An Algorithm for Multiple Output Minimization

B. GURUNATH, MEMBER, IEEE, AND NRIPENDRA N. BISWAS, SENIOR MEMBER, IEEE

Absfracf—A **computer-aided design procedure for the minimization of multiple output Boolean functions as encountered in the synthesis of VLSI logic circuits is presented in this paper. A fast technique for the determination of essential prime cubes without generating all the prime cubes is among the salient features of the algorithm. The paper also describes a new class of selective prime cubes called valid selective prime cubes. This new class of prime cubes has proved to be a very powerful tool inasmuch as it guides the algorithm to the minimal set of selective prime cubes while encountering either an independent chain or an interconnected chain of cyclic prime cubes. In many cases, this avoids branching which is computationally an expensive operation. The algorithm does not generate either the complement or all the prime cubes of the functions. Therefore, it is well suited to minimize functions with large complement size and/or very high number of prime cubes. The algorithm has been implemented in Pascal and evaluated using a large number of programmable logic arrays (PLA's) including those of the Berkeley PLA test set. Results of comparison with ESPRESSO II [1] and McBOOLE [2] indicate that the program produces absolute minimal solution in most of the cases and near minimal in a few others.**

I. INTRODUCTION

MINIMIZATION of multiple output Boolean functions has assumed special significance due to the extensive use of programmable logic arrays (PLA's) in VLSI circuits. The minimization algorithms for obtaining a minimal sum of products expression suitable for a PLA implementation can be classified broadly into two categories: minterm based and cube based.

Among the minterm-based algorithms is the well-known Quine [3] and McCluskey [4] method of logic minimization. Bartee [5] extended the Quine–McCluskey method to multiple output minimization. Biswas [6] presented the adjacency method for single output minimization where essential prime implicants are selected during the process of forming the combination table. Rhyne *et al.* [7] described a directed search algorithm for the minimization of single output functions. The potential of degree of adjacency introduced in [6] has been exploited by Biswas to obtain a two-pass procedure CAMP [9] for single output minimization. Multiple output minimization (MOM) by Agrawal *et al.* [8] depends heavily on the philosophy and procedures of CAMP and uses the frequency of the minterms as a main parameter for computing the prime implicants. Biswas and Gurunath have shown in BANGALORE [10] that any algorithm which is mainly guided by the frequency of occurrence of the minterms is likely to generate more product terms in many cases. Also, as pointed out in [9], for functions consisting of mostly selective prime implicants, the minimality obtained by MOM may be poor. The introduction of the concepts of unique cover and valid cover in BANGALORE has resulted in optimal solutions in most of the cases. The main drawbacks with these minterm-based algorithms are the exponential growth of storage and time requirements with the number of variables and the number of minterms. **As** the functions encountered in a typical VLSI circuit may have 15–40 input variables, the minterm-based algorithms are found to be inadequate in handling such large functions.

Many algorithms have been proposed to directly minimize the cubes without converting them into minterms. MINI proposed by Hong *et al.* [11] was conceptually new and represented a significant departure from the classical approaches. The method computes the final solution by an iterative improvement of an initial solution. Brayton *et al.* [1] reported ESPRESSO II, another heuristic minimization algorithm based on the philosophy of MINI. The McBOOLE logic minimizer reported by Dagenais ***et*** *al.* [2] is based on the philosophy of the Quine–McCluskey method which guarantees the exact minimal solution. Sasao [12] has proposed a hardware scheme for logic minimization.

Most of the earlier minimization algorithms were based on the Quine–McCluskey philosophy requiring the generation of all prime implicants and then selection of a set of prime implicants that constitutes the optimal solution. Although this two-step processing yields an exact minimal solution, the computational complexity of the algorithms that depend on this philosophy grows exponentially with the number of variables. Miller [13] has shown that the number of prime implicants of an n-variable function may be as high as $3^n/n$. As shown by Breitbart and Vairavan [14] the generation of all prime implicants can be wasteful for an already complex problem of logic minimization. They have also shown that there are functions for which the useful prime implicants form an extremely small fraction of all the prime implicants. In fact, the authors of McBOOLE [2] have mentioned that the very large number of prime implicants was the limitation in most of the examples that McBOOLE could not handle. On the

Manuscript received November 2, 1987; revised September 27. 1988 and March 16, 1989. The review of this paper was arranged by Associate Editor M. R. Lightner.

B. Gurunath was with the Department of Electrical Communication Engineering. Indian Institute of Science, Bangalore. India. He is now at 78 South End Road, Basavanagudi. Bangalore 560 004, India.

N. N. Biswas was with the Centre for Advanced Computer Studies. University of Southwestern Louisiana, Lafayette. LA. He is now with the Department of Electrical communication Engineering. Indian Institute of Science, Bangalore 560 012, India.

IEEE Log Number 8929549.

other hand, most of the heuristic algorithms generate the complement or the OFF set of the functions which also increases the computational complexity. Moreover, as mentioned in [1], there are perfectly reasonable functions whose complements are perfectly unreasonable.

In this paper, we present a multiple output minimization algorithm which requires neither the complement nor all prime cubes of the functions. Consequently, it is well suited for minimizing functions with large complement size and/or very large number of prime cubes.

II. PRELIMINARIES

It is well known that a Boolean function of n-variables can be represented as a sum of product terms. Any product term may be the product of up to m literals ($1 \leq m \leq n$). Each of these product terms is known as an implicant and can be represented as a cluster of 1s on a Karnaugh map, the 1s being the minterms. These clusters can also be represented in an m-dimensional space where the clusters take the shape of cubes. It can be verified that a product term of m literals of an n-variable function ($1 \leq m \leq n$) will be a cluster of 2^{α} minterms where $\alpha = n - m$ and also will be a cube with 2^{α} vertices.

The cubes that belong to a Boolean function F can be grouped into F_{ON} and F_{DC} where F_{ON} is the set of true cubes and F_{DC} is the set of dontcare cubes. The complement of the function is denoted by F_{OFF}.

In this paper, we use the ternary notation (0, 1, 2) to represent the cubes and the product terms. In the input part, a cube will have a 1 for a literal in the true form, 0 for a literal in the complemented form. A 2 is used to indicate the literal that is eliminated. The status of the cube in the output function is represented by 0, 1, or 2 which indicates absent, true cube, or don'tcare cube, respectively.

The cubes are defined as follows.

Definition 2.1: A product term in the sum of products representation of a Boolean function which is also an implicant of the function will be called a *cube.* A cube can also be defined as an n-tuple $a = (a_1, a_2, a_3, \cdots, a_n)$ where $a_i \in \{0, 1, 2\}$.

Definition 2.2: A cube having 2^{α} vertices is said to have a *dimension* α. Hence, all minterms are cubes of dimension 0 and a cluster of 8 minterms on the Karnaugh map has a dimension 3. The minterms are the cubes with the least dimension (0) and will, therefore, be also called the *elementary cubes.*

Definition 2.3: Two cubes are said to be *adjacent* to each other if the ith variable is present as a true variable in one cube and as a complementary variable in the other while the values of other variables, if present in both the cubes, must be same.

Example 2.1: Consider the following single output function.

Cube $C_1 = 1002$; Cube $C_2 = 0200$;

Cube $C_3 = 0002$;

Cube $C_4 = 0011$; Cube $C_5 = 2101$.

In the above example, cubes C_1 and C_2, cubes C_1 and C_3, and cubes C_1 and C_5 are adjacent to each other while cubes C_1 and C_4 are not adjacent as they are differing in more than one variable.

Definition 2.4: A cube C_i is said to have a cube C_j as a *partially adjacent* cube if C_i and C_j differ in only one bit position and if no variable of C_i can be eliminated by combining C_i and C_j.

For instance, cubes C_1 and C_2 are adjacent in the first bit position, but cube C_1 cannot be expanded by considering only cube C_2 in the above example.

Definition 2.5: A cube C_i is said to have a cube C_j as a *completely adjacent* cube if C_i and C_j differ in only one variable and the differing variable can be eliminated by combining C_i and C_j.

In Example 2.1, cubes C_1 and C_3 are completely adjacent since the variable in the first bit position gets eliminated when cube C_1 is expanded using cube C_3.

However, a cube can also be completely expanded by using two or more partially adjacent cubes. Cube C_5 of Example 2.1 can be expanded by eliminating the variable in the second bit position if cubes C_1 and C_3, which are partially adjacent to cube C_5, are considered.

Definition 2.6: Two cubes C_i and C_j are said to be *intersecting* if both cubes share at least a single vertex. Their intersection is denoted by $C_i \cap C_j$.

Definition 2.7: A cube C_i which is completely contained within another cube C_j will be called a *subcube* of C_j and is denoted as $C_i \subseteq C_j$.

The following definition of prime cube is taken from [11].

Definition 2.8: A cube of a Boolean function F which cannot be expanded in any direction without admitting at least a single vertex of the complement or the OFF set of the function is called a *prime cube.*

Definition 2.9: If a prime cube has at least a single vertex covered by this and only this prime cube, then the prime cube is called an *essential prime cube (EPC).*

Definition 2.10: If a subcube generating an EPC belongs to only one of the functions F, then such an EPC is called an *exclusive essential prime cube (EEPC)* of the function F.

Definition 2.11: If every vertex of a prime cube is covered by other essential prime cubes then that prime cube is called a *redundant prime cube (RPC).*

Theorem 2.1 {Redundant Cube Theorem): A cube is redundant if all the 2^{m} combinations of the m eliminated variables in it are also present in the cubes that are intersecting with the given cube.

Proof: A cube with m variables eliminated covers 2^{m} vertices. As the intersecting cubes also cover all the 2^{m} vertices, the given cube is redundant. Also, cube C_i becomes redundant if it is completely covered by another cube C_j as $C_i \subseteq C_j$. Q.E.D.

Corollary 2.1: A redundant cube can be detected by considering only the cubes intersectng with it and ascertaining the presence of all the 2^{m} combinations of variables in the m detected variables positions of the given cube.

Dejinition 2.12: A cube C of dimension α ($0 \leq \alpha \leq n$) of an n-variable function will produce a *candidate product cube (CPC)* of dimension between α and n. The CPC is computed by deleting one variable at a time if C can completely expand in that variable.

Theorem 2.2 (Candidate Product Cube Theorem): The candidate product cube (CPC) generated by a cube C is the largest cover of C provided the CPC is completely present in the given function.

Proof: The candidate product cube is formed by considering the cubes that are either partially or completely adjacent to the cube generating the CPC. Hence, all the possible directions of expansion of the given cube have been considered while computing the CPC and the cube cannot be expanded in any other direction outside the CPC. Therefore, the CPC is the largest cover for the cube generating it, if it is completely present in the given function. Q.E.D.

The CPC is one of the important parameters required throughout the algorithm.

Dejinition 2.13: A prime cube which is neither an essential prime cube nor a redundant prime cube is a *selective prime cube (SPC).* In an SPC there is at least one vertex which is covered neither by any EPC nor by this and only this prime cube. Such a vertex is covered by at least one other SPC.

It is also obvious from the definition that the existence of one SPC implies the existence of at least another and the SPC's will also intersect at the cube (or subcube) generating them. When a cube generates only two SPC's, they appear as two interconnecting links of a chain. There may be cases where such a chain is constituted by a number of SPC's.

Dejinition 2.14: Among all the SPC's generated by a subcube C, if there exists one or more SPC's which cover all the uncovered cubes (subcubes) within the union of all these SPC's, then this (these) SPC(s) will be called a *valid selective prime cube (VSPC).*

Definition 2.15: If a subcube generating SPC's belongs to only one of the functions F, then the SPC's are the *exclusive SPC's (ESPC)* of the function F.

Dejinition 2.16: The cubes which are partially adjacent to the given cube but do not participate in the formation of the CPC are known as *cognate cubes.*

III. MINIMIZATION ALGORITHM

The multiple output minimization algorithm is a divide and conquer algorithm wherein the minimization is carried out by four main procedures:

1) SELECT–ESSENTIAL–PRIME–CUBES;
2) SELECT_VALID_SELECTIVE_PRIME_CUBES;
3) SELECT–INTERSECTING–CUBES;
4) SELECT–EXCLUSIVE–CUBES.

The first two procedures compute shared product terms (which belong to two or more functions) as well as exclusive product terms (which belong to only one of the functions). Procedure 3 selects only shared product terms while only exclusive product terms are obtained in procedure 4.

The cubes are read using the procedure READ–CUBES. The procedure also checks for duplicated cubes, null cubes, syntax errors in the input specification, etc. The cubes are stored using the linked list data structure of Pascal, where each node of the list represents a cube. Initially, the status of each cube in every function is set to either uncovered or don'tcare if it is present in the given function. A cube number is assigned to each cube in the list. The cubes are arranged according to their dimensions. Within each dimension, the cubes are ordered according to their weights (the number of 1's present in the cube). Further, cubes of the same weight are grouped with respect to the position of 2's, which implies that all cubes of the same group have 2's in the same bit positions. Thus the cubes differing in at least two bit positions are grouped together. This considerably reduces pairwise cube comparisons while computing the candidate product cubes. Also, loosely coupled cubes are easily identified in many cases with the help of this data structure.

During procedure SELECT–ESSENTIAL–PRIME–CUBES, the cubes are processed for computing the essential prime cubes in decreasing order of their dimensions. The first uncovered true cube in the list is selected. Its CPC is then computed. As the cubes of the same group differ in at least two bits, mutual comparisons of cubes within a particular group is avoided while computing the CPC. After the CPC has been computed, the algorithm checks whether the CPC is an essential prime cube by applying the following theorem.

Theorem 3.1 (Essential Prime Cube Theorem): A candidate product cube C is an essential prime cube of function F, if

i) $C \subseteq F_{ON} + F_{DC}$;
ii) there exists at least one subcube (may be an elementary cube) which does not have any cube, either partially or completely adjacent to it outside C.

Proof: A candidate product cube must be completely present in the given function if it is to be selected as a product term of the solution. A subcube with no other cube which is partially or completely adjacent to it implies that this subcube can be covered by this and only this CPC, and hence, the CPC is an essential prime cube of the function. Q.E.D.

Corollary 3.1: A candidate product cube of dimension α produced by a cube of dimension α or $\alpha - 1$ is completely present in the given function.

Proof: If the difference in dimensions of the candidate product cube and the cube generating it is 0, then the CPC is equal to the cube generating it. which is completely present in the given function. On the other hand, if the difference is 1, then one of the variables of the given cube has been eliminated which implies that the cube has expanded fully in that variable by combining with another completely adjacent cube or two or more partially adjacent cubes. Therefore, the CPC is completely present in the given function. Q.E.D.

Before ascertaining the presence of the CPC in the given function, a preprocessing detects the existence of the EPC

generating subcube. For this, the algorithm considers the cognate cubes. These cubes are identified and stored during the computation of the CPC. According to the EPC theorem, the CPC must contain at least one subcube which does not have any cube either partially or completely adjacent to it outside the CPC. This criterion can be verified by considering the values of eliminated variables of the given cube, if it is not an elementary cube, in the cognate cubes. (For an elementary cube, no cube can be a cognate cube since all the adjacent cubes participate in the generation of the CPC). If the cognate cubes contain all the 2^m combinations of the m eliminated variables of the cube, then the CPC is not an EPC, as this implies that every subcube has at least one partially adjacent cube outside the CPC. On the other hand, if at least a single combination is not present in the cognate cubes, the CPC is a potential EPC. The algorithm then ascertains whether the CPC is completely present in the given function considering both the true cubes and the dontcare cubes that are subsuming the CPC. The CPC is present in the given function if the disjoint sharp [11] of CPC with the intersecting cubes is null. If the CPC is completely present in the given function, the CPC is further processed to find, if it is an EEPC of the given function. If it is not an EEPC, then it is used for generating a possible shared product term. Once a product term is selected, it is printed along with the function(s) in which it is present. The cubes that are subsuming the CPC are flagged as covered and those that are intersecting are marked as partially covered along with the information regarding the portions covered. However, if the CPC is not selected as a product term, it is stored for possible future use. The procedure terminates after scanning the list of true cubes once. At the end of procedure **SELECT–ESSENTIAL–PRIME–CUBES**, both exclusive EPC's and shared EPC's have been selected.

Cubes contributing valid selective prime cubes (VSPC) are selected in procedure SELECT_VALID_SELECTIVE_ **PRIME–CUBES**. Since the exclusive and shared essential prime cubes have been covered, only the cubes generating the selective prime cubes are processed in this procedure. The algorithm starts with an uncovered cube (subcube) whose CPC has already been computed and stored during procedure SELECT–ESENTIAL–PRIME–CUBES. The CPC is the maximum possible cover for the cube generating the CPC. Now the uncovered cubes subsuming this CPC are considered for computing the VSPC. Let C_i be the cube being tried for generating the VSPC. Initially, C_{VSPC} = C,. For each cube C_j ($C_j \subseteq$ CPC of C_i and $C_i \subseteq$ CPC of C_j), the cube C_{VSPC} covering both C_i and C_j is updated as follows. The kth bit of C_{VSPC} ($k = 1, n$ for an n-variable function) is shown below:

C_{VSPC} \ C_j	0	1	2
0	0	2	2
1	2	1	2
2	2	2	2

C_{VSPC} is the minimum dimension cube required to cover all the uncovered cubes (subcubes) within the CPC of C_i. Hence, the algorithm checks for the existence of this cube. If C_{VSPC} is completely present in the given function, then it is the VSPC for C_i. Once the existence of the VSPC is ascertained, the algorithm checks whether the VSPC is an ESPC. If not, the algorithm tries to generate a shared product term if it is completely present in two or more functions and covers at least one uncovered vertex in every such function. The cubes covered by the VSPC are appropriately flagged. If a VSPC cannot be generated then the next cube in the list is tried. This procedure is repeated till no more VSPC's can be generated. An SPC generating subcube C_i which does not initially have a VSPC cover may have one after a subcube C_j has been covered. It is to be noted that CPC of $C_i \cap$ CPC of $C \neq$ nil. **As** the algorithm first tries to generate as many VSPC's as possible, arbitrary branching for the selection of SPC's is avoided in many cases. It has been observed that this not only saves CPU time but also results in minimal solution in many cases.

In procedure **SELECT–INTERSECTING–CUBES**, the shared product terms are selected according to a different criterion since shared EPC's and shared SPC's have been selected in the previous procedures. A common cube C_{COM} is formed by taking the intersection of CPC's of the given cube in different functions in which it is yet to be covered. For a given cube C_i in F_k, initially C_{COM} = CPC (C,). Each bit of C_{COM} is computed as follows. C_{COM} in every function F_j ($j \neq k$);

C_{COM} \ CPC (C_i)	0	1	2
0	0	—	0
1	—	1	1
2	0	1	2

C_{COM} is the largest cube which is common to these functions. Hence, this is selected as a shared product term and subsuming cubes are flagged as covered while those that are intersecting are flagged as partially covered. The list of true cubes is scanned for uncovered cubes before te - rminat ing this procedure.

It is evident that all the shared product terms have been selected by now. However, there may be some more cubes left uncovered or partially covered in different functions which cannot be covered by shared product terms. Therefore, procedure **SELECT–EXCLUSIVE–CUBES** is executed to cover such cubes. In this procedure the cubes are covered functionwise. In every function, the procedure first explores the possibility of forming EPC's out of the uncovered or partially covered cubes. Subsequently, if still some cubes remain uncovered, the concept of VSPC is applied and the cubes are covered. The procedure terminates after covering all the uncovered cubes.

PLA Name	In	Out	Cubes in	Essential cubes	Number of Output Cubes			Memory in K			CPU time in seconds		
					ESPR	McBL	Proposed algorithm	ESPR	McBL	Proposed algorithm	ESPR	McBL	Proposed algorithm
xor12	12	1	2048	2048	2048	2048	2048	6112	434	422	4467.2	40.6	9.9
xor10	10	1	512	512	512	512	512	998	190	234	403.0	8.4	2.3
alu1	12	8	19	19	19	19	19	232	656	182	4.4	531.3	3.2
pope	6	48	64	12	63	62	63	806	642	237	393.3	5239.1	144.9
co14	14	1	47	14	14	14	14	284	92	180	3.6	0.6	0.3
alu3	10	8	12	21	66	64	66	418	536	221	61.3	296.8	35.2
clpl	11	5	20	20	20	20	20	256	221	188	5.3	28.0	4.0
dk27	9	9	52	0	10	10	10	326	246	191	15.6	33.6	5.3
in2	19	10	137	85	137	134	137	558	514	275	114.3	184.3	71.0
gary	15	11	214	60	107	107	107	592	310	237	106.6	61.4	50.5
dk17	10	11	93	0	18	18	18	348	255	201	18.1	31.4	17.2
dc2	8	7	58	18	40	39	40	324	132	196	15.9	6.5	7.4
in0	15	11	138	60	107	107	107	584	304	240	124.6	49.3	95.7
in1	16	17	110	54	104	104	104	842	328	255	170.0	118.1	131.8
apla	10	12	134	0	26	25	25	424	238	206	27.1	31.3	33.6

In all the above procedures, the product terms are printed as soon as they are selected. The output will be in the form of personality matrix of the PLA.

IV. Examples

Example 4.1:

Initial Personality Matrix						
a	b	c	d	f_1	f_2	f_3
0	2	1	0	1	0	0
2	1	1	1	1	0	0
1	1	1	1	0	1	1
1	0	1	1	0	0	1
0	1	2	1	0	1	0
1	1	0	0	0	1	0
1	2	1	0	1	0	0
1	1	1	2	1	0	0
0	1	0	2	0	1	0
1	0	0	0	0	1	0
1	0	2	1	0	1	0

Output of the Program						
a	b	c	d	f_1	f_2	f_3
2	2	1	0	1	0	0
1	2	1	1	0	1	1
1	0	0	2	0	1	0
2	1	0	0	0	1	0
0	1	2	1	0	1	0
2	1	1	2	1	0	0

The cubes are processed in decreasing order of their dimensions. In procedure SELECT_ESSENTIAL_PRIME–CUBES, the CPC of cube 0210 in f_1 is computed. Its CPC 2210 is an EPC of f_1 and as the EPC generating cube is exclusive to f_1, 2210 is selected as an exclusive EPC of f_1. Similarly, CPC's of other uncovered cubes are computed and the possibility of selecting EPC's is exploited. Cube 1011 in f_3 has a CPC of 1211 (an EPC) which also covers some of the uncovered cubes of f_2, and hence, 1211 is selected as a shared product term off2 and f_3. In procedure SELECT–VALID–SELECTIVE–PRIME–CUBES, cube 1021 in f_2 has 1001 as an uncovered subcube. 1001 is tried for generating VSPC. Its CPC 1022 contains the other uncovered cube 1000. Hence, C_{VSPC} of 1001 is 1002. Instead of processing cube 1001, if 1000 is processed first, then its CPC 1202 contains uncovered cubes 1100 and 1001. Hence, C_{VSPC} of 1000 is 1202 which does not exist. However, as cube 1000 has two possible covers 1002 and 1200 which are equally preferable, if 1200 is selected after branching, a non-minimal solution is obtained. As the program first explores the possibility of generating VSPC's, no product term will be selected in such a situation. After cube 1001 generates a VSPC, cubes 1100 and 0121 in f_2 generate 1200 and 0121 as VSPC's, respectively. No cube is selected in procedure SELECT_INTERSECTING–CUBES. Finally, in procedure SELECT_EXCLUSIVE_CUBES, subcube 0111 generates an EPC 2112 in f_1.

Example 4.2

Initial Personality Matrix					
a	b	c	d	f_1	f_2
0	1	0	2	1	0
2	0	1	1	2	0
2	1	1	1	1	1
1	1	2	0	0	1
0	1	0	1	0	2
1	1	0	1	0	1
0	1	1	0	1	0

Output of the Program					
a	b	c	d	f_1	f_2
0	1	2	2	1	0
1	1	2	2	0	1
2	1	1	1	1	1

In procedure SELECT–ESSENTIAL–PRIME–CUBES, cube 0102 generates 0122 in f_1 and cube 1120 generates I122 in f_2. No product term is selected in procedure SELECT_VALID_SELECTIVE_PRIME_CUBES. In procedure SELECT–INTERSECTING–CUBES, cube 2111 is processed as it is uncovered in f_1 and f_2. Its CPC is 2211 in f_1 and 2121 in f_2. Hence, C_{COM} is 2111 which is selected as a shared product term.

V. Results

The multiple output minimization algorithm described in this paper has been implemented in Pascal [15]. The program is highly portable as it is transported from a DEC 1090 (on which it was developed) to a VAX 11/750 with very few modifications. The program has also been implemented on a HCL Workhorse 11, an 8086-based minicomputer. The program has been tested using a large number of PLA's including those of Berkeley PLA test set. In Table I, the results as obtained on a **VAX** 11/750 have been summarized. Results of EXPRESS0 II and McBOOLE have been compared on a **VAX** 11/750 and reported in [2].

It can be seen that the program has given the exact minimal solution in most of the cases. The quality of the result with respect to the number of product terms obtained by the proposed algorithm is quite satisfactory. An ESPRESSO II type of result has been obtained in most of the cases.

As pointed out in [2], the memory requirements of ESPRESSO II seem to be most strongly correlated to the size of the function and the size of the complement of the function. McBOOLE's memory requirements are directly correlated to the number of prime cubes and the number of variables. Memory required by the proposed algorithm is directly proportional to the number of input cubes and the number of variables as it does not generate either the complement or all the prime cubes.

The CPU time required by ESPRESSO II is related to the number of cubes in the function which are not essential prime cubes, the number of variables in the function, and the size of complement of the function while it is related to the number of prime cubes and the number of nested cycles in the function in the case of McBOOLE [2]. In the proposed algorithm it is related to the number of non-essential prime cubes and the valid selective prime cubes.

VI. Conclusions

The salient features of the algorithm presented in this paper include a fast technique for the determination of essential prime cubes and the introduction of a new class of selective prime cubes called valid selective prime cubes (VSPC). In many cases, VSPC's avoid branching which is computationally an expensive operation. McBOOLE computes alternate solutions after branching and the solution with the lowest cost is selected. However, when nested cycles are encountered, many branches might have to be computed and compared for obtaining the minimal solution. However, the number of branches grows exponentially with the number of nested cycles. If this exceeds a limit fixed by the user, then the minimal solution is not guaranteed by McBOOLE.

The other significant features of our algorithm are that it does not generate either the complement or all the prime cubes of the functions. Therefore, it is well suited to minimize functions with large complement size and/or having a large number of prime cubes. **A** comparison with ESPRESSO II and McBOOLE indicates that the program produces absolute minimal solution in most cases and near minimal in a few others. We believe that if the algorithm is implemented in C, it will be much faster than the present Pascal version as many features of C can be exploited.

Acknowledgment

The authors would like to thank Prof. **A.** L. Sangiovanni-Vincentelli and R. Rudell, University of California, Berkeley, for providing the Berkeley PLA test set. B. Gurunath acknowledges the useful discussions with **S.** Srinivas, Indian Institute **of** Science, Bangalore, India. The authors also thank the reviewers for their constructive criticism.

References

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

[2] M. R. Dagenais. V. K. Agarwal, and N. C. Rumin, "McBOOLE: A new procedure for exact logic minimization," *IEEE Trans. Computer-Aided Design,* vol. CAD-5, pp. 229–238, Jan. 1986.

[3] W. V. Quine, "A way to simplify truth functions." Amer. Math *Mon.*, vol. 62, pp. 627–631, Nov. 1955.

[4] E. J. McCluskey, "Minimization of Boolean functions." *Bell System* Tech. *J.*, vol. 35, pp. 1417–1444, Apr. 1956.

[5] T. *C.* Bartee, "Computer design **of** multiple output logical networks," *IRE* Trans. Elect. *Comput.*, vol. EC-10, pp. 21–30. Jan. 1961.

[6] N. N. Biswas, "Minimization of Boolean functions." *IEEE* ***Trans. Comput.***, vol. C-20, pp. 925–929, Aug. 1971.

[7] 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. *Comput.*, vol. C-26, pp. 757–764, Aug. 1977.

[8] P. Agrawal, V. D. Agrawal. and N. N. Biswas, "Multiple output minimization." in Proc. 22nd *Design* Automution *Conf.*, pp. 674–680, June 1985.

[9] N. N. Biswas, "Computer-aided minimization procedure for Boolean functions," *IEEE* Trans. Computer-Aided ***Design.*** vol. CAD-5. pp. 303–304, Apr. 1986.

[10] N. N. Biswas and B. Gurunath. "BANGALORE: An algorithm for the optimal minimization of programmable logic arrays," *Int.* **J.** Electron., vol. 60. pp. 709–725. June 1986.

[11] S. J. Hong. R. G. Cain. and D. L. Oatapko, "MINI: A heuristic approach for logic minimization." IBM *J. Res. Develop.*, vol. 18, pp. 443–458, Sept. 1974.

[12] T. Sasao, "HART: A hardware for logic minimization and verification," in Proc. *Int. Conf. on Computer Design,* pp. 713–718, 1985.

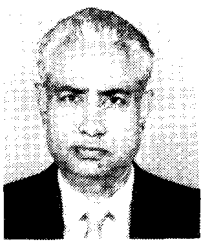
[13] R. E. Miller, ***Switching*** *Theory.* Vol *I: Combinatorial* Circuits. New York: Wiley, 1965.

[14] **Y.** Breitbart and **K.** Vairavan, "The computational complexity of a class of minimization algorithms for switching functions," *IEEE* Trans. *Comput.*, vol. C-28. pp. 941–943, Dec. 1979.

[15] B. Gurunath and N. N. Biswas, "An algorithm for multiple output minimization," in Proc. *IEEE Int. Conf.* ***on*** Computer-Aided *Design (ICCAD 87)*, pp. 74–77. Nov. 1987.

B. Gurunath (S'80–M'88) received the B.E. degree in electronics engineering from Bangalore University, Bangalore, India in 1981 and the Ph.D. degree from Indian Institute of Science, Bangalore, India in 1987

He worked as a System5 Engineer at Management Systems Analysts, Bangalore, until 1988 He was an Assistant Professor with Monmouth College, West Long Beach, NY during Spring 1989 Currently he is with Lambda-India. He was awarded travel grant by the Department of Science & Technology and Council of Scientific & Industrial Research, Government of India for presenting a paper at ICCAD-87. His research interests are in the areas of logic synthesis, computer networking and programming languages.



Vripendra N. Biswas (M'58–SM'65) received the B Sc (hons) degree in physics from the University of Calcutta in 1948. the B S and M S degrees from electronics and communication engineering from the Indian Institute ot Science in 1952 and 1954, respectively, and the Ph D. degree from the Indian Institute of Technology, Kharagpur, India, in 1960

He was on the faculty of the University of Roorkee, India, from 1955 *to* 1967 In 1967, he Joined the faculty ot St Louis University, MO In 1970, he became a professor of electrical communication engineering and computer science at the Indian Institute of Science, Bangalore, India In the Summer and Fall of 1987 he visited USA as a visiting Scientist at the AT&T Bell Laboratories, Murray Hill, NJ and as a Visiting Professor of Electrical and Computer Engineering at the Syracuse University From 1987 1988 he was a visiting professor at the Center tor Advanced Com uter Studies University of Southwestern Louisiana Lafayette Currently he is Professor [illegible] at the Indian Institute of Science Bangalore He has published many papers in technical journals 25 of which [illegible] published six books in the areas of communication engineering [illegible] ming Languages and logic and switching [illegible]