COMPUTER AIDED MINIMIZATION PROCEDURE FOR BOOLEAN FUNCTIONS

Nripendra N. Biswas

Indian Institute of Science
Bangalore 560012, India

ABSTRACT — The paper describes CAMP, a Computer Aided Minimization Procedure for Boolean functions. The procedure is based on theorems of switching theory and fully exploits the power of degree of adjacency. The program does not generate any superfluous prime implicant and all the essential and selective prime implicants are chosen with no or minimum iteration. For shallow functions consisting mainly of essential prime implicants (EPIs) and a few selective prime implicants (SPIs), CAMP produces the exact minimal sum of product form. For dense functions consisting of a large number of inter-connected cyclic SPI chains, the solution may not be exactly minimal, but near minimal.

I. INTRODUCTION

With the advent of very large scale integration (VLSI), and the extensive use of custom designed Programmable Logic Arrays (PLAs) in VLSI circuits the subject of computer-aided minimization of Boolean functions, particularly for large number of variables, has assumed special significance. Most of the earlier methods depend on the philosophy of Quine and McCluskey [1,2,3], where all the prime implicants of the function are first calculated, and then a subset of these prime implicants is found to cover the function. In a paper in 1971 [4], Biswas introduced the concept of *degree of adjacency* and showed how the essential prime implicants can be determined without the help of a cover table. Thereafter many authors [5,6,7,8] utilized this concept to develop fast techniques for Boolean function minimization. Notable advances were made by Sureshchander [5], and Rhyne et al. [6]. A significant feature of these methods is that they completely discard the cover table. However, even in these methods the power of the degree of adjacency has not been fully exploited. This paper presents a Computer-Aided Minimization Procedure (CAMP) in which the degree of adjacency guides the entire operation and all the essential and selective prime implicants to cover the function are chosen directly with minimum or no iteration.

II. THE PRINCIPLE

In this paper we use the terms minterm, prime implicant, essential prime implicant (EPI), redundant prime implicant (RPI), and minimal sum of product (MSP), to have their usual definitions [3,9]. The term selective prime implicant (SPI) and the degree of adjacency (DA) will have the definitions as given in [4,9]. However, because of its special significance in the present work we shall define and discuss the degree of adjacency.

Definition 1: Two minterms of n variables are said to *combine* with each other to produce a product term of $n-1$ variables, if the variable eliminated, and only this variable, appears in the true form in one of the minterms and in the complemented form in the other, whereas all other variables are in the same form in both the minterms. Thus

$$abcd + a\bar{b}c\bar{d} = a\bar{b}c$$

In this paper we shall express a minterm in its binary form by writing a 1 for a literal in the true form and a 0 for a literal in the complemented form. We shall also use 2 to indicate a variable which has been eliminated. Thus $abcd$, $ab\bar{c}d$, and $a\bar{b}c$ will be represented by 1011, 1010, and 1012 respectively. The decimal designation of a minterm will be the decimal number obtained by converting its binary form into a decimal form. Thus the minterm $abcd$ has the decimal designation of 15. The input to our program is the list of minterms in the decimal form, whose sum constitutes the Boolean function.

Definition 2: If a minterm in a list of minterms constituting a Boolean function can combine separately with a number α of other minterms in the list, then that minterm is said to have a *degree of adjacency (DA)* α.

Consider the function

$$f(abcd) = \Sigma(0,2,4,7,8,9,10,13)$$

which has been plotted on the Karnaugh map in Fig. 1. Here, the minterm 0 can combine with the minterms 2, 4, and 8. It, therefore, has a DA 3. Another minterm which has a DA 3 is 8. The minterms 2, 9 and 10 have DA 2 each, and the minterms 4 and 13, 1 each. The DA of minterm 7 is 0.

Every time two n-variable minterms combine, they produce a product term of $n-1$ variables. Hence it can be easily verified that a product term from which m variables have been eliminated is the result of the combination of 2^m minterms. Thus the 4-variable ($abcd$) minterms, 4(0100), 5(0101), 6(0110) and 7(0111) produce the 2-variable product term $\bar{a}b$ (0122).

Definition 3: The set of 2^m n-variable minterms whose combinations produce a product term of $n-m$ variables will be called the *subsuming set of minterms (SSM)* of the product term.

Theorem 1: The SSM of a product term expressed in the ternary form (0,1,2) can be obtained by replacing the 2s with all combinations of 0s and 1s.

Thus the SSM of the product term 02102, is obtained by replacing the two 2s by the four binary combinations as shown below:

Product term	Binary combinations	SSM	Decimal designation
02012	00	00100	(4)
	01	00101	(5)
	10	01100	(12)
	11	01101	(13)

0738-100X/84/0000/0699$1.00 © 1984 IEEE

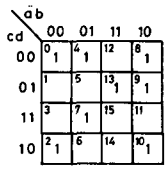


Fig. 1. A 4-variable function on map showing minterms and their degree of adjacency.

The proof of this theorem is obvious from the rules of combination of minterms. It can be easily seen that the SSM of a product term with m 2s is a cluster of 2^m minterms. The SSM is said to have an order m, since the cluster of 2^m minterms is an m-cube in the n-dimensional space.

Definition 4: A minterm M in the list L of minterms constituting a Boolean function ($M \in L$) having a DA α produces a *candidate product term (CPT)* with α 2s in place of variables, which get eliminated every time it combines with another minterm.

The minterm 0(0000) of Fig. 1 has a DA 3. The variable c gets eliminated when it combines with 2(0010), the variable b gets eliminated when it combines with 4(0100), and the variable a is eliminated on combination with 8(1000). Thus it produces the CPT, 2221.

Theorem 2*: A candidate product term (CPT) is an essential prime implicant (EPI) of the Boolean function if the subsuming set of minterms is within the function (SSM $\in L$).

Corollary 2.1: The candidate product term (CPT) of a minterm having degree of adjacency (DA), either 0 or 1, is an essential prime implicant (EPI).

Proof: The proof of the Corollary follows from the fact that when the DA is 0, the minterm is itself the CPT and its SSM, and is therefore in L. When the DA is 1, the SSM of the CPT are the two minterms which combine to produce the CPT and hence are in L

QED

Theorem 3: If the SSM generated by the CPT of a minterm is not in L, then the minterm is covered by at least two intersecting prime implicants.

Proof: If the SSM generated by the CPT of a minterm is not in L, then all the adjacent minterms are not in a single cube. Therefore, the minterms must be in at least two cubes. Each of these cubes may represent either an essential, or a redundant, or a selective prime implicant with the minterm (generating the CPT) as a common minterm. Hence the minterm must be covered by at least two intersecting prime implicants.

QED

With this background, we now describe the minimization program. In the first phase of the CAMP program all the EPIs are selected one after another. The program takes up the first minterm in the list, calculates its DA and the CPT. Both the DA and the CPT are stored in their respective arrays. If the DA turns out to be either 0 or 1, the CPT is printed out as *product term (PT)* of the solution, and the minterm(s) subsuming the PT is(are) flagged off. If the DA is 2 or more, the program computes its SSM from the corresponding CPT, and checks if the SSM is in L. If the SSM is in L, then the CPT is an EPI (Theorem 2). It is then printed out as a product term of the solution, and all the members of the SSM are flagged off. If the SSM is not in L, then the CPT is not a product term of the solution (Theorem 3), but it is still saved for possible future use.

The program then goes to the next minterm in the list which has not yet been flagged off, and repeats the procedure, until the entire list of true minterms is scanned.

It should be obvious from Theorems 2 and 3 that at the end of phase I of the program, the following situation will prevail.

(a) All EPIs have been selected and printed out as PTs of the solution.

(b) All minterms covered by the EPIs and RPIs, if any, have been flagged off.

(c) Only those minterms which are covered by SPI chains alone remain uncovered. But the program has already computed and stored their DAs and CPTs.

The procedure in phase I ensures that a minimum number of incursions is performed through the *DO* loop of the program. In fact the number of incursions in phase I of the program does not depend on the number of minterms in the function but is either equal to or a little more than the number of product terms in the final solution.

Phase II of the program chooses the selective prime implicants. This is the most expensive part of the program. Obviously SPIs are generated by the minterms which remain uncovered by the EPIs. The program takes up these minterms one by one in ascending order of their degree of adjacency, which have already been calculated in the phase I. If a minterm with a DA value of α generates an EPI, that EPI will always be an α-cube, on the other hand a minterm with a DA value of α that generates an SPI, will produce a cube of dimension between 1 and $\alpha-1$. We must try to generate the largest possible cube. Here we introduce the concept of direction which may be assigned to each degree of adjacency of a minterm. Take the Boolean function shown on the Karnaugh map of Fig. 2. The minterm 15 which is to be covered by an SPI has a DA of 3. The CPT generated by the minterm has the following form:

2 1 2 2

Let us write above each of the 2s that minterm which combines with the minterm 15 to eliminate the variable at the bit position occupied by the 2:

7 13 14
2 1 2 2

Now we shall say that the adjacency in the position of the first 2 is in the direction of 7, that of the second 2 is in the direction of 13, and that of the third 2 is in the direction of 14. Also, the largest cube that covers 15 is obtained by expanding in the directions of 7 and 13, and abandoning the direction of 14. Hence the 2s under 7 and 13 must be retained, and the 2 under 14 must be replaced by the original bit of the minterm. Hence the product term which is an SPI is given by

2 1 2 1

Thus we must find a way to determine those adjacencies which participate in forming the cube of the SPI. The following algorithm gives an indication. Generate the SSM of the CPT of the minterm, and determine those minterms which are not in the function. Write these terms in a separate table. Calculate the

* This theorem is a restatement of Theorem 3 of [4]. When the SSM of the CPT is $\in L$, the CPT gets generated in Table T_α and it becomes an EPI.

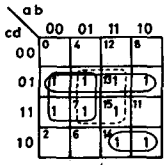


Fig. 2. The function $\Sigma(1,3,5,7,9,10,13,14,15)$

difference of the number of 1s and the number of 0s ($|N(1)-N(0)|$) for each column of this table. Let us call this difference the *column difference (CD)*. Consider the CDs of only those columns which are headed by 2s, that is, which have an adjacency direction. Find the CD which has the largest value. Add 0 to the 2 of the CPT of this column. If there are more than one column having this value of CD, add 0 to the 2s of all these columns. Then find the column(s) having the next largest CD. Add 1 to the 2(s) of the CPT of this(these) column(s). Repeat this procedure of adding graded weight (0,1,2,3,...) to the 2s of the CPT until all the adjacency columns are covered.

Let us work out this procedure for the minterm 15 of the above example.

DIRECTIONS	7		13	14		
CPT	2	1	2	2		
	0	1	0	0	4	X
	0	1	0	1	5	
	0	1	1	0	6	X
SSM	0	1	1	1	7	
	1	1	0	0	12	X
	1	1	0	1	13	
	1	1	1	0	14	
	1	1	1	1	15	

	7		13	14	
	2	1	2	2	
	0	1	0	0	4
	0	1	1	0	6
	1	1	0	0	12
$\|N(1)-N(0)\|$	1		1	3	

Direction 14 has the largest CD. So the 2 under 14 is made 2+0 or 2. The next value of CD is 1 (directions 7 and 13). These 2s are then made 2+1 or 3. The CPT now appears as

3 1 3 2

The graded weights in the CPT now show an ordering (starting from the least weight 2) which indicates the ordering of abandoning the adjacency directions. In this example the 2 is in the direction of 14 which must be abandoned first. The program will replace the 2 under 14 by the original bit of the minterm (in this case 1), and will check if the SSM generated by 3131 is in L. If that is so (it is so in this example), 3131 will be the largest cube which covers the minterm, and therefore, will be the selected SPI. The cube 3131 is then a PT of the solution. Before, printing out the PT, all weights higher than 2 are replaced by 2s, so that the PT of the solution is printed out in the ternary form:

2 1 2 1

In this example, all the three directions of 15, namely 7, 13, and 14 are covered. There may be other cases where one or more directions remain uncovered. Then these must be made the most deferred directions in the ordering of abandoning the adjacency directions. The CPT of a minterm which generates an SPI, therefore, goes through a two-step preprocessing. In the first step, each of the adjacency directions is checked to find out if it has already been covered. If it is not covered, and also if the 'direction-minterm' is not a don't-care minterm, then the corresponding 2 in the CPT is made α+2 where α is the DA of the minterm. In the second step of preprocessing, the graded weights of the adjacency columns are determined, and the corresponding 2s or (α+2) are augmented accordingly.

Thus, a CPT

1 2 0 2 2 1 2 0 2

after the first preprocessing step, may become

1 2 0 7 7 1 2 0 2

and after the second preprocessing step, may appear as

1 3 0 8 7 1 2 0 2

Now, the program inserts the original bit of the minterm in the place of the first 2, and checks if the new CPT becomes a PT. If not, it repeats the process with the second 2 and so on. After all 2s have been replaced by original bits of the minterm, if no PT is found, then the process is repeated with 3s, 4s, 5s, and so on until a PT is found. It is obvious that due to this different weightage which is being assigned to each direction of adjacency, the program selects the best cube of the SPI chain to cover the minterm.

Sometimes, the SSM of a CPT may have only one term which is missing in the function. In such a situation, no ordering of adjacency columns can be determined as all CDs will have the same value. However, it is obvious that in this case the minterm of the CPT can be definitely covered by an SPI which is an (α−1) cube. The program determines this PT by at most α iterations.

The program takes up the minterms of one particular DA value at a time, starting from the DA value of 2 to the maximum value DAMAX. Even when taking up the minterms of one particular value of DA, it takes up a minterm if at least one of its adjacency direction has already been covered by an EPI or an SPI which has been already selected. If the given function is a cyclic function, then it does not have any EPI, and in the first pass of phase II of the program no minterm is taken up. The program then decides to branch, and the first minterm in second pass is chosen, and the SPI to cover it is selected. This SPI covers one or more other minterms. Thus, in the second pass several SPIs are selected. Every time the CPT of the SPI goes through the two-step preprocessing and the appropriate weightage is assigned to each direction of adjacency. As a result, no superfluous product term is generated even if the function is cyclic.

Don't care terms are easily accounted for by including them in the function array, so that they may participate first in the calculation of DA, and then in the formation of the SSM. Don't care terms are not included in the array of true minterms, and hence they do not initiate any action in any phase of the program.

III. THE PROGRAM

The CAMP program was written in FORTRAN. The Boolean function to be minimized is expressed as a sum of minterms. The minterms are read and stored in an array named *AT*, and the don't care minterms are read and stored in an array named

AX. The number *NV* of variables in the function, the number *NT* of true minterms, and the number *NX* of don't care minterms are also supplied as data. The program prepares a third array, *AFUN*, which has both the true and don't care minterms in it. The complete program consists of one function subprogram, three subroutines, and a main program. The entire program has about 240 lines. The main program has two phases. Phase I selects all EPIs. It also calculates the degree of adjacencies of minterms which are not covered by the EPIs. The CPTs and the directions of the adjacencies of all uncovered minterms are kept stored, and are used in the second phase of the program. Phase II of the program selects the SPIs. The computer prints out the values of *NV, NT, NX,* and the list of true and don't care minterms which are supplied as data. The result of minimization is printed out as minterm, DA and the product terms. In the product term a 0 represents a complemented variable, and a 1, a true variable. A 2 represents a variable which has been eliminated. The program also prints out a series of 1s and 0s to indicate the status of minterms after each phase. A 1 means an uncovered minterm, and a 0 means a minterm which has been covered. Fig. 3 shows a typical printout, along with the function plotted and solved on the map.

IV. CONCLUSIONS

The minimization program as described in this paper can handle functions of large number of variables and requires very little CPU time. If the function consists mainly of essential prime implicants and a small number of selective prime implicants, then the result is precise and accurate and we get the minimal sum-of-product (MSP) form as would be obtained in any classical method. This is because the procedure is based on exact theorems of switching theory. If the function is very dense, that is, consists of a relatively large number of minterms, then there may be many interconnected cyclic chains of SPIs. In such a case, the solution may still be near minimal as a few extra product terms might be generated. However, in such a case, it is advisable to minimize the complementary function. If the original function is very dense, then the complementary function is very sparse and will have smaller number of product terms than the original function. So, it will be more economical to realize the complementary function and use an inverter to get back the original function.

The program is also quite fast. We solved a 31-variable function with 44 minterms. The solution obtained had 17 EPIs in phase I, and 2 SPIs in phase II. The program running on DEC 1090 computer produced this result in 2.23 CPU seconds. We also ran a 9-variable dense function having 420 minterms. The solution, which consisted of 130 cubes (all SPIs), was obtained in 26.30 seconds of CPU time. The authors of MINI [10] have reported obtaining a solution consisting of 85 cubes in about 20 minutes of IBM 360/75 CPU time. So, it is very likely that CAMP will be faster than MINI. We also solved the complementary function of the dense 9-variable function, and obtained a solution consisting of 72 cubes (all EPIs) in 1.40 seconds of DEC 1090 CPU time.

CAMP minimizes a single function. In general, a PLA realizes multiple output functions. Hence after the product terms of every individual output have been obtained, the entire set of product terms is reduced further [8,11].

ACKNOWLEDGMENT — The author is thankful to Dr. Prathima Agrawal of the AT&T Bell Laboratories, Murray Hill, N.J., for many valuable discussions about the procedure in this paper.

REFERENCES

[1] W. V. Quine, "The problem of Simplifying Truth functions," *Am. Math. Monthly,* Vol. 59, pp. 521-531, Oct. 1952.

[2] E. J. McCluskey, "Minimization of Boolean Functions," *Bell System Tech. Journal,* Vol. 35, pp. 1417-1444, Nov. 1956.

[3] E. J. McCluskey, *Introduction to the Theory of Switching Circuits,* New York: McGraw-Hill, 1965.

[4] N. N. Biswas, "Minimization of Boolean Functions," *IEEE Trans. on Computers,* Vol. C-20, pp. 925-929, August 1971.

[5] Sureshchander, "Minimization of Switching Functions — A Fast Technique," *IEEE Trans. on Computers,* Vol. C-24, pp. 753-756, July 1975.

[6] V. T. Rhyne, P. S. Noe, M. H. McKinney, and U. W. Pooch, "A New Technique for the Fast Minimization of Switching Functions," *IEEE Trans. on Computers,* Vol. C-26, pp. 757-764, August 1977.

[7] H. A. Curtis, "Adjacency Table Method of Deriving Minimal Sums," *IEEE Trans. on Computers,* Vol. C-26, pp. 1136-1141, November 1977.

[8] Z. Arevalo and J. G. Bredeson, "A Method to Simplify a Boolean Function into a Near Minimal Sum-of-Products for Programmable Logic Arrays," *IEEE Trans. on Computers,* Vol. C-27, pp. 1028-1039, November 1978.

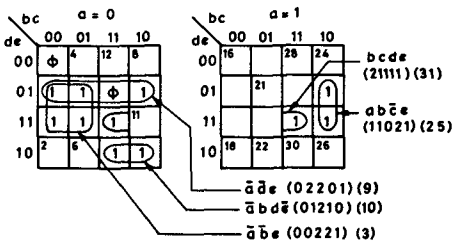
[9] N. N. Biswas, *Introduction to Logic and Switching Theory,* New York: Gordon and Breach, 1975.

[10] S. J. Hong, R. G. Cain and D. L. Ostapko, "MINI: A Heuristic Approach for Logic Minimization," *IBM Journal of Research and Development,* Vol. 18, pp. 443-458, Sept. 1974.

[11] J. F. Martinez-Carballido and V. M. Powers, "PRONTO: Quick PLA Product Reduction," *Proceedings of the 20th Design Automation Conference,* Miami Beach, Florida, June 1983, pp. 545-550.

```
NV   NT   NX
 5   11    2
TRUE MINTERMS
 1   3   5   7   9   10   14   15   25   27   31
DONT CARE MINTERMS
 0  13
MINTERM     DA     PROD TERMS      STATUS OF MINTERMS
   3         2      0 0 2 2 1
  10         1      0 1 2 1 0
                                   0 0 0 0 1 0 0 1 1 1 1
  25         2      1 1 0 2 1
  31         2      2 1 1 1 1
   9         3      0 2 2 0 1
                                   0 0 0 0 0 0 0 0 0 0 0
```

(a) CAMP printout.



(b) Map solution.

Fig. 3. CAMP result for a five-variable function