

0001 + 0101 + 0110 + 1100 + 0111 + 1011 + 1101 + 1111

which is equivalent to the given expression.

After applying the procedure, a simpler form is obtained:

0*01 + 011* + 110* +1*11

or

f =^a^c.d+^a.b.c+a.b^c+a.c.d

Cost of circuit = four AND gates + one OR gate + (4 x 3) inputs in AND gates + four inputs in OR gate = 5 gates + 16 inputs in gates = 21

Note that the same result was obtained by applying Karnaugh map method.

5. CONCLUSION AND FUTURE WORK

In the paper, the Quine-McCluskey algorithm for logic gate minimisation is described. The experience so far has shown that the program developed is reliable and fast.

This program minimizes switching functions with maximum 10 variables. The code can easily be widened to correspond to the functions whose number of variables is restricted by memory of the computer. Development and incorporation of user interface is also suggested for future work.

REFERENCES

[1] Brown, S. and Vranesic, Z. 2000. Fundamentals of Digital Logic with VHDL Design, McGraw-Hill.

[2] Tomaszewski, S., Celik, I., Antoniqu, G. 2003. WWW-based Boolean function minimization, Int. J. Appl. Math. Comput. Sci., Vol. 13, No. 4, 577-583.

[3] Manojlovic, V. 2013. Cubical Representation of Switching Functions, Simposium YU-INFO 2013, Kopaonik.