A Novel Approach to Simplifying Boolean Functions of Any Number of Variables

T Mathialakan
Department of Physical Science
Vavuniya Campus, University of Jaffna,
Jaffna, Sri Lanka
mathialakan@mail.vau.jfn.ac.lk

Dr S Mahesan
Department of Computer Science
Faculty of Science, University of Jaffna,
Jaffna, Sri Lanka
mahesans@jfn.ac.lk

***Abstract*- Expressions of Boolean functions in the minimal form would be essential for many needs such as hardware designs. There are several ways such as K-map technique and tabular method of Quine-McCluskey to simplify the Boolean expressions. These currently used techniques have drawbacks such as the limitation on number of variables and dependence on 'minterms'. Hence, these methods do not adapt programming perfectly. A new method has been introduced in this paper to minimise the Boolean functions without considering the minterms. This method deals with the sum of product (SOP) expressions – it takes the input as SOP and gives the output as SOP. Each product in SOP is encoded in a novel way and represented in a row of a table where the columns correspond to the variables involved in the expressions. The encoded products are taken pair by pair and an appropriate rule of the set of four sound rules is applied to simplify. The encoding is used to select the most promising pairs to apply the rules in a systematic way. This process is continued until no further pair selection is possible. This novel idea makes possible that the expression can contain any number of variables without increasing the complexity of the simplification process though it needs a little more work in encoding and in selecting the most promising products. Also, this idea can be easily programmed as the algorithm is very systematic. The implementation of the algorithm in C# and testing proves that the idea works well efficiently though looks simple. In fact, no similar idea is reported to our knowledge**.

Keywords- Boolean function, Boolean expression, K-map, minterm, sum of product (SOP), tuple.

I. INTRODUCTION

Boolean expression expressed the logical functions in terms of logical variables. Values taken by the logical functions and logical variables are in the binary form. Any logical variable can have only one of the two values 0 and 1, and any logical variable (binary variable) may appear either in its normal form (A) or in its complemented form (A'). Minimisation of Boolean function is one of the major tasks in digital design process with considering the number of gates used, physical space covered and power consumption. The traditional way of minimisation process such as K-Map and Tabular is used with applying of basic axioms and theorems. [1] Further, these methods are found to be dependent on minterms of all the variables involved, thus makes it time consuming even converting the given expression into a sum of minterms.

For example, the Boolean expression $A'BC + A + AB$ of three variable has to be converted into $A'BC + ABC + ABC' + AB'C' + ABC + ABC'$ by replacing A by $ABC + AB'C + AB'C'$ and AB by $ABC + ABC'$, whereas in the method proposed in this paper handles the expression in the given form by way of introducing encryption by giving 1 for a variable for its unprimed form and 0 for its primed form and -1 for its absence.

Hence, an algorithm is designed for simplification of Boolean expression without obtaining minterms to simplify in a systematic way and thus to implement in a high-level language.

II. METHODOLOGY

The expression is given as a sum of products, not necessarily in minterms, from which the products are extracted and represented in its encryption form in a table row by row.

A new technique is learned by analysing the Ideal theorems and proper rules of Boolean algebra. The product terms in the encrypted form are compared pair by pair and reduced into simpler form as much as possible.

For example, consider the input expression $A'BC + A + AB$.From this expression, the product terms $A'BC$, A, and AB are identifying the occurrences of three variables A, B, C, the products are encrypted into following records respectively: [0, 1, 1]

[1, -1, -1]

[1, 1, -1]

The size of a record resulting from the encryption depends on the number of variables involved in the given expression, each number in a record corresponds to the variables involved: 1 representing the unprimed form, 0 representing primed form and -1 its absence.

The following tabular representation makes this fact clearer:

Record	Variables		
	A	B	C
1	0	1	1
2	1	-1	-1
3	1	1	-1

The next step is to introduce a measure called *index* to each row representing the number of 1s occurred in that row. [1] The indices for the above example are 2, 1, and 2 respectively. Then, the records are reordered in the ascending of indices.

The rows are compared in order pair by pair and do reduction/elimination as appropriate and possible by considering the following identities of two variables:

1. $xy + x'y = y$
2. $x + x'y = x + y$
3. $xy + x' = x' + y$
4. $x + xyz = x$
5. $x + x' = true$

How we apply these rules: Let us see one by one of the above five rules.

Case 1 (Logical adjacency): $xy + x'y$ corresponds to two records, namely,

1	1
0	1

In which one column has 1 and 0 while the other column has the same bit 1. In such a case, these two rows will be replaced by a new row -1, 1 which corresponds to y.

Case 2 (Absorption): $x + x'y$ which corresponds to the records

1	-1
0	1

Here one column has 1 and 0 and the other has -1 and 1. In such a case, the bit which diagonally opposite to -1 will be replaced by -1 resulting in

1	-1
-1	1

corresponding to $x + y$.

Case 3 (Absorption): $xy + x'$ - this case is similar to case 2, resulting in

-1	1
0	-1

corresponding to $x' + y$

Case 4 (Absorption):$x + xyz$ corresponds to

1	-1	-1
1	d	e

d, e can be any encryption number.

In this table, one column has the same bit 1 whereas all the other encryption number is -1 in one row. In such a case, the row other than the one which has all entries -1 except for that in one column will be removed.

Case 5 (Complement):$x + x'$ corresponds to

1
0

in which one entry is 1 and the other is 0, resulting in the Boolean value true, and further processing is not needed.

The above idea can be extended to any number of variables.

A. Algorithm

The records are stored in a matrix $m[][]$, the size of whichis p times q, where p is the number of product terms and$q$ is the number of variables. The matrix is sorted in the ascending order indices of each record - where `index' means the number of 1s in the record.

Simplification is performed as follows:

$row1 = 1$

while ($p - 1$) do

{

$row2 = row1 + 1$

while (p) do

{

compare $row1$ and $row2$

let $c10$ = number of occurrences such that $m[row1, j] == 1$ and $m[row2, j] == 0$ for some j
+ number of occurrences such that $m[row1, k] == 0$ and $m[row2, k] == 1$ for some k

$c1\_1$ = number of occurrences such that $m[row1, j] == 1$ and $m[row2, j] == -1$ for some j

+ number of occurrences such that $m[row1, k] == 0$ and $m[row2, k] == -1$ for some k

c_11 = number of occurrences such that $m[row1, j] == -1$ and $m[row2, j] == 1$ for some j

+ number of occurrences such that $m[row1, k] == -1$ and $m[row2, k] == 0$ for some k

$c11$ = number of occurrences such that $m[row1, j] == 1$ and $m[row2, j] == 1$ for some j

+ number of occurrences such that $m[row1, k] == 0$ and $m[row2, k] == 0$ for some k

if ($c10 + c1\_1 + c\_11$ is 1 and it occurs in column j)

set $m[row1, j] = -1$,

remove row2 (this row2 will not be included after this step)

set $row1 = -1$

break inner loop

else if ($c10 == 1$ & $c1\_1 > 0$ & $c\_11 == 0$ and it $c10$ occurs in column$j$)

set $m[row1, j] = -1$

set $row1 = -1$

break inner loop

else if ($c10 == 1$ & $c1\_1 == 0$ & $c\_11 > 0$ and it $c10$ occurs in column$j$)

set $m[row1, j] = -1$

set $row1 = -1$

break inner loop

else if ($c10 == 0$ & $c1\_1 == 0$ & $c\_11 > 0$ and it $c10$ occurs in column*j*)

remove *row2*

else if ($c10 == 0$ & $c1\_1 > 0$ & $c\_11 == 0$ and it $c10$ occurs in column*j*)

remove *row1*

}

}

Decrypt the matrix *m* into final simplified Boolean expression.

Let us consider another example to see how this algorithm works:

The input expression is $xy' + y + yz + yzw + z$

The matrix $m[][]$ and the index will be as follows corresponding to the five products:

Record	Variables				Index
	x	y	z	w	
1	1	0	-1	-1	1
2	-1	1	-1	-1	1
3	-1	1	1	-1	2
4	-1	1	1	1	3
5	-1	-1	1	-1	1

This matrix is sorted in the ascending order of index as follows:

Record	Variables				Index
	x	y	z	w	
1	1	0	-1	-1	1
2	-1	1	-1	-1	1
3	-1	-1	1	-1	1
4	-1	1	1	-1	2
5	-1	1	1	1	3

During the process, when record 1 and record 2 are compared, the counts $c10, c1\_1, c\_11$ and $c11$ will be

c10	c1_1	c_11	c11
1	1	0	0

and since $c10 == 1, c1\_1 > 0, c\_11 == 0, c10$ is 1 corresponding to variable y in column 2, and $m[2,1] == -1$, so the diagonal of $m[2,1]$, $m[1,2]$ is set to be -1, where row 1 has unprimed *x*. The resulting matrix will be

Record	Variables			
	x	y	z	w
1	1	-1	-1	-1
2	-1	1	-1	-1
3	-1	-1	1	-1
4	-1	1	1	-1
5	-1	1	1	1

Then compare records 1 and 3, the corresponding counts are

c10	c1_1	c_11	c11
0	1	1	0

In this case, no action can be taken.

Then compare records 1 and 4, the corresponding counts are

c10	c1_1	c_11	c11
0	1	2	0

In this case also, no action can be taken.

Then compare records 1 and 5, the corresponding counts are

c10	c1_1	c_11	c11
0	1	3	0

In this case also, no action can be taken.

Then, in the next iteration, records 2 and 3 are compared and the corresponding counts are

c10	c1_1	c_11	c11
0	1	1	0

In this case, there are no action can be applied. Then compare records 2 and 4, the corresponding counts are

c10	c1_1	c_11	c11
0	0	1	1

and since $c10 == 0$, $c1\_1 == 0$, and $c\_11 > 0$, the record 4 will be removed. The resulting matrix will be

Record	Variables			
	x	y	z	w
1	1	-1	-1	-1
2	-1	1	-1	-1
3	-1	-1	1	-1
5	-1	1	1	1

Then records 2 and 5 are compared and the corresponding counts are

c10	c1_1	c_11	c11
0	0	2	1

and since $c10 == 0$, $c1\_1 == 0$, and $c\_11 > 0$, in the similar way of previous step the record 5 gets removed. The resulting in a new matrix as

Record	Variables			
	x	y	z	w
1	1	-1	-1	-1
2	-1	1	-1	-1
3	-1	-1	1	-1

Since no more possible comparisons, this method is stopped with the resultant matrix:

Record	Variables			
	x	y	z	w
1	1	-1	-1	-1
2	-1	1	-1	-1
3	-1	-1	1	-1

The best minimal expression will be: $x + y + z$

III. RESULTS AND DISCUSSION

The algorithm is implemented and tested by giving several Boolean expressions of varying complexity. The simplification resulted in the best minimal form possible

for the expression considered. The examples range from a simple one like $x'y + xy + y'$ to a complicated one with several variables like $dr' + r + ma' + a + he' + ef' + ef + sa'g' + sag' + sag + ma't + mat + m't + h'i + hi + bo'l' + lyz + lyz' + ly' + ok' + okn' + okn + expr' + exp' + p + r + e'x$.The corresponding simplified expressions1(true) and $d + r + m + a + h + e + s + p + i + b + t + x + l + o$ are shown in Fig. 1 and Fig. 2 respectively – the snapshot of the output of the program.

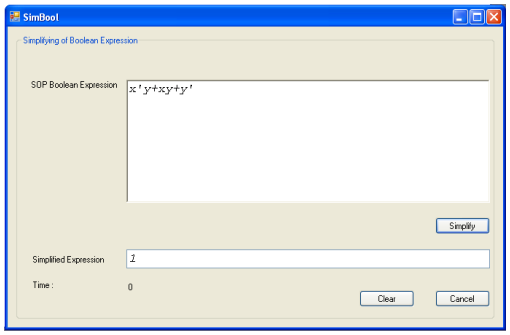

Figure 1. Snapshot of the output for the expression $x'y + xy + y'$

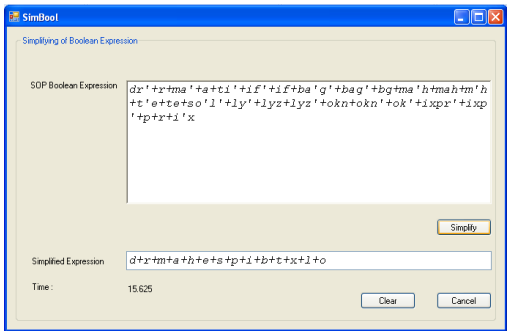

Figure 2. Snapshot of the output for the expression $dr' + r + ma' + a + he' + ef' + ef + sa'g' + sag' + sag + ma't + mat + m't + h'i + hi + bo'l' + lyz + lyz' + ly' + ok' + okn' + okn + expr' + exp' + p + r + e'x$

The following table shows 10 sample expressions and their reduced form produced by the program:

TABLE 1: TEST CASE OF EXPRESSIONS WITH VARIED NUMBER OF VARIABLES, OBTAINED REDUCED FORM AND TIME TAKEN.

Expression	Number of variables	Time taken (ms)	Reduced form	Further reduction possible
$x'y + xy + y'$	2	0	1	No
$x'yz + xy + z'y$	3	0	y	No
$x'yz + x'yw' + x'y'z + xyz + xyw' + y'z + yw$	4	0	$y + z$	No
$ab'c + ab'c' + bcde + bcd' + bce' + bc' + cd' + de' + e$	5	0	$a + b + c + d + e$	No
$abcd' + cdef' + efba' + abd + abc' + cdf + cde' + efa + efb'$	6	0	$ab + cd + ef$	No
$ad' + d + bcdg + bcd'g' + g + a'ef + aef + abhi + a'hi + b'hi$	9	0	$a + bc + d + ef + g + hi$	No
$a'bcd + a'bc' + a'bd' + c'de + c'de' + e'fgh + e'fgh' + e'fg' + g'hijk + g'hj + g'hj' + i'j + ab'c'dk'l + abc'dk'l + a'c'dk'l + cdk'l + d'k'l$	12	15.625	$i'j + c'd + g'h + a'b + e'f + k'l$	No
$ao + b'n + cm + d'l' + ek + fj' + gi + h + aoe'k' + aoe'k + aoe + b'nh'i + b'nh'i' + cmd + cmd' + d'l'f'j + d'l'fj + d'l'j' + efk + ef'k + fgij' + gij + f'gi + gij + fj'h' + h$	15	15.625	$fj + h + b'n + d'l' + ao + cm + ek + ig$	No
$higate + higate' + higat' + hig'a + hia' + a + i'mbc + i'mb'c + i'mc' + tinyd'f' + tinyd'f + tinyd + expr's' + expr + exps$	18	15.625	$hi + i'm + a + tiny + exp$	No
$dr' + r + ma' + a + he' + ef' + ef + sa'g' + sag' + sag + ma't + mat + m't + h'i + hi + bo'l' + lyz + lyz' + ly' + ok' + okn' + okn + expr' + exp' + p + r + e'x$	21	15.625	$d + r + m + a + h + e + s + t + i + b + o + l + x + p + r$	No

REFERENCES

[1] Singh, A.K., Manish Tiwari, and Arun Prakash, Digital Principles Switching Theory. New Delhi: New age International Publishers, 2006. ch.2,3

[2] Matt Telles, and Kogent Solutions Inc, C\# 2005 Programming, India: Dreamtech Press, 2008.

[3] James, L. Hein. (2004). Discrete Structures, Logic, and Computability, New Delhi: Narosa Publishing House, 2004, pp.572-581

Dr S Mahesan graduated from University of Jaffna, specialising in Statistics, did M.Sc. in Computing at Cardiff University of Wales, UK, and then obtained a Ph.D. in Computer Science from University of Wales. He is interested in range of fields in Computer Science: Theory of Languages, High Performance Computing, Numerical Computing, Knowledge Representation, Natural Language Processing, Machine Learning, Image Processing, and Bio Informatics.

T Mathialakan was born in Alaveddy, Jaffna, Sri Lanka. He graduated from University of Jaffna, Jaffna, Sri Lanka, specializing in Computer Science. He has been working as a lecturer at the Department of Physical Science, Vavuniya Campus of the University of Jaffna since 2008. Before attached this department he served as an assistant lecturer at the Department of Computer Science, Faculty of Science. He is interested in Satellite Image Processing, Numerical Computing, Scientific Computing and High Performance Algorithms.