Cubical CAMP for Minimization of Boolean Functions

Nripendra N. Biswas nnb@ece.iisc.ernet.in
C. Srikanth* csri@miel.mot.com
James Jacob james@ece.iisc.ernet.in
Department of Electrical Communication Engineering
Indian Institute of Science
Bangalore 560 012 India

Abstract

The paper presents QCAMP, a cube-based algorithm for minimization of single Boolean functions. The algorithm does not generate all the prime cubes, nor does it require the Off-set of the function. Two significant contributions of QCAMP are the UNATE_TEST which tests if a given function is a unate function, and the BISECT procedure which minimizes a cyclic function without taking recourse to branching. A well known property of a unate function is that the prime cubes subsuming a unate function are all essential prime cubes. Hence as soon as a function passes the UNATE_TEST, all its prime cubes are recognized as solution cubes without any further processing. Many special functions, such as both the On and Off-sets of Achilles' heel functions which ESPRESSO II finds hard to minimize are also unate functions. Consequently, as will be evident from the computational results QCAMP exhibits far better performance compared to ESPRESSO II in all such and many other functions.

1 Introduction

The importance of single function minimization algorithms is well known, since it forms the most basic and important ingredient of multiple output minimization, a valuable tool in the design of VLSI circuits. However, VLSI technology demands that an algorithm should be capable of handling a very large number of input variables. This shift of emphasis from small or medium to very large number of variables has made the minterm based algorithms virtually obsolete in a VLSI environment. To respond to this new and challenging situation many cube based algorithms have been developed[1,2,3,4]. Most of these evolve around two schools of philosophy. The Quine McClusky[5] school generates all prime cubes, which proves to be very expensive in some situations. ESPRESSO[4] school generates the Off-set of the function. This also renders the algorithm very inefficient in many cases. For example, although the On-set of a 30-variable Achilles' heel function has only 10 cubes, its Off-set has 59049 cubes. Hence even to minimize only these 10 cubes of the On-set, ESPRESSO II has to spend considerable time in simply generating the 59049 cubes of the Off-set. The third school of philosophy, namely that of the CAMP algorithm[6,7,8,9] avoids both these undesirable features. In this paper we describe QCAMP, a cubical minimization procedure based on the CAMP philosophy.

*During the course of this work C. Srikanth was a post graduate student in the Department of Electrical Communication Engg. of the Indian Institute of Science, Bangalore. He is now with Motorola India Electronics Limited (MIEL), "The Senate", 33-A, Ulsoor Road, Bangalore-560 042 India.

2 Ternary Notations and Cubical Operations

Before we present the algorithm, the ternary notations and cubical operations extensively used in the paper may be explained. We assume that some of the basic terms of logic design and switching theory, such as minterm, product term, sum of product(SOP) form, minimum sum of product(MSOP) form, conjunctive and disjunctive canonical form (CCF and DCF) and others are already familiar to the reader. In this paper a product term will be represented in the ternary notation of 0, 1 and 2 where 0 represents a variable in the complemented form, 1 in the true form, and 2 indicates the absence of the variable. For example, the 4 variable function, $f_1 = a'b + bc'd + ab'cd' + acd$ will be written as

$$f_1 = 0122 + 2101 + 1010 + 1211$$

Each product term is called a cube. The dimension of a cube gives the number of variables absent in the product term which the cube represents. A minterm is a cube of dimension 0 and is therefore called a 0-cube. The cube 0122 representing $a'b$ has a dimension 2, and is therefore a 2-cube. Note that the dimension of a cube is given by the number of 2's. Also note that a cube of dimension α of an n-variable function is obtained by combining 2^α minterms of the n-variable function. Again an α-cube of an n-variable function ($0 \leq \alpha \leq$ n) is a product term of (n $-$ α) number of variables.

Definition 2.1 : The *distance* between two cubes is the number of bit positions where one cube has a 0(1) and the other a 1(0).

Thus the distance between 0122 and 2101 of the function f_1 given above is 0, whereas that between 2101 and 1010 is 3.

Definition 2.2 : Two cubes are said to be *intersecting* with each other if the distance between them is 0.

Definition 2.3 : Two cubes are said to be *adjacent* with each other if the distance between them is 1.

Thus the cube 1211 is adjacent to all the three other cubes of the function f_1. To visualize the intersection and adjacency of cubes pictorially, plot the four variable function on a Karnaugh map. A detailed discussion of these and the cubical operation sharp can be found in [9,10].

In this paper, we shall also define the prime cubes in three categories as has been done in [9].

Definition 2.4 : If among the minterms subsuming a prime cube, there is at least one that is covered by this and only this prime cube, then the prime cube is called an *essential prime cube* (EPC).

The subcube of an EPC that is covered by this and only this prime cube is called a *Distinguished cube*. The minterms subsuming a distinguished cube are called distinguished minterms.

Definition 2.5 : If each of the minterms subsuming a prime cube is covered by other essential prime cubes, then the prime cube is called a *redundant prime cube* (RPC).

Definition 2.6 : A prime cube which is neither essential nor redundant is called a *selective prime cube* (SPC).

Among the minterms subsuming such a prime cube there is at least one that is covered neither by an EPC nor by this and only this prime cube. Therefore the existence of one SPC implies the existence of another. We now proceed to present the QCAMP algorithm. The algorithm has three phases. (1) The essential prime cube or EPC phase; (2) The selective prime cube or SPC phase and (3) The don't care or DC phase.

3 The EPC Phase

In this phase all the EPCs of the given function are selected and stored in the solution cube matrix, SCM. The given function is input in two matrices. In the first, denoted as the the true cube matrix TCM, all the true cubes are written in the ternary notation. In the other, denoted as the don't care matrix DCM, only the don't care cubes are written in the ternary notation. The variables are written in decimal numbers 1,2, and so on in the 0th row of the matrix, whereas the 0th column has the serial number of the cubes. In the EPC phase, only the TCM is considered. Before processing the TCM, it is checked whether the switching function is unate.

Definition 3.1 : If, in the minimum-sum-of-products form of a switching function, each variable appears in its true form or its complemented form, but not both then the function is called a *unate function*[9].

3.1 The UNATE_TEST

First the TCM is subjected to the UNATE_TEST. If the function passes the test then it is a unate function and all the input cubes are put in the solution cube matrix SCM, without any further processing. This is based on the following theorem[9].

Theorem 3.1.1 : A Switching function is unate if and only if it can be expressed as a sum of prime cubes, all intersecting at a common subcube. Furthermore, all the prime cubes are essential prime cubes.

The proof of the theorem can be found in Sec 3.10 of [9].

The UNATE_TEST is very simple and inexpensive. The TCM is scanned columnwise, starting from the first column. If there are only 1s or 0s (ignoring 2s), but not both, in each of the columns then the function is unate. This test effectively checks whether each variable of the function is appearing only in its true or complemented form. Let us consider the 6-variable function

$$f_2 = 220220 + 202220 + 022220 + 220202 + 202202 + 022202 + 220022 + 202022 + 022022$$

The TCM for this switching function is as shown in Table 1. The 0-th row of Table 1 gives the variables in decimal numbers, and the 0-th column gives the serial number of cubes also in decimal number. If the UNATE_TEST is applied to this TCM, it will pass the test and all the 9 cubes will be stored in the SCM as EPCs.

If a function fails to pass the UNATE_TEST then the input cubes are made to grow into adjacent cubes as much as possible, and the derived matrix is called the Prime Cube Matrix (PCM). Now the UNATE_TEST is applied again to the PCM. This is because the input function may not be initially expressed as a sum of prime cubes. After expansion all the cubes are prime cubes. If the function is unate it will pass the test at this stage, and all the prime cubes are essential prime cubes and will be stored in the SCM. Let us now consider a non-unate switching

Table 1: A 6 variable unate function

0	1	2	3	4	5	6
1	2	2	0	2	2	0
2	2	0	2	2	2	0
3	0	2	2	2	2	0
4	2	2	0	2	0	2
5	2	0	2	2	0	2
6	0	2	2	2	0	2
7	2	2	0	0	2	2
8	2	0	2	0	2	2
9	0	2	2	0	2	2

function having 5 cubes

$$f_3 = 0002 + 1020 + 2201 + 1112 + 0211$$

This will obviously fail the UNATE_TEST. Now the cubes are made to grow into the adjacent cubes. The new cube matrix having all the cubes in the fully expanded form consists only of prime cubes, and therefore becomes the prime cube matrix PCM. The PCM of f_3 is shown in Fig. 1(b). Note the important fact that two prime cubes 2121 and 1210 which will be generated in the QM (Quine McCluskey) algorithm have not been generated here. Nor have we generated the Off-set of the function. Now the UNATE_TEST is applied to the PCM. It fails the test again indicating that

it is not a unate function. Now, each cube is subjected to the EPC_TEST. If the result of the EPC_TEST is positive, then the cube is an EPC, and is stored in the solution cube matrix SCM.

3.2 The EPC_TEST

Each fully expanded cube of the PCM (Fig. 1(b)) is now subjected to the essential prime cube test, EPC_TEST. For this a matrix called adjacent and intersecting cube matrix AIM, having all cubes which are adjacent to or intersecting with the cube under test is formed. From this another matrix called test matrix TM is derived. TM is obtained from AIM by deleting all the columns headed by 0s and 1s of the cube under test. The TM therefore, has α number of columns and m rows, where α is the dimension of the cube under test C_t, and m is the number of rows of its AIM. The TM is therefore a cube matrix having m cubes of α variables. It can be shown that these cubes are those subcubes of C_t which can be covered by other cubes of the function.

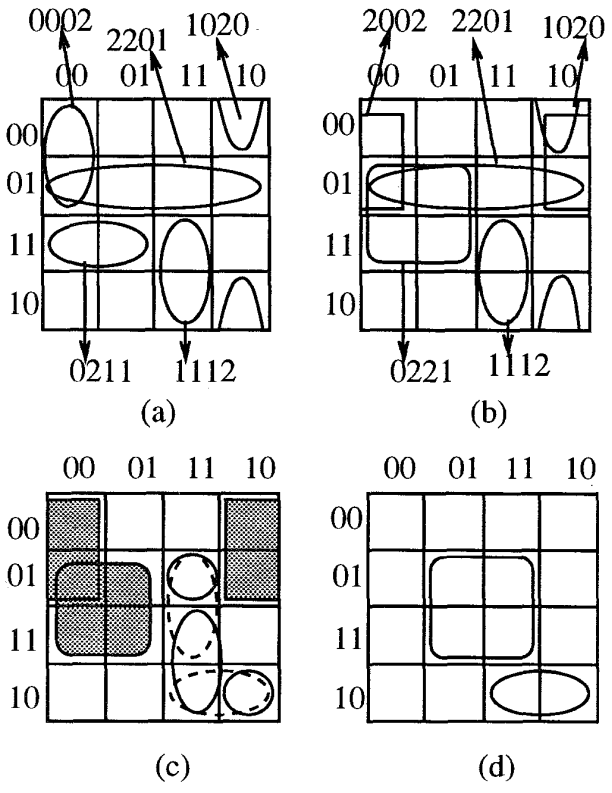

Figure 1: (a) Input CM of f_3 (b) PCM (c) EPCs and Sharped SPCs (d) Selected and expanded SPCs

Hence if the cubes of TM can combine and expand to form the all-2 cube, then all the minterms of C_t can be covered by other prime cubes and C_t cannot be an EPC. On the other hand if the TM does not expand to the all-2 cube, then there is at least one subcube (which may be a 0-cube or a minterm) which is covered by C_t and only C_t. Hence C_t is an EPC. Table 2 gives the EPC_TEST tables for the cubes 0221 and 2201. We see that the cube 0221 passes the EPC_TEST while 2201 fails the test. Among the other cubes of the PCM, cube 2002 will pass the EPC_TEST, whereas cubes 1020 and 1112 will fail the EPC_TEST. Thus after the EPC_TEST, all EPCs will be identified, and the rest of the cubes will be either RPCs or SPCs. The EPCs are stored in the solution cube matrix SCM. The EPC phase ends here and the SPC phase begins.

Table 2: THE EPC_TEST TABLES

Cube Under Test	0221	2201
AIM	2002 2201 1112	2002 1020 1112 0221
TM	00 20 11	20 10 11 02
TM expands to all-2 cube	No	Yes
EPC ?	Yes	No

4 The SPC Phase

4.1 Iterative Mode : EPC_SPC Procedure

If at the end of the EPC phase, all cubes turn out to be EPCs then the algorithm terminates. If on the other hand some or all cubes remain as RPCs or SPCs, then SPC phase begins. Now the EPCs stored in the SCM sharp the remaining cubes of the PCM. The sharped cubes for our example are shown as the three unshaded cubes of Fig. 1(c). Note that after this first sharp operation, all RPCs, if any, get eliminated. These sharped cubes (unshaded in Fig. 1(c)) are now treated as a new CM and also processed as a new CM. After the EPC phase of the new CM, in our example two EPCs (shown dotted in Fig. 1(c)) 1121 and 1210 are formed. These are then made to expand into the entire function. As a result, the cube 1121 expands to cube 2121. The cube 1210 does not expand. No cube of the new CM is left out after the new EPCs are selected. Hence, the algorithm terminates, giving two EPCs and two SPCs as solution cubes in the SCM. Thus, the function f_3 after minimization becomes,

$$f_3 = 0221 + 2002 + 2121 + 1210$$

If at any time of the iterative mode of the SPC phase the new CM does not generate an EPC, then the cubes of the new CM are only SPCs, and the function is a cyclic function. Then the cyclic mode of the SPC phase begins.

Table 3: BISECT ALGORITHM

(a)	(b)	(c)	(d)	(e)
0122 C_c	2021 C_c	2210 C_c	1202 C_c	0122 C_c
1202	1202	1202 C_a2	0122 C_a3	2021 C_a4
1022	1022 C_bx	0122	2021	2210
2210	2210 C_a1	2021	2210	1202
0221 C_b1	0122			

4.2 The Cyclic Mode : BISECT Procedure

Whenever any CM, either the input CM or any intermediate CM in the iterative mode does not generate even a single EPC, then this CM is that of a cyclic function. QCAMP then goes into the cyclic mode, and the BISECT procedure is invoked.

Let us explain the procedure by working out the solution of the following cyclic function plotted in Figure 2(a).

$$f_4 = 0122 + 1202 + 1022 + 2210 + 0221$$

The function f_4 has 5 prime cubes with no redundant cube. Let the processing start from cube 1. Part (a) of Table 3 is the input CM. It goes through several modifications until it comes to the minimum solution form as given in part (e) of Table 3. The steps of BISECT procedure are as follows.

First set up two count variables, *acount* (adjacent count) and *ecount* (bisect and expand count). Initialize each of them to 0.Take the first cube of the CM as the cube under consideration. Call it C_c. The algorithm for the BISECT procedure is as follows:

1. Search in other cubes in serial order for a cube C_b which gets bisected by C_c.
2. If C_c gets a C_b check if the half of C_b which is outside C_c is included wholly in one or more cubes of the function. If yes, then delete C_b, and continue search for another C_b for C_c.
3. If the half of C_b which is outside C_c is not included wholly, then expand this half of C_b in the function. Call the expanded cube C_e. Delete C_b (denoted by x in Table 3).
4. Move the cubes below C_b up. Place C_c as the last cube of the CM, and C_e as the first cube of the CM.

This completes one cycle of operations. Now, in the modified CM, C_e which is the first cube becomes the C_c, and the procedure repeats. Sometimes the outside half of a C_b may not be able to expand in the function. In this case cube C_c abandons this C_b (and C_b is kept undisturbed in the CM) and searches for another C_b. If at any time C_c does not get a C_b, it starts searching for a cube C_a that is adjacent to C_c. A C_c that does not get a C_b and then a C_e will surely get a C_a. When it gets a C_a, the *acount* is incremented and *ecount* is reset to 0. Similarly when a C_c gets a C_e *ecount* is incremented and *acount* is reset to 0 (the current value of the counts are shown after C_a and C_b in Table 3). Whenever any one of the two counts equals the number of cubes in the CM, the procedure terminates, and the cubes of the CM becomes the solution cubes. When a C_c gets a C_a the C_a becomes the first cube of the CM, and therefore, C_c of the next cycle. Cubes below C_a are moved up, and cube C_c is placed as the last cube of the CM. The cycle repeats itself, until the final SCM is obtained. To summarize, in each cycle the C_c first tries to get a C_b and then a C_e. On failure, it gets a C_a. In each cycle either the *acount* is incremented and *ecount* is reset to 0, or vice-versa. When one of the counts equals the number of cubes of CM, the procedure terminates and the CM becomes the SCM. Both at the beginning and end of BISECT, redundant cubes if any, are removed. Table 3 (a) to (e) show the 5 cycles of the BISECT algorithm to produce the 4 cube solution of the function f_4,

$$f_4 = 0122 + 2021 + 2210 + 1202$$

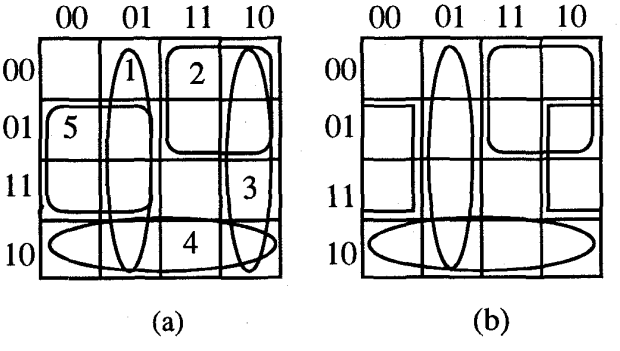

Figure 2: (a) Input CM of f_4 having 5 cubes (b) Bisect solution of f_4 having 4 cubes

5 The DC Phase

Before coming to the DC phase, the true cubes of the function are fully minimized, and the solution cubes are stored in the solution cube matrix SCM. Now, the cubes of the SCM are made to expand and combine further in the cubes of the don't care matrix DCM. Some of the cubes of the SCM may not expand. A new matrix is now formed having the expanded true

cubes, and is called XCM. Another matrix called included cube matrix ICM, having those true cubes of SCM which have expanded, is also formed. Consider the function,

$$f_5 = \sum(1,2,5,15) + \phi \sum(0,3,6,7,9,12,13,14)$$

The function as plotted on the Karnaugh map is also shown in Fig. 3(a). If an included cube IC is covered by an expanded cube XC only partly and not fully then the IC is split into subcubes, so that each subcube is now covered fully, by an XC. Each subcube of a split IC is treated as a separate IC and is also stored as such in the ICM.

In XCM the XCs are arranged in ascending order of their dimensions. Another matrix called cover matrix CVM is now derived from the XCM and the ICM. Each row of the CVM represents an XC of the XCM and each column of the CVM represents an IC. CVM also has an extra row and an extra column. The extra row has the weights of each column, whereas the extra column has the weights of each row, where weight is defined as follows:

Definition 5.1 The number of 1's of a row (column) will be called its *weight.*

All the solution cubes of the function are found by processing the CVM. While processing we will have to delete some of the XCs and ICs. The weights of each row and column of the CVM are now computed and placed in the extra column and row of the CVM. Now, first search for a column having weight 1. Such a column will be called a *one weight column* OWC. If an OWC exists, then select the XC of the row having the singleton 1 of the OWC as a solution cube. Delete this row and also the ICs covered by this XC.

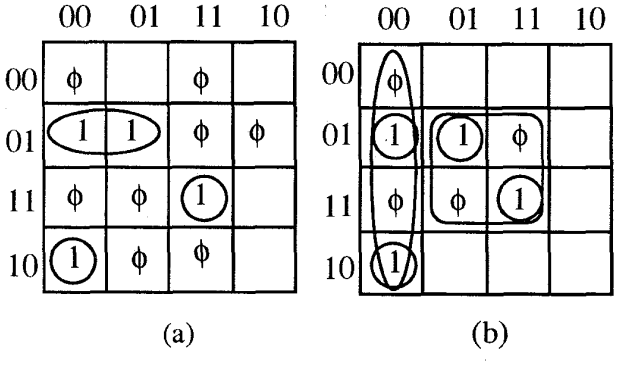

Figure 3: (a) Input CM of f_5 with the don't cares. (b) Final Solution of function f_5.

Now, delete the selected row and the ICs covered by the XC. If no OWC is found, then find the row having the least weight. Such a row will be called a *least weight row* LWR. Search for a LWR starting from the first row of the table. Delete the first LWR found. Then compute the weights of the rows and columns once again. This gives a new table. Search for OWC, and repeat this procedure until all the rows of the table are deleted. For the function f_5, the CVM goes through 7 iterations, and yields two solution cubes. Thus f_5 minimizes to

$$f_5 = 2121 + 0022$$

also shown in Figure 3(b).

6 Special Functions

There are some special functions which ESPRESSO II finds "unreasonable" and therefore hard to minimize. Achilles' heel function is a good example of such functions[4]. It is defined as

$$f = x_1x_2x_3 + x_4x_5x_6 + \cdots + x_{3k-2}x_{3k-1}x_{3k}$$

where k is any positive integer

This function is unate, has 3k variables, and its minimum cover has k cubes. However, its complement f' has a minimum cover of 3^k cubes.

The 6-variable unate function shown in Table 1 is in fact the Off-set of a 6-variable (k = 2) Achilles' heel function. The last property of Achilles' heel functions (the Off-set having a minimum cover of 3^k cubes) poses problem to ESPRESSO II since it has to generate the complement of the function. QCAMP does not have this problem. However, a very helpful characteristics of the function, which apparently has not been utilized by ESPRESSO II is its unateness. Since, the function, and therefore also its complement are unate, once it is expressed as a sum of prime cubes, all the prime cubes will be EPCs due to Theorem 3.1.1 The computational results shown in Table 4 shows beyond doubt the effectiveness of UNATE_TEST. The fact that QCAMP is faster than ESPRESSO II for the Achilles' heel functions shows that UNATE_TEST has not been incorporated in the latter. It will be a good thing if UNATE_TEST is incorporated in all the minimization algorithms.

Another class of functions which we have called "ODS" (one diagonal short) functions has all minterms of the function except a pair of opposite minterms. (Such a pair of opposite minterms constitute a diagonal in an n-dimensional hypercube). An ODS function of n variables is a cyclic function having more than one solution of n cubes. The BISECT algorithm of QCAMP handles the function as efficiently as ESPRESSO II. Figure 2(a) and (b) show a 4-variable ODS function.

7 Computational Results

The Table 4 compares the CPU time taken by ESPRESSO II and QCAMP by running various functions on both the algorithms.

8 Conclusion

Apart from presenting a very efficient and economical algorithm for minimization of Boolean functions, the algorithm presents for the first time the UNATE_TEST and a new way of handling cyclic functions in the BISECT procedure which does not require branching , inasmuch as the processing may start from any cube to obtain most of the time one

Table 4: Computation Results

Function Name	Input File $(i,n)^\dagger$	No. of cubes in Soln.		CPU Sec on IBM RS 6000/580	
		ESPRESSO	QCAMP	ESPRESSO	QCAMP
Achilles' heel (On-set)	21,7	7	7	.2	< .1
	24,8	8	8	.8	< .1
	30,10	10	10	8.7	< .1
	33,11	11	11	31.7	< .1
	36,12	12	12	103.5	< .1
Achilles' heel (Off-set)	12,81	81	81	.2	.1
	15,243	243	243	2.2	.6
	18,729	729	729	22.0	6.9
	21,2187	2187	2187	200.0	65.0
XOR	8,128	128	128	.1	< .1
	9,256	256	256	.6	.2
	10,512	512	512	2.2	.5
	11,1024	1024	1024	8.4	2.0
	12,2048	2048	2048	32.7	8.1
ODS	7,12	7	7	.2	.1
	9,16	9	9	1.1	.7
	11,18	11	11	11.6	10.6
	12,22	12	12	47.4	45.5

† : i = number of input variables
n = number of input cubes

of the minimum valid solutions. The UNATE_TEST is very inexpensive but is extremely powerful in handling even the Off-set of Achilles' heel functions which ESPRESSO II finds hard to minimize. We recommend that UNATE_TEST be incorporated in ESPRESSO II, and for that matter, in all Boolean function minimization algorithms.

Acknowledgment– The authors wish to record their appreciation for the faculty and staff of the Supercomputer Education and Research Centre, ERNET and PROTOCOL laboratories of the ECE Department, of the Indian Institute of Science, Bangalore, for extending all facilities and cooperation during the progress of this work.

References

[1] S.J. Hong, R.G. Cain, D.L. Ostapko. MINI: A Hueristic approach for Logic Minimization.*IBM J. Res. Dev.,* Vol. 18,September 1974, pages 443-458.

[2] B. Gurunath, and N.N. Biswas. An Algorithm for Multiple Output Minimization. *IEEE Trans. Comput.-Aided Design,* Vol. CAD-8, No. 9, September 1989, pages 1007-1013.

[3] M.R. Dagenais, V.K. Agarwal, and N.C. Rumin. McBOOLE: A New Procedure for Exact Logic Minimization.*IEEE Trans. Comput.-Aided Design,* Vol. CAD-5, No. 1, January 1986, pages 229-238.

[4] R.K. Brayton, G.D. Hachtel, C.T. McMullen, and A.L. Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis,* Kluwer Academic Publishers, Boston, 1984.

[5] E.J. McCluskey. Minimization of Boolean functions.*Bell Syst. Tech. Journal,* Vol. 35, No. 11, November 1956, pages 1417-1444.

[6] N.N. Biswas. Computer Aided Minimization Procedure for Boolean functions.*Proc. 21st Design Automation Conference*, Albuquerque, N.Mex., June 1984, pages 699-702.

[7] N.N. Biswas. Computer Aided Minimization Procedure for Boolean functions.*IEEE Trans, Comp. Aided Design of Integrated Circuits and Systems,* Vol. CAD-5, No. 2, April 1986, pages 303-304.

[8] N.N. Biswas. On Covering Distant Minterms by the CAMP Algorithm.*IEEE Trans, Comp. Aided Design of Integrated Circuits and Systems,* Vol. CAD-9, No. 7, July 1990, pages 786-789.

[9] N.N. Biswas. *Logic Design Theory.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[10] D.L. Dietmeyer. *Logic Design of Digital Systems.* 2nd edition, Allyn and Bacon, Boston, MA, 1978.