AN EXACT ALGORITHM TO MINIMIZE MIXED-RADIX EXCLUSIVE SUMS OF PRODUCTS FOR INCOMPLETELY SPECIFIED BOOLEAN FUNCTIONS

Marek Perkowski, and Malgorzata Chrzanowska-Jeske

Portland State University, Department of Electrical Engineering
P.O. Box 751, Portland, OR, 97207, tel. (503) 725-3806 x23

ABSTRACT

An exact algorithm for the synthesis of mixed polarity Exclusive Sum Of Product (ESOP) expressions for arbitrary size incompletely specified Boolean functions is presented. For more than 4 input variables, this problem has not been solved yet. A decision function H is constructed for a Boolean function f that describes all possible ESOP solutions to f. The function H plays for the ESOP minimization problem the role analogous to that of the Petrick function for the minimization of the inclusive sum of product expression problem of the classical logic. Each product of literals that satisfies the function H corresponds to one ESOP solution of f. The algorithms to create and solve function H are presented.

1. INTRODUCTION

Various circuit realizations of Boolean functions that use a high percentage of Exor gates have been proposed. All of them will be called the *Exor circuits*. They include *Reed-Muller forms (RM)* [20,30], *Generalized Reed-Muller forms (GRM)* [10,19], *Exclusive Sums of Products* [12,31] *Multi-Valued Input Exclusive Sums of Products* [24] *Multi-Valued Generalized Reed-Muller Trees (MGRM)* [25], and many other multi-level and multi-output realizations in which an Exor gate is repeatedly used as a basic logic component. Out of those the RM, GRM and MGRM are canonical forms. Such Exor circuits have several advantages over the realizations which include only other kinds of gates, for instance the circuits that use only NAND and NOR gates. The main advantage of the Exor circuits is their essentially improved testability: for those of them that are canonical forms, the test generation is very easy, and a number of tests is small. Moreover, for some of them (like the Reed-Muller or generalized Reed-Muller forms) the set of all tests is universal and does not depend on a particular Boolean function that is realized by the circuit. Also, the Exor-based circuits often require smaller layout area for their realization on VLSI chips. This is especially common for arithmetic, error detection/correction and telecommunication circuits. However, two perceived disadvantages of Exor circuits cause that they are not as often used in practical applications as they deserve. The first disadvantage is the lack of a good minimization theory and corresponding practical minimization algorithms. This problem has been addressed in many papers in the last few years and several efficient approximate algorithms have been recently programmed or proposed. The other, much more essential, disadvantage of the Exor circuits is the slow speed of the many-input Exor gate. There are, however, some hopes that this will be overcome in new technologies, including the logic families based on current switching [5], and the optical technology realizations [11,32].

Reed-Muller forms and other Exor circuits are useful outside the area of circuit design. They can serve as *efficient data structures* for logical forms manipulation in symbol processing for CAD [28] as well as in the *unification*, being a base of automatic theorem provers and logic programming languages like Prolog. They have been also used in *image processing, coding, and recognition* problems [29]. The above reasons have recently caused an increased interest in the development of the theory of Exor circuits that would find practical applications. The classical but yet unsolved problems of finding the minimal forms become more interesting again.

The basic circuit model used in this paper can be formulated as follows. Given is a single output, incompletely specified Boolean function f of n binary inputs in the form of a Karnaugh map, or a set ON of *true minterms* (called sometimes the "ones" of the function) and a set OFF of *false minterms* (the "zeros" of the function). The Mixed Polarity (Radix) Exclusive Sum Of Products (*ESOP* [31], or *MRESP* [12,24]) of a Boolean function is an Exclusive sum (Exor) of *products of literals*, each *literal* being a *variable* or its negation. The same variable can stand in both *positive* and *complemented* forms in various products of the ESOP. This formulation is the most general basis for a two level Exor/And-based realization of a general purpose Boolean function. The other forms of such a realization include the Reed-Muller and so-called "generalized Reed-Muller forms". In a Reed-Muller form all variables have positive *polarities*. GRM forms are created for all possible *polarities* of the input variables. Each variable in a GRM can be either positive *(positive polarity, 1)* or complemented *(negative polarity, 0)*, but it cannot have both polarities in one form. Since there are two polarities for each of the n input variables there are 2^n polarities of all variables and, therefore, there exist 2^n various GRM forms out of which one with the least cost should be selected.

In the ESOP minimization problem one wants to find a solution with the *minimum number of products* or with the *minimum total literal cost*. As mentioned above, the problem to find an exact minimum solution for a Boolean function in an ESOP form is an old one [1, 4, 6, 7, 8, 10, 16, 17, 19, 28], but it still remains unsolved, even for the case of completely specified Boolean functions. Several papers that propose theories and programs to produce quasiminimum solutions have been published. With the exception of a paper by Papakonstantinou [21] who shows how to find a minimum solution for four variable functions, nobody has attempted to formulate an exact algorithm for a Boolean function f with an arbitrary number of input variables.

In this paper an algorithm to solve this important problem is presented for the first time. It finds always the exact solution and has no theoretical limits on the size of the function. Moreover, it finds the exact minimum solution for incompletely specified functions as well. Interestingly, for a function with a high percentage of don't cares the solution can be found even more efficiently, as opposed to most algorithms for the sum of products (SOP) realization of Boolean functions. Finally, although the approach proposed here is very time consuming, its speed can be essentially increased by using parallel processing techniques or the special VLSI data-flow hardware accelerator described in [14].

Section 2 introduces *the decision function H* that finds applications in all *odd/even covering problems*. It is a direct counterpart of the well-known *Petrick function* used in *a set covering problem*. Creation and use of the function H is illustrated with examples. Section 3 presents various methods to minimize such functions.

2. THE HELLIWELL'S DECISION FUNCTION FOR EVEN/ODD COVERING PROBLEMS

The basic idea of this method is to create, for a given Boolean function $f(x_1, x_2, ... x_n)$, a new Boolean function $H(g_1, g_2, ..., g_K)$, where $K \leq 3^n$, that describes all possible solutions to the function f. The name H comes after Martin Helliwell who has first proposed this kind of decision function [13].

Definition 2.1. A *literal* is a *variable* with negation or not. It is assumed that $\bar{x} = x^0$ (*negative literal*) and $x = x^1$ (*positive literal*).

Let us denote by $G = \{g_1, g_2, ..., g_K\}$ the set of all *decision variables*. All those variables are used in the function H only as *positive literals*.

The function H is created as follows. First one finds all possible *products of literals* from the set $\{x_1^{i_1}, x_2^{i_2}, ... x_n^{i_n}\}$, where $i_j \in \{0,1\}$ for j=1, ... n, that can be used as *product groups* in the ESOP minimization problem of f. A product group is denoted as G_j, and the set of all such groups will be denoted by GG. In the worst case GG is the set of all 3^n possible literal products calculated by *ternary counting*. Ternary counting creates all possible *cubes* with variable x_i position values: 0 - for $\bar{x}_i$, 1 - for x_i, and X for x_i absent in the product. Next, a unique *new variable (a positive literal)* $g_1, g_2, \dots, g_K$ is assigned to each of the product groups from GG.

A *true minterm* (a "one") is denoted by m_i, a *false minterm* (a "zero") by M_i. The set of all true minterms (or, in general ON-cubes) is denoted by ON. The set of all false minterms (OFF-cubes) is denoted by OFF.

Now, the formal method of creating the decision function H for the function f and the proof of its usefulness in minimization will be given.

Definition 2.2. A *redundant ESOP* is an ESOP that includes products that can be removed and the ESOP will still be a solution. A *non-redundant ESOP* is an ESOP that is not redundant.

Example 2.1. If f is a solution then $f_1 = f \oplus ab \oplus a\bar{b} \oplus a$ is a redundant solution since: $f_1 = f \oplus a \oplus a, f_1 = f \oplus 0, \ f_1 = f$.

Theorem 2.1. Each non-redundant ESOP solution of the function f is an Exor term of groups G_j corresponding to those decision variables g_j that stand as positive literals in a product that satisfies the following logic equation:

$$1 \ = H(g_1, ..., g_K) = \tag{2.1}$$

$$\prod_{m_i \in ON} \Big(\sum^{\oplus}_{\substack{g_j \text{ is a variable} \\ \text{for a product group } G_j \supseteq m_i \\ G_j \in GG}} g_j \Big) \cdot \prod_{M_i \in OFF} \Big(1 \ \oplus \sum^{\oplus}_{\substack{g_j \text{ is a variable} \\ \text{for a product group } G_j \supseteq M_i \\ G_j \in GG}} g_j \Big)$$

Proof. To describe covering possibilities of a single true minterm m_i one has to create an Exor term:

$$H(m_i) = \sum^{\oplus}_{g_j \text{ is a variable for a product group } G_j \supseteq m_i} g_j \tag{2.2}$$

which is an Exor of the variables g_j describing al the product groups G_j that cover the true minterm m_i. If the product group G_j is selected, a value one is assigned to its variable g_j. If the group G_j is not selected, a value 0 is assigned to g_j. If an even number of g_j variables is selected the logic ones assigned to them will anihilate to zero in the *Exor term*. If an odd number of g_j variables is selected then the resultant value of the Exor term will be 1. One can now create a product H_1 of such Exor terms $H(m_i)$ for all true minterms of f:

$$H_1 = \prod_{m_i \in ON} H(m_i) \tag{2.3}$$

Similarly as in the Petrick function, the product of such terms equals 1 whenever all true minterms are covered.

Function H_1 is, however, not sufficient as a satisfiability formula since one has also to take into account that while all the true minterms are covered by the selected groups from GG, all the false minterms must not be covered by these groups. This can be done for a single false minterm M_i by an exor of value 1 with an Exor of all variables g_j for groups G_j that cover this false minterm. Therefore, the Exor term for a false minterm M_i has the form:

$$H(M_i) = 1 \ \oplus \sum^{\oplus}_{g_j \text{ is a variable for a group } G_j \supseteq M_i} g_j \tag{2.4}$$

The function for all false minterms of a function f will be a product of the above Exor terms:

$$H_2 = \prod_{M_i \in OFF} H(M_i) \tag{2.5}$$

Now, the Boolean product of functions H_1 and H_2 :

$$H(g_1, ..., g_K) = H_1 \cdot H_2 \tag{2.6}$$

will be 1 when all true minterms of f are covered and all false minterms of f are not covered by groups G_j. The equation

$$H(g_1, ..., g_K) = H_1 \cdot H_2 \ = \ 1 \tag{2.7}$$

is therefore a *generator of all non-redundant solutions.* Q.E.D.

In the first approach, presented in this paper, it is assumed that the product groups G_j correspond to *all possible extensions* of the true and false minterms of f. (More efficient approach is presented in [26, 27]). These extensions are created by the removal from the minterms of all possible subsets of $x_i^{i_j}$ literals (also empty subsets). For instance, for a minterm $a\bar{b}$ one creates product groups: $G_1 = a\bar{b}$, $G_2 = a$, $G_3 = \bar{b}$, $G_0 = 1$. The value 1 is *an universe* - the entire Karnaugh map of a function f. The corresponding variables of H, assigned to groups G_1, G_2, G_3, and G_0, will be g_1, g_2, g_3, and g_0, respectively.

Example 2.2. Given is a completely specified function $f(a,b) = \sum 1,2,3$. The product groups are: $G_0 = 1$, $G_1 = a\bar{b}$, $G_2 = a$, $G_3 = \bar{b}$, $G_4 = ab$, $G_5 = b$, $G_6 = \bar{a}b$, $G_7 = \bar{a}$, $G_8 = \bar{a}\bar{b}$.

All product groups that can be used for the realization of true minterm $a\bar{b}$ are: G_0, G_1, G_2, G_3. All product groups that can be used for the realization of the true minterm ab are: G_0, G_2, G_4, G_5. All product groups that can be used for the realization of the true minterm $\bar{a}b$ are: G_0, G_5, G_6, G_7. All product groups that can be used for the realization of the false minterm $\bar{a}\bar{b}$ are: G_0, G_3, G_7, G_8.

Function H_1 is therefore:

$$H_1 = (g_0 \oplus g_1 \oplus g_2 \oplus g_3) \cdot (g_0 \oplus g_2 \oplus g_4 \oplus g_5) \cdot (g_0 \oplus g_5 \oplus g_6 \oplus g_7).$$

Function H_2 is:

$$H_2 = (1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8).$$

Therefore:

$$H = H_1 \cdot H_2 = (g_0 \oplus g_1 \oplus g_2 \oplus g_3) \cdot$$
$$(g_0 \oplus g_2 \oplus g_4 \oplus g_5) \cdot (g_0 \oplus g_5 \oplus g_6 \oplus g_7) \cdot (1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8).$$

Definition 2.3. The Exor term that includes logic 1 is called the *1-term.* The Exor term that has no logic 1 is called *0-term.*

The first three terms in the above expression are 0-terms and the last one is a 1-term. Let us observe that in the initial function H 1-terms correspond to false minterms and 0-terms *to true minterms of the function f.*

Example 2.3 In the next section we will systematically describe methods to solve (minimize) this kind of decision functions. Let us, however, observe now that by selecting variable g_0 (it means substituting logic value 1 for variable g_0) function H obtains the form:

$[H \mid_{g_0 = 1}] =$

$$(1 \oplus g_1 \oplus g_2 \oplus g_3) \cdot (1 \oplus g_2 \oplus g_4 \oplus g_5) \cdot (1 \oplus g_5 \oplus g_6 \oplus g_7) \cdot (1 \oplus 1 \oplus g_3 \oplus g_7 \oplus g_8)$$
$$= (1 \oplus g_1 \oplus g_2 \oplus g_3) \cdot (1 \oplus g_2 \oplus g_4 \oplus g_5) \cdot (1 \oplus g_5 \oplus g_6 \oplus g_7) \cdot (g_3 \oplus g_7 \oplus g_8)$$

The above formula would be satisfied if all variables from the first three terms were not selected and a single variable from the last term was selected. Since variables g_3 and g_7 occur in terms 1 and 3, they cannot be selected. Variable g_8 can be selected and the function H becomes:

$[H \mid_{g_0 \cdot g_8 = 1}] =$

$$(1 \oplus g_1 \oplus g_2 \oplus g_3) \cdot (1 \oplus g_2 \oplus g_4 \oplus g_5) \cdot (1 \oplus g_5 \oplus g_6 \oplus g_7) \cdot (g_3 \oplus g_7 \oplus 1)$$

In the above function each Exor term includes a logic 1 (is a 1-term). This means that the termination condition is satisfied and all the remaining variables in H are substituted by logic 0. After substitution:

$$[H \mid_{g_0 \cdot \bar{g}_1 \cdot \bar{g}_2 \cdot \bar{g}_3 \cdot \bar{g}_4 \cdot \bar{g}_5 \cdot \bar{g}_6 \cdot \bar{g}_7 \cdot g_8 = 1}] = (1)\cdot(1)\cdot(1)\cdot(1) = 1.$$

It was shown above that the product $g_0 \bar{g}_1 \bar{g}_2 \bar{g}_3 \bar{g}_4 \bar{g}_5 \bar{g}_6 \bar{g}_7 g_8 = 1$ satisfies H. Therefore $G_0 \oplus G_8 = 1 \oplus \bar{a}\bar{b}$ is an ESOP solution to f. It has one product term and a term 1, so this is a solution of *cost* 2 (the cost of each product group is 1) or of $cost_1$ 1 (the cost of each product group other than one is 1). In a similar way, it can be easily verified in H that the only two solutions of f with two product terms (and no term 1) are: $G_2 \oplus G_6 = a \oplus \bar{a}b$ and $G_1 \oplus G_5 = a\bar{b} \oplus b$, both of them of cost 2.

Among all solutions to function H one selects those that minimize certain *cost function.* The simpliest three cost functions used by us are:

1. *cost* = number of all ESOP product groups, together with logic 1.
2. $cost_1$ = number of ESOP product groups other than logic 1.
2. $cost_2$ = a total number of literals in product groups of ESOP.

Also, any kind of a linear combination of $cost_1$ and $cost_2$ can be used, as well as any *additive cost function* that does not decrease with the addition of product terms to ESOP. All such cost functions can be easily used for cut-off in tree-searching methods described in section 3.

The meaning of variables g_j is as follows. The *selection* of a given product group G_j to some ESOP solution of f corresponds to assigning the logic value 1 to variable g_j in H assigned to this product group. Not selecting a product group to ESOP corresponds to assigning the value 0 to the respective variable g_j in H. Each ESOP solution of f corresponds to a product of literals $g_j^{i_j}$ of H that logically satisfies the function H. The product of $g_j^{i_j}$ literals which satisfies H and has the minimum cost ($cost_1$, $cost_2$, ...) corresponds to an exact minimum solution of a function f. n the simpliest case when the number of groups G_j is minimized, the solution is sought that has the minimum number of selected variables g_j. The ESOP solution is an *Exor sum* of all product groups G_j that correspond to positive literals g_j from a product satisfying H.

The role of the function H is analogical to that of the *Petrick function* used in the exact minimization of SOP forms, which is well-known from many logic design textbooks [14,18], and is a particular case of the *Satisfiability Formula* of the program complexity theory [9] (The Satisfiability Formula allows complemented variables as well). Several algorithms for Petrick function minimization and satisfiability decision making have been already proposed, implemented and analyzed by many authors [14,]. All these algorithms can be found useful while creating algorithms to minimize the function H.

The form of Helliwell's function H is similar to the Petrick function P as well, but an Exclusive summing (Exor) of the decision variables g_j is used in H instead of their Inclusive Or. In the classical SOP minimization, a set of *prime implicants* of f is first found, and next the Petrick function P is created as a product of inclusive sums of positive literals. Each literal is assigned to a prime implicant of f. Each inclusive sum from P corresponds to a single true minterm m_i of the function f. This sum contains variables corresponding to prime implicants of f that cover m_i. In the ESOP minimization problem the creation of the decision function is more complicated because for such circuits there is no concept of a prime implicant. Therefore, any product of literals that covers a true minterm (i.e. a group from GG) can be theoretically considered as a potential candidate for the exact optimum cover. Also, in the Petrick function the inclusive sums of variables are used, since a repeated covering of a true minterm by the prime implicant is allowed. In the exor-based design the product groups are not summed but exored, therefore an odd number of groups covering a Karnaugh map cell produces *a true minterm m_i,* while an even number of products covering a cell produces a "zero" *a false minterm M_i.*

Let us observe, that it is not necessary to take the don't cares of f into account while creating the function H. This means that when the ratio of the true and false minterms to the don't cares decreases, the complexity of the function H decreases as well. Such functions with a high percentage of "don't cares" - the strongly unspecified Boolean functions - are recently of an increased interest with respect to their usefulness in a multi-level logic synthesis (Brayton and U.C. Berkeley research) [2,3]. Our approach to ESOP minimization is an interesting example of a logic synthesis algorithm which becomes relatively more efficient for the case of incompletely specified functions.

The algorithm to find an exact minimum solution to the function f can be now formulated as follows.

Algorithm 2.1.

1. For each true minterm $m_i \in$ ON find the set GG(m_i) of its all *extensions,* i.e. literal products that can be found by the removal of any (also empty) subset of literals from m_i.
2. $GG_1 := \bigcup_{m_i \in ON} GG(m_i).$
3. For each false minterm $M_i \in$ OFF find set GG(M_i) of all its extensions.
4. $GG_2 := \bigcup_{M_i \in OFF} GG(M_i).$
5. Assign uniquely variables g_j *to groups from* GG = $GG_1 \cup GG_2$. Create set G of these variables.
6. For each $m_i \in ON$ create $H(m_i) = \sum^{\oplus}_{g_j \text{ is a variable for a product group } G_j \in GG(m_i)} g_j.$
7. For each $M_i \in OFF$ create $H(M_i) = (1 \oplus \sum^{\oplus}_{g_j \text{ is a variable for a product group } G_j \in GG(M_i)} g_j).$
8. Create the function H according to formula (2.6).
9. Find the set *MIN_SOL* of all the minimal solutions to formula (2.6). Each element of the set *MIN_SOL* is a product of variables g_j.
10. For each solution product $\prod_j g_j \in$ *MIN_SOL* create an ESOP solution expression $\sum^{\oplus}_j G_j$.

In another variant of the above algorithm one looks for a first subset of all minimal solutions only.

3. COMPUTER ALGORITHMS TO SOLVE THE HELLIWELL'S FUNCTION.

There are several possible methods to find all literal products satisfying the function H:

1. Conversion of the function H to an equivalent Generalized Propositional Formula (GPF) [14] *and solving the GPF formula: by the method of tree search, by the Boolean* manipulation, or by any other method.
2. The Boolean manipulation of the function H that transforms this function to the form of an Exor of products of variables, from which all solutions can be found.
3. Direct tree search of the function H to find the solutions, i.e. the products of literals satisfying the function H.
4. The sophisticated algorithm of tree search with several sorting/selecting heuristics and heuristic evaluation functions that operates on ON and OFF cube array representation of the function f (with cubes being not minterms) [25,27]. This algorithm makes also use of the theory how to simplify the function H, by introducing fundamental concepts of elementary transformations, implixors, minimal implicants and implicates.

The first three above methods will be described in the sequel.

3.1. Conversion of the function H to an equivalent GPF formula.

Let us first consider, how an Exor of term can be transformed to an equivalent Inclusive OR of Product of Literals. For instance, the Exor term $g_1 \oplus g_2 \oplus g_3 \oplus g_4$, being an *odd function,* is replaced by an Inclusive Sum of Products:

$g_1 \bar{g}_2 \underline{\bar{g}_3} \bar{g}_4 \cup \bar{g}_1 g_2 \bar{g}_3 \underline{\bar{g}_4} \cup \bar{g}_1 \bar{g}_2 g_3 \bar{g}_4 \cup \bar{g}_1 \bar{g}_2 \bar{g}_3 g_4 \cup \bar{g}_1 g_2 g_3 g_4 \cup g_1 \bar{g}_2 g_3 g_4 \cup$
$g_1 g_2 g_3 g_4 \cup g_1 g_2 g_3 g_4$ composed of products having odd numbers of positive literals.

If there is a logic 1 in the Exor term, then the logic 1 is treated as one of the variables, i.e. the term

$(1 \oplus g_1 \oplus g_2)$

is replaced by

$(1 \bar{g}_1 \bar{g}_2 + \bar{1} g_1 \bar{g}_2 + \bar{1} \bar{g}_1 g_2 + 1 g_1 g_2) = (\bar{g}_1 \bar{g}_2 + g_1 g_2).$

In an equivalent method, the Exor term of 1 and decision variables is replaced with an Inclusive Or of the products of the decision variables, where all the products are created only for g_j variables (logic 1 is not treated as a variable). Since such a function (complement of Exor) is an even function, each of the created products includes an **even** number of positive literals, and the literals for the remaining variables are negative (Zero is treated as an even number). For instance, the term $(1 \oplus g_1 \oplus g_2 \oplus g_3)$ is replaced by

$(\bar{g}_1 \bar{g}_2 \bar{g}_3 + g_1 g_2 \bar{g}_3 + g_1 \bar{g}_2 g_3 + \bar{g}_1 g_2 g_3).$

Let us first consider, how the function H being a product of Exors of terms can be transformed to an equivalent product of Inclusive ORs of Products of Literals.

Theorem 3.1. Function H from formula (2.1) is logically equivalent to the following GPF formula:

$$H(g_1, \dots, g_K) = \tag{3.1}$$

$$\prod_{\substack{\text{for all Exor terms} \\ E_r = \sum\oplus g_{r,i} \\ i=1,\dots,v_r}} \Big(\bigcup_{\substack{m \in 0,1,\dots,2^{v_r}-1 \\ deg(m) \text{ is odd}}} [g_{r,1}, g_{r,2}, \dots, g_{r,v_r}]^m \Big)$$

$$\cdot \prod_{\substack{\text{for all Exor terms} \\ E_s = (1 \oplus \sum\oplus g_{s,i}) \\ i=1,\dots,v_s}} \Big(\bigcup_{\substack{m \in 0,1,\dots,2^{v_s}-1 \\ deg(m) \text{ is even}, deg(m)=0,2,\dots}} [g_{s,1}, g_{s,2}, \dots, g_{s,v_s}]^m \Big)$$

Proof. Let us observe that Exor is an odd function, so that the *Exor term* $E = \sum_{j=1,\dots,v} \oplus\ g_j$ of decision variables g_j can be replaced by an *Inclusive Or of Products* of corresponding decision literals $g_j^{i_j}$, where all the products are created for variables g_j, j = 1, ..., v. Each of the created products includes an **odd** number of positive literals, and the literals for the remaining variables are negative.

Let m = $[m_1, m_2, \dots, m_v]$ be a binary number of v bits where $m_i \in \{0,1\}$ and $[g_1, g_2, \dots, g_v]^m = [g_1, g_2, \dots, g_v]^{[m_1, m_2, \dots, m_v]} = g_1^{m_1}\, g_2^{m_2} \cdots g_v^{m_v}$

Let $deg(m)$ be a number of bits equal one in vector m.

Then for the 0-term of variables one gets:

$$\sum_{i=1,\dots,v} \oplus g_i = \bigcup_{\substack{m \in 0,1,\dots,2^v-1 \\ deg(m) \text{ is odd}}} [g_1, g_2, \dots, g_v]^m \tag{3.2}$$

Analogously, for the 1-term the Inclusive Or of Products of literals is:

$$1 \oplus \sum_{i=1,\dots,v} \oplus g_i = \bigcup_{\substack{m \in 0,1,\dots,2^v-1 \\ deg(m) \text{ is even}, deg(m)=0,2,\dots}} [g_1, g_2, \dots, g_v]^m \tag{3.3}$$

By using (3.2) and (3.3) each Exor term of H is, therefore, converted to an Inclusive Or of products of literals. The whole function H has been then transformed to the form of the *Product of Sums of Products of Literals.* Formula (3.1) is obtained from formulas (2.1), (3.2) and (3.3). Q.E.D.

Formula (3.1) is a particular case of the *Generalized Propositional Formula (GPF)* introduced in [14,].

Example 3.1. For the function H from Example 2.2 the Inclusive Or terms corresponding to the Exor terms are as follows.

For $t_1 = (g_0 \oplus g_1 \oplus g_2 \oplus g_3)$ the new term is

$\text{GPF}(t_1) = (g_0 \bar{g}_1 \bar{g}_2 \bar{g}_3 + \bar{g}_0 g_1 \bar{g}_2 \bar{g}_3 + \bar{g}_0 \bar{g}_1 g_2 \bar{g}_3 + \bar{g}_0 \bar{g}_1 \bar{g}_2 g_3 +$
$g_0 g_1 g_2 g_3 + g_0 g_1 g_2 g_3 + g_0 g_1 g_2 g_3 + g_0 g_1 g_2 g_3)$.

For $t_2 = (g_0 \oplus g_2 \oplus g_4 \oplus g_5)$ the new term is

$\text{GPF}(t_2) = (g_0 \bar{g}_2 \bar{g}_4 \bar{g}_5 + \bar{g}_0 g_2 \bar{g}_4 \bar{g}_5 + \bar{g}_0 \bar{g}_2 g_4 \bar{g}_5 + \bar{g}_0 \bar{g}_2 \bar{g}_4 g_5 +$
$g_0 g_2 g_4 g_5 + g_0 g_2 g_4 g_5 + g_0 g_2 g_4 g_5 + g_0 g_2 g_4 g_5)$.

For $t_3 = (g_0 \oplus g_5 \oplus g_6 \oplus g_7)$ the new term is

$\text{GPF}(t_3) = (g_0 \bar{g}_5 \bar{g}_6 \bar{g}_7 + \bar{g}_0 g_5 \bar{g}_6 \bar{g}_7 + \bar{g}_0 \bar{g}_5 g_6 \bar{g}_7 + \bar{g}_0 \bar{g}_5 \bar{g}_6 g_7 +$
$g_0 g_5 g_6 g_7 + g_0 g_5 g_6 g_7 + g_0 g_5 g_6 g_7 + g_0 g_5 g_6 g_7)$.

For $t_4 = (1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$ the new term is

$\text{GPF}(t_4) = (\bar{g}_0 \bar{g}_3 \bar{g}_7 \bar{g}_8 + g_0 g_3 \bar{g}_7 \bar{g}_8 + g_0 \bar{g}_3 g_7 \bar{g}_8 +$
$g_0 g_3 g_7 g_8 + g_0 g_3 g_7 g_8 + g_0 g_3 g_7 g_8 + g_0 g_3 g_7 g_8 + g_0 g_3 g_7 g_8)$.

Therefore, a GPF equivalent to H is:

$$GPF = GPF(t_1) \cdot GPF(t_2) \cdot GPF(t_3) \cdot GPF(t_4)\,.$$

By repetitive application of standard Boolean rules (like $a(b+c)=ab+ac$, $\bar{a} \cdot a=0$, or $a+a=a$) the above GPF is converted to an inclusive sum of products of positive and negative literals. Each such product describes a single solution to the GPF and hence to H. One with the smallest cost is selected.

Another method to solve GPF is the heuristic recursive simplification based on: variable selection, substitution the value 1 for it, and subsequent simplification of the decision function. For instance, an analysis of the GPF using this method, demonstrates that the products:

$g_0 \bar{g}_1 \bar{g}_2 \bar{g}_3 \bar{g}_4 \bar{g}_5 \bar{g}_6 \bar{g}_7 g_8$, $\bar{g}_0 \bar{g}_1 g_2 \bar{g}_3 \bar{g}_4 \bar{g}_5 g_6 \bar{g}_7 \bar{g}_8$, and
$g_0 g_1 g_2 g_3 g_4 g_5 g_6 g_7 g_8$

satisfy this GPF and hence are the solutions to it.

The *decision problem* for GPF is, like for the Petrick function (Product of Sums - POS), NP-complete [9]. Similarly the optimization problems for the GPF, POS and H functions are NP-hard. It is, extremely unlikely that an efficient (polynomial) algorithm will be ever found for the function H manipulation. However, among such NP-hard algorithms, there are still some that are more efficient and can be applied practically for problems of smaller dimensions. Formula manipulation, recursive substitution/simplification, and tree searching programs have also been created for exact and approximate minimization [27]. A special data-flow wavefront array architecture has been designed for this task [14, 15]. Discussion of the efficiency of a class of algorithms for GPF minimization/decision is given, together with the presentation of several program benchmarks in [15].

Theorem 3.2. The minimum solution to the function *GPF* (3.1) is the minimum solution *to the function H* (2.1).

Proof. Since function *GPF* is equivalent to the function H ($GPF \equiv H$ is a Boolean tautology), each minimum solution to *GPF* is a minimum solution to the function H as well. Q.E.D.

This method is the most time and memory consuming out of the methods presented in this paper and its only advantage is a reduction of the function H minimization problem to the well-known problem of *GPF* minimization for which computer programs exist and computer architectures have been proposed.

3.2. Boolean Manipulation.

Boolean manipulation method transforms the function H to a "flattened decision function" FH being an Exor sum of Products form. The transformation is based on application of the following rules of Boolean algebra:

$$a(bc)=(ab)c. \tag{3.4}$$

$$(a\oplus b)c=ac\oplus bc. \tag{3.5}$$

$$a(b\oplus c)=ab\oplus ac.$$

$$a\oplus a=0. \tag{3.6}$$

$$a\oplus 0=a. \tag{3.7}$$

$$aa=a. \tag{3.8}$$

$$ba=ab. \tag{3.9}$$

$$1a=a1=a. \tag{3.10}$$

$$b\oplus a=a\oplus b. \tag{3.11}$$

The function H is processed by the program from left to right. Starting from left, rule (3.4) is applied to every first two *Exor terms (terms* for short). For each such pair of terms rules (3.5) are applied creating a new term t. Rules (3.8), (3.9), (3.10) are used. All pairs of repeated products in term t are removed by applying rule (3.6) followed by rule (3.7). The products are sorted using rule (3.11). Rule (3.4) is applied to term t and the next term in H and the algorithm is repeated until all terms in H are exhausted. This is all programmed in a symbol manipulation program that permits for combining the above rules together in order to increase the processing efficiency.

The final formula FH is in the form of an Exor of products of variables. It can be converted to an equivalent *Inclusive Or of Products Form (IOF)* as follows:

1. New form FH' is created by removing from FH all products that are included in other products of FH. Such products can be removed because the solutions corresponding to them, even if they exist, would be inferior than the solutions corresponding to the cubes subsuming them.
2. Create an Inclusive Or Form (IOF) in which each product P_r is a logical AND of a product from the FH' form and the product of the negations of the remaining variables from G:

$$\sum_{product \in FH'} P_r = \sum_{product \in FH'} product \cdot \Big(\prod_{g_j \in G - \{g_i \mid g_i \in product\}} \bar{g}_j \Big) \tag{3.12}$$

Theorem 3.3. The function *IOF* created as above describes all minimal solutions to the flattened decision function *FH*.

Proof. Let us assume first that *FH'=FH*. Since each *product* in *FH'* is distinct and is not included in any other, each corresponding to it product P_r satisfies the function *FH* and the function *IOF*. Each *product* of variables that is included in a *product* from *FH'* would create *product P sub r* that would satisfy no *product* in *FH*. Therefore $FH' \supseteq IOF$, where *IOF* is created as in (3.12). For each product that includes a $product_1$ from *FH'* the *FH* can be satisfied or not but the solution created for the respective *product* in *IOF* would be more expensive than one corresponding to the $product_1$. On the same base one can exclude from *FH* those products that are included in other products, creating function *FH'*. Q.E.D.

Example 3.2. For function H = (a ⊕ b) (a ⊕ c) the function FH is a ⊕ ac ⊕ ab ⊕ bc. After removal of ac ⊂ a and of ab ⊂ a, FH'.= a ⊕ bc. Hence according to formula (3.12): $IOF = a\bar{b}\bar{c} + \bar{a}bc$.

3.3. Direct Tree Search Method Model.

The tree search method consist of the systematic selection of all possible subsets of a set of all positive literals and creating a branch for each subset. Most of the branches are not completely extended since the cost-based cut-off principle of the branch-and-bound programming is also used. This is the most efficient method from those proposed here.

Example 3.3. Application of the Direct Tree Search method for the *function from* Example 2.2 is presented in Figure 3.1. Branching is done with respect to the variables from the *"best terms"* (explained below). The best terms in each node of the tree are underlined. Variables g_j are the *search operators.*

The function H for $node_k$ is denoted by $H(node_k)$. Whenever a positive literal g_j is selected, it is replaced with logic 1 in all respective terms of $H(node_k)$. If two 1's occur in a term, they are anihilated. Thus a new $node_{k+1}$ is created to which an arrow (operator) g_j leads from the previous $node_k$. The function H for $node_{k+1}$ is, after simplification,
denoted by $H(node_{k+1})$. The goal of the search is to select such positive literals that ones will appear in all Exor terms (all terms become 1-terms). In such case the solution is a product of two product terms:

- the product of all the positive literals selected in a branch leading to the node with all terms being 1-terms,

and

- the product of the negations of all the remaining variables from the set G.

Let us observe that the expression $H(node_k)$ is the function H simplified by substitution of ones for all the literals along the branch leading to $node_k$.

The literals for branching are selected in each node as follows. One Exor term of the expression is selected according to some heuristic criteria, similar to those from [22]. For simplification, in this example, a term is selected that maximizes the *total usefulness* of its literals.

$$\sum_{\substack{g_j \in t_i \\ t_i \in H(node)}} usefulness(g_j) \tag{3.13}$$

Definition 3.1. The *usefulness of a variable in* $node_k$ is the number of Exor terms in the function $H(node_k)$ that are converted from 0-terms to 1-terms minus the number of terms converted from 1-terms to 0-terms by selecting this variable.

Definition 3.2. The *best term* of $node_k$, denoted by t_{max} is a term of $H(node_k)$ which maximizes function (3.13).

*node*1. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)\underline{(g_0 \oplus g_2 \oplus g_4 \oplus g_5)}(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_0

*node*2. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus g_2 \oplus g_4 \oplus g_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)\underline{(g_3 \oplus g_7 \oplus g_8)}$

g_8

*node*3. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(1 \oplus g_2 \oplus g_4 \oplus g_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_3 \oplus g_7)$.
cost = 2, SOLUTION = $g_0 g_8 \overline{g_1}\, \overline{g_2}\, \overline{g_3}\, \overline{g_4}\, \overline{g_5}\, \overline{g_6}\, \overline{g_7}$.

g_2

*node*4. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(1 \oplus g_0 \oplus g_4 \oplus g_5)\underline{(g_0 \oplus g_5 \oplus g_6 \oplus g_7)}(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_6

*node*5. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(1 \oplus g_0 \oplus g_4 \oplus g_5)(1 \oplus g_0 \oplus g_5 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$.
cost = 2, SOLUTION = $g_2 g_6 \overline{g_0}\, \overline{g_1}\, \overline{g_3}\, \overline{g_4}\, \overline{g_5}\, \overline{g_7}\, \overline{g_8}$.

g_5

*node*6. $\underline{(g_0 \oplus g_1 \oplus g_2 \oplus g_3)}(1 \oplus g_0 \oplus g_2 \oplus g_4)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_1

*node*7. $(1 \oplus g_0 \oplus g_2 \oplus g_3)(1 \oplus g_0 \oplus g_2 \oplus g_4)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$.
cost = 2, SOLUTION = $g_1 g_5 \overline{g_0}\, \overline{g_2}\, \overline{g_3}\, \overline{g_4}\, \overline{g_6}\, \overline{g_7}\, \overline{g_8}$.

g_4

*node*8. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)\underline{(1 \oplus g_0 \oplus g_2 \oplus g_5)}(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$

g_0

*node*9. $(1 \oplus g_1 \oplus g_2 \oplus g_3)(g_2 \oplus g_5)(1 \oplus g_5 \oplus g_6 \oplus g_7)(g_3 \oplus g_7 \oplus g_8)$.
cost = 2, cut-off.

g_2

*node*10. $(1 \oplus g_0 \oplus g_1 \oplus g_3)(g_0 \oplus g_5)(g_0 \oplus g_5 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$.
cost = 2, cut-off.

g_5

*node*11. $(g_0 \oplus g_1 \oplus g_2 \oplus g_3)(g_0 \oplus g_2)(1 \oplus g_0 \oplus g_6 \oplus g_7)(1 \oplus g_0 \oplus g_3 \oplus g_7 \oplus g_8)$.
cost = 2, cut-off.

Figure 3.1.

After the selection of the best term t_{max}, the variables g_j in the term t_{max} are sorted according to the non-increasing values of function *usefulness*(g_j). Next the *search operators* are created, based on those variables. If the *best term* is a 0-term the *search operators* are simply the variables g_j. If the *best term* is a 1-term than the operators are:

- all the positive literals - the g_j variables from the 1-term,

and

- the product of the negative literals corresponding to those variables.

This branching method results from the fact that the function H can be satisfied only when each of its Exor terms is satisfied, and the above method describes all substitutions that satisfy the selected term.

The tree from Figure 3.1 assumes the *depth-first searching method*. In node 1 term $(g_0 \oplus g_2 \oplus g_4 \oplus g_5)$ is selected as the base for branching. Node 2 is created by substituting $g_0 = 1$ and simplifying $1 \oplus 1 = 0$ in the last term. Term $(g_3 \oplus g_7 \oplus g_8)$ is selected in node 2. Now, let us observe that in node 2 the branching is done only for g_8. This is due to the fact that g_8 is the only variable that occurs in the best term and does not occur in the other terms of $H(node_2)$. Other variables from this best term (g_3, g_7) occur also in other terms and hence they would make other terms unsatisfied after application of the $a \oplus a = 0$ cancellations. They will, therefore, not lead to a solution node of the same (or better) cost as one to which g_8 leads. Application of the operator g_8 leads to a node 3 with a solution $g_0 g_8 \overline{g_1}\, \overline{g_2}\, \overline{g_3}\, \overline{g_4}\, \overline{g_5}\, \overline{g_6}\, \overline{g_7}$ of cost 2. The value of 2 becomes the temporary optimal solution cost that will be used for cut-off. Backtrack to node 1 occurs and literal g_2 is selected. Node 4 is created by substitution $g_2 = 1$. Term $(g_0 \oplus g_5 \oplus g_6 \oplus g_7)$ is selected as the best term in node 4. The variable g_6 is the only one that occurs in the best term and does not occur in any other terms of $H(node_4)$. Selection of variable g_6 as an operator leads to node 5. There is no need to do the branching for the other variables from that best term, as that will not create the solution with the same or better cost than the solution at node 5. Node 5 corresponds to a solution $g_2 g_6 \overline{g_0}\, \overline{g_1}\, \overline{g_3}\, \overline{g_4}\, \overline{g_5}\, \overline{g_7}\, \overline{g_8}$ of cost 2. Now, a backtrack to node 1 is done and the branching is continued from it. Similarly the entire tree from Fig. 3.1 is created.

Theorem 3.4. The Direct Tree Search method creates all minimal ESOP solutions.

Proof. It results from the method of search organization that the entire solution space of product groups is searched: in each term one has to select a positive literal out of its variables, or no positive literal is selected for this term (which means selecting a product of all its variables with negations). The selection of best terms and sorting of literals inside the terms have no meaning other than being a speeding-up heuristics which enables to find a good solution quickly, thus improving cut-offs in the next phases. It was also explained in the Example 3.3 why branching in some nodes (such as $node_2$) can be limited. The above explained type of cut-off principle, as well as the standard cost-based cut-off will not prevent a generation of all exact solutions. Q.E.D.

The search is organized in such a way that the cut-off is expected as soon as possible. The following explanation can be given:

1. For each 1-term one has to select as an operator either a product of all negated literals (no positive literal selected for this term) or one positive literal. A generation of a tree based on the best terms is more efficient than the others branching methods.
2. For each 0-term one has to select as an operator only one positive literal to convert it to a 1-term. A 0-term with the literals that make most improvement (cover as many true minterms and as few false minterms as possible) is selected as the best term. This is a *heuristic rule* for the best branching.
3. If a term can be find so that by selecting one of its positive literals, all terms are converted to 1-terms - the term is selected and branching is done only for this one positive literal. This terminates the search for this branch. This is a strict rule.

The advantage of this approach is a speed, assuming a sufficient memory.

Another tree searching strategy for the best product of variables is the so-called "tree of all subsets of a set" (the set of all positive literals in this search variant) [22]. The are many methods to search such a tree. One method assumes a lexicographical order of variables. For each node the branching is done for all variables that are higher in the order than the variable leading to this node. In another variant of this method, the variables are sorted in each node after creating it, according to heuristic evaluation of their local quality. In yet another variant sorting is also done after backtracking to a node. In all these algorithms the cut-off principle is used to prune nodes of the solution tree which costs exceed the stored cost of the actually found minimum solution. The advantage of this approach is memory efficiency.

4. CONCLUSION

A new concept of a decision function for even/odd covering problems (particularly ESOP minimization), together with the algorithm to minimize such functions, have been introduced. Such an approach permits exact minimization of ESOPs and is to our knowledge the first attempt to create an exact minimization algorithm for an arbitrary number of input variables.

The introduced here method is very time and memory consuming. Recently, the new theory and the exact algorithm have been developed, which improves time and memory constrains by essentially decreasing the size of the searching tree [26,27]. This new algorithm has been also generalized for multi-valued input logic and multi output functions [27]. Variants of these algorithms to find quasi-minimum solutions are presented in [27]. Furthermore, several systolic architectures to solve this problem have also been proposed by the authors [14,15].

Since the Reed Muller Forms and the Generalized Reed Muller Forms are special cases of ESOPs, with additional restriction imposed on product groups - all the above methods can be easily generalized for RMs and GRMs of incompletely specified Boolean functions. We are not aware of any exact algorithms to create Reed-Muller forms for incompletely specified functions. For the GRM forms of the completely specified functions this problem has not been solved satisfactorily until now, even approximate algorithms have not been proposed. For the incompletely specified functions it has not been solved at all. In addition, thanks to some additional search reductions (more powerful than by only restricting polarities), which are particular to the polarity constraints of GRM forms, the search is essentially reduced and even larger functions can be minimized [27].

5. ACKNOWLEDGMENT

The authors would like to thank Martin Helliwell for the development of the initial concept of function H for minterms.

6. LITERATURE

[1] Besslich, Ph.W., "Efficient Computer Method for EXOR Logic Design", *Proc. IEE,* Vol. 130, Part E, CDT, No. 6., pp. 203-206, 1983. [2] Brayton, R.K., Camposano, R., De Micheli, G., Otten, R.H.J.M., and J. Van Eijndhoven, *"The Yorktown Silicon Compiler System"*, Chapter 7 in Gajski, D., (ed), *Silicon Compilation,* 1987. [3] Brayton, R., Rudell, R., Sangiovanni-Vincentelli, A., Wang, A., "MIS: Multiple-Level Logic Optimization System", *IEEE Transactions on Computer-Aided Design of Integrated Circuits* and Systems, Nov. 1987, pp. 1062-1081. "Inconsistent canonical forms of switching functions", *IRE Trans. Electron. Comput.,* Vol. EC-11, p. 284, April 1962. [4] Csanky, L., "On the Generalized Reed-Muller Canonical Form of Boolean Functions", *M. S. Thesis, University of California, Berkeley,* December 4, 1972, California 94720. [5] Daasch, R., *Private Communication.* [6] Davio, M., Deschamps, J.P., and A. Thayse, *"Discrete and Switching Functions"*, McGraw-Hill Book Co., Inc., New York, 1978. [7] Fleisher, H., Tavel, M, and J. Yeager, "Exclusive-OR representations of Boolean functions", *IBM J. Res. Develop.,* Vol. 27, pp. 412-416, July 1983. [8] Fleisher, H., Tavel, M., and J. Yeager, "A Computer Algorithm for Minimizing Reed-Muller Canonical Forms", *IEEE Trans. on Computers,* Vol. C-36, No. 2, February 1987. [9] Garey, M. and Johnson D., "Computers and Intractability: a Guide to the Theory of NP-Completeness", *Freeman, San Francisco, CA, 1979.* [10] Green, D., *"Modern Logic Design"*, Electronic Systems Engineering Series, 1986. [11] Handschy, M.A., Johnson, K.M., Cathey, W.T., and L. A. Pagano-Stauffer, "Polarization-based optical parallel logic gate utilizing ferroelectric liquid crystals", *Optics Letters,* Vol. 12, No. 8, August 1987. [12] Helliwell, M., and M.A. Perkowski, "A Fast Algorithm to Minimize Multi-Output Mixed-Polarity Generalized Reed-Muller Forms", *Proc. 25-th ACM/IEEE Design Automation Conference,* paper 28.2, pp. 427-432, June 12- June 15, 1988. [13] Helliwell, M., *Private Information.* [14] Phuong Minh Ho, M. Perkowski, "Systolic Architecture for Solving Combinatorial Problems of Logic Design", *Proc. International Symposium on Circuits and Systems, ISCAS'89,* May 9-11, 1989. [15] Phuong Minh Ho, M. Perkowski, "Performance Analysis of a Parallel Architecture for Solving Combinatorial Problems", submitted to *17th Intern. Symp. on Computer Architecture, Seattle, WA, May 28-31, 1990.* [16] Hurst, S.I., *"Logical processing of digital signals"*, Edward Arnold, London: Crane-Russak, N.Y., 1978. [17] Kodandapani, K.L., and R.V. Setlur, "A note on minimum Reed-Muller canonic forms of switching functions", *IEEE Trans. Comp.,* Vol. C-26, pp. 310-313, 1977. [18] Kohavi Z., "Switching and Finite Automata Theory.", (2nd edition), McGraw-Hill, New York, 1978. [19] Mukhophadhyay, A., and G. Schmitz, "Minimization of exclusive-OR and logical equivalence switching circuits", *IEEE Trans. Comp.,* Vol. C-19, No. 2. , pp. 132-140, February 1970. [20] Muller, D.E., "Application of Boolean algebra to switching circuit design and to error detection", *IRE Trans. Electron. Comp.,* Vol EC-3, pp. 6-12, September 1954. [21] Papakonstantinou, G., "Minimization of modulo-2 sum of products", *IEEE Trans. on Computers.,* Vol. C-28, pp. 163-167, February 1979. [22] Perkowski, M.A., Liu, J., and J.E. Brown, "Quick Software Prototyping: CAD Design of Digital CAD Algorithms", *In G. Zobrist (ed) "Progress in Computer Aided VLSI Design"*, Ablex Publishing Corp., 1989. [23] Perkowski, M.A., and P. Wu, "KUAI-EXACT: A New Approach for Multi-Valued Logic Minimization in VLSI Synthesis", *Proc. 1989 ISCAS - International* Symposium on Circuits and Systems, May 9-11, 1989. [24] Perkowski, M.A., Helliwell, M., and P. Wu, "Minimization of Multiple-Valued Input, Multi-Output Generalized Reed Muller Forms", *Proc. International Symposium on Multi-Valued Logic,* Guangzhou, May 29-31 1989, People's Republic of China. [25] Perkowski, M.A. Dysko, P., and B.J. Falkowski, "Two Learning Methods for a Tree-Search Combinatorial Optimizer", *Proceedings of* IEEE International Phoenix Conference on Computers and Communication, Scottsdale, Arizona, March 1990. [26] Perkowski, M.A., and M. Chrzanowska-Jeske, "Tree Search Algorithms to Find Exact ESOP Forms", *PSU EE. Dept. Report, 1990.* [27] Perkowski, M.A., and M. Chrzanowska-Jeske, "Approximate and Exact Tree Search Algorithms for Minimization of Binary and Multiple-Valued Input ESOPs, RMs and GRMs for Strongly Unspecified Boolean Functions", *PSU EE. Dept. Report, 1990.* [28] Pitty, E.B., Salmon, J.V.: "Input Irredundancy of Mixed-Polarity Reed-Muller Equations", *Electronics Letters,* March 3, 1988, Vol. 24, No. 5., pp. 258-260. [29] Reddy, B.R.K., and A.L. Pai, Reed - Muller Transform Image Coding, *Computer Vision, Graphics, and Image Processing,* Vol. 42, pp. 48 - 61 (1988). [30] Reed, I.S., "A class of multiple-error-correcting codes and their decoding scheme", *IRE Trans. Inf.Th.,* Vol. PGIT-4, pp. 38-49, 1954. [31] Sasao, T., Besslich, P.: "On the Complexity of MOD-2 Sum PLA", *Institute of Electronics and Communication Engineers of Japan,* FTS86-17, pages 1-8, Nov. 17, 1986. [32] Yu, Francis T.S., Suganda, J., and D.A. Gregory, *"Real-time liquid crystal TV XOR- and XNOR-gate* binary image subtraction technique", *Applied Optics, Vol. 26, No. 14/15 July 1987.*