Three-Level Logic Minimization Based on Function Regularities

Anna Bernasconi, Valentina Ciriani, *Student Member, IEEE*, Fabrizio Luccio, *Fellow, IEEE*, and Linda Pagli

***Abstract*—We exploit the "regularity" of Boolean functions with the purpose of decreasing the time for constructing minimal three-level expressions, in the sum of pseudoproducts (SPP) form recently developed. The regularity of a Boolean function f of n variables can be expressed by an *autosymmetry degree* k (with $0 \leq k \leq n$). $k = 0$ means no regularity, that is we are not able to provide any advantage over standard synthesis. For $k \geq 1$ the function f is said to be *autosymmetric*, and a new function f_k depending on $n - k$ variables only, called the *restriction* of f, is identified in time polynomial in the number of points of f. The relation between f and f_k is discussed in depth to show how a minimal SPP form for f can be build in linear time from a minimal SPP form for f_k. The concept of autosymmetry is then extended to functions with don't care conditions, and the SPP minimization technique is duly extended to such functions. A large set of experimental results is presented, showing that 61% of the outputs for the functions in the classical ESPRESSO benchmark suite are autosymmetric. The minimization time for such functions is critically reduced, and cases otherwise intractable are solved. The quality of the corresponding circuits, measured with some well established cost functions, is also improved. Finally, we discuss the role and meaning of autosymmetric functions, and why a great amount of functions of practical interest fall in this class.**

***Index Terms*—Circuit optimization, circuit synthesis, logic design.**

I. INTRODUCTION

THE DESIGN flow of logic circuits always includes a phase of Boolean function synthesis. In this phase, reduced, and possibly minimal, algebraic forms are determined for the functions, in order to reduce the final size of the corresponding circuits.

The standard synthesis is performed with sum of products (SOP) minimization procedures, leading to two level circuits. More-than-two level minimization is much harder, but the size of the circuits can significantly decrease. In many cases three-level logic is a good tradeoff among circuit speed, circuit size, and the time needed for the minimization procedure. In any case algorithms for exact minimization have exponential complexity, hence the time to attain minimal forms becomes huge for increasing size of the input.

Two level minimization is well developed, see any of the classical references, e.g., [16], or the more recent [26], [6], [21]. Techniques for three level minimization have generally been given for specific algebraic expressions. A relevant three-level minimization algorithm was given in [20] for networks of the form $f = g_1 \circ g_2$ where g_1 and g_2 are SOP forms and $\circ$ denotes a *binary* operation. The case of $\circ =$ AND (called AND-OR-AND) is described in [11]. The case of $\circ =$ EXOR (called *EX-SOP*) has been widely studied (see for details [24], [22], [12], [7]–[9], [13]). For example, $(x_0 \cdot x_1 + x_0 \cdot \overline{x}_2) \oplus (x_2 \cdot x_3 + x_1 \cdot x_4)$ is an EX-SOP form. Indeed many practical functions (e.g., arithmetic functions) have a more concise expression if we allow the use of EXORs other than classical AND and OR gates. Another well known advantage of EXOR gates is their excellent testability. An EX-SOP three-level network is one of the simplest EXOR of sum of product architecture, since it contains only a single two-input EXOR gate. An algorithm for exact minimization of EX-SOP networks is described in [7], limited to functions with up to five variables. Some interesting heuristics are described in [9], [13]. An estimation metric which measures whether an input function is suitable for EX-SOP minimization is also developed in [13].

A different three-level form called sum of pseudoproducts (or SPP) was introduced in [19]. SPP expressions can be seen as a direct generalization of SOP expressions using EXOR gates. An SPP form consists of the OR of *pseudoproducts*, where a pseudoproduct is the AND of EXOR factors. For example, $(x_0 \oplus \overline{x}_1) \cdot x_2 + (x_0 \oplus x_2 \oplus x_3) \cdot (x_2 \oplus \overline{x}_4) + x_1 \cdot x_4$ is an SPP form. Experimental results show that the average size of SPP forms is approximately half the size of the corresponding SOP, and SPP forms are also smaller than EX-SOP [3]. As a limit case each EXOR factor reduces to a single literal in SPP, and the SOP and SPP forms coincide.

In this work, we focus to SPP minimization. Initially this can be seen as a generalization of SOP minimization, and in fact an extension of the Quine-McCluskey algorithm was given in [19] for SPP. In particular the pseudoproducts to be considered can be limited to the subclass of *prime* pseudoproducts, that play the same role of prime implicants in SOP. The algorithm for SPP, however, was more cumbersome than the former, thus failing in practice in minimizing very large functions. A deeper understanding of the problem, together with the use of ad-hoc data structures, has allowed to widely extend the set of functions practically tractable [3]. Still a number of standard benchmark functions can be hardly handled with this technique.

The aim of this paper is to exploit the "regularity" of any given Boolean function, in order to decrease the time needed for its logical synthesis. Function regularities have been exploited in different contests [2], [17], [18].

Our main results are: 1) the regularity of a Boolean function f of n variables can be expressed by an *autosymmetry degree* k (with $0 \leq k \leq n$), which is computed in time polynomial in

Manuscript received June 14, 2002; revised October 7, 2003. This paper was presented in part at the ACM-IEEE 39th DAC (2002) under the title "Fast Three-Level Logic Minimization Based on Autosymmetry." This paper was recommended by Guest Editor D. T. Blaauw.

The authors are with Dipartimento di Informatica, Università di Pisa, 56127 Pisa, Italy (e-mail: annab@di.unipi.it; ciriani@di.unipi.it; luccio@di.unipi.it; pagli@di.unipi.it).

Digital Object Identifier 10.1109/TCAD.2003.814950

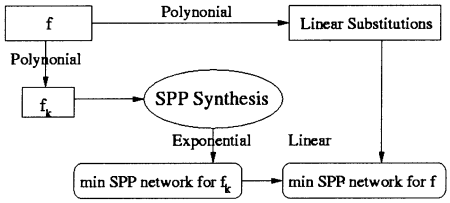

Fig. 1. SPP synthesis of an autosymmetric function f through the SPP synthesis of its restriction f_k.

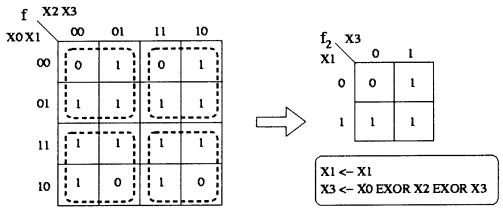

Fig. 2. A 2-autosymmetric function f, in a Karnaugh map (on the left) of four variables. The corresponding reduced function f_2, which depends on the variables x_1 and x_3, and the linear substitutions are on the right.

the number of points of f; 2) $k = 0$ means no regularity, that is we are not able to provide any advantage over standard synthesis; for $k \geq 1$ the function f is said to be *autosymmetric*, and a new function f_k, called the *restriction* of f, is identified in polynomial time. In a sense f_k is "equivalent" to, but smaller than f, and depends on $n - k$ variables only. The relation between f and f_k is discussed in depth, to show how a minimal SPP form for f can be build in linear time from a minimal SPP form for f_k; 3) the concept of autosymmetry is extended to functions with don't care conditions, and SPP minimization technique is duly extended to such functions; 4) a large set of experimental results is presented, showing that 61% of the outputs for the functions in the classical ESPRESSO benchmark suite are autosymmetric: the SPP minimization time for them is critically reduced, and cases otherwise intractable are solved. Indeed, although autosymmetric functions form a subset of all possible Boolean functions, a great amount of standard functions of practical interest fall in this class. In the last section, we speculate on the possible causes of this fact to substantiate the interest of our work. Note that an autosymmetric function f depends in general on all the n input variables, however we shall be able to study f in a $n - k$ dimensional space; i.e., f is in general non degenerated, whereas all degenerated functions are autosymmetric.

In Section II, we show with an example, the main idea of our minimization method. In Section III, we recall the basic definitions and results of SPP theory, and present a companion algebraic formulation later exploited for testing autosymmetry. In Section IV, we discuss the properties of autosymmetric functions, and how the problem of determining their minimal SPP forms can be studied on a reduced number of variables. In Section V, we show how autosymmetry can be tested in polynomial time, and derive a new minimization algorithm that includes such a test in the initial phase. In Section VI, we extend the notion of autosymmetry to functions with don't care set, showing the theoretical and practical consequences of such an extension. In Section VII, we present a large set of experimental results which validate the proposed approach, also proving that the number of benchmarks practically tractable is significantly increased. A discussion on the role of autosymmetry, and why it deserves great attention, is finally developed in Section VIII.

II. EXAMPLE

Fig. 1 shows the minimization strategy for an autosymmetric function f. First we detect, in time polynomial in the number of points of f, the autosymmetry degree k (as explained in Section IV), and if $k > 0$ we derive its restriction f_k and linear substitutions (Section IV-B). Second, we minimize f_k in SPP framework; this task usually requires time exponential in the number of variables of f_k. Finally, we derive the minimal SPP network for f from the one of f_k and the linear substitutions, as explain in Section V. This final task can be performed in linear time.

To better explain our minimization method, let us apply it to the function f of Fig. 2. A minimal SOP form for f is $x_1 + \overline{x}_0\overline{x}_2x_3 + \overline{x}_0x_2\overline{x}_3 + x_0\overline{x}_2\overline{x}_3 + x_0x_2x_3$, while its minimal SPP form is $x_1 + (x_0 \oplus x_2 \oplus x_3)$. First, we can observe that the minimal SPP form is much more compact than the SOP expression. However, SPP synthesis is more expensive, than the SOP minimization, in computational time. To overcome this problem, we can exploit the regularities, if any, of the function. Consider the Karnaugh map on the left side of Fig. 2. The four subspaces of the points within the dotted lines present a sort of symmetry: they are rotations of the Karnaugh map on the right side, which represent the function called f_2.

Starting from the two input variables function f_2, we could derive many four input variables functions by combinations of different rotations of f_2. We, thus, need an additional information to reconstruct the starting function f. This information is provided by the linear substitutions given in Fig. 2.

The function f could then be studied through the smaller function f_2. Given an SPP minimal form for f_2 and the linear substitutions, we can finally generate a minimal SPP form for f.

Following our example, a minimal SPP form for f_2 is $x_1 + x_3$. The substitutions $x_1 \leftarrow x_1$, $x_3 \leftarrow x_0 \oplus x_2 \oplus x_3$ give a minimal SPP form for f: $x_1 + (x_0 \oplus x_2 \oplus x_3)$. Obviously, the minimization time for f_2 is much smaller. Fortunately, as we shall show in this paper, deriving f_2 and the linear substitutions is an easy task.

III. UNDERLYING THEORY

SPP theory was posed in [19] and extended in [3]. We report here some basic definitions and properties together with new results needed to developing the theory of autosymmetry. We work in a Boolean space $\{0,1\}^n$ described by n variables $x_0, x_1, \ldots, x_{n-1}$, where each point is represented as a binary vector of n bits. A set of k points can be arranged in a $k \times n$ matrix whose rows correspond to the points and whose columns correspond to the variables. Fig. 3 represents a set of eight points in a space of six variables. A Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$ can be specified with an algebraic expression where the variables are connected through Boolean operators, or as the set of points for which $f = 1$. $|f|$ denotes the number of such points.

Let $\boldsymbol{u}$ be a (Boolean) vector; $\overline{\boldsymbol{u}}$ be the element-wise complementation of $\boldsymbol{u}$; and $\widehat{\boldsymbol{u}}$ denote $\boldsymbol{u}$ or $\overline{\boldsymbol{u}}$. The *constant* vectors $\mathbf{0}$ and $\mathbf{1}$ are made up of all 0s or all 1s, respectively. Vector $\boldsymbol{uv}$ is the

	$\mathbf{c_0}$	$\mathbf{c_1}$	$\mathbf{c_2}$	$\mathbf{c_3}$	$\mathbf{c_4}$	$\mathbf{c_5}$
$\mathbf{r_0}$	0	1	0	0	0	1
$\mathbf{r_1}$	0	1	0	0	1	0
$\mathbf{r_2}$	0	1	1	1	0	1
$\mathbf{r_3}$	0	1	1	1	1	0
$\mathbf{r_4}$	1	1	0	1	0	0
$\mathbf{r_5}$	1	1	0	1	1	1
$\mathbf{r_6}$	1	1	1	0	0	0
$\mathbf{r_7}$	1	1	1	0	1	1

Fig. 3. A canonical matrix representing a pseudocube P in $\{0,1\}^6$. The canonical columns are $\mathbf{c_0}$, $\mathbf{c_2}$, and $\mathbf{c_4}$.

concatenation of $\boldsymbol{u}$ and $\boldsymbol{v}$. A vector $\boldsymbol{u}$ of 2^m elements, $m \geq 0$, is *normal* if $m = 0$, or $m > 0$ and $\boldsymbol{u} = \boldsymbol{v}\widehat{\boldsymbol{v}}$ with $\boldsymbol{v}$ (hence $\widehat{\boldsymbol{v}}$) normal. For instance, $\boldsymbol{u} = 0110$ is normal since $\boldsymbol{u} = \boldsymbol{v}\overline{\boldsymbol{v}}$ with $\boldsymbol{v} = 01$, which is in turn normal. All the columns in the matrix of Fig. 3 are normal vectors.

A matrix M with 2^m rows is *normal* if all its rows are different and all its columns are normal. A normal matrix is *canonical* if its rows, interpreted as binary numbers, are arranged in increasing order (the matrix of Fig. 3 is canonical). A normal vector is k-*canonical*, $0 \leq k < m$, if it is composed of an alternating sequence of groups of 2^k 0s and 2^k 1s. In Fig. 3, $\boldsymbol{c}_0$ is 2-canonical, $\boldsymbol{c}_2$ is 1-canonical, and $\boldsymbol{c}_4$ is 0-canonical. A canonical matrix M contains m columns $\boldsymbol{c}_{i_0}, \ldots, \boldsymbol{c}_{i_{m-1}}$ of increasing indexes, called the *canonical columns* of M, such that $\boldsymbol{c}_{i_j}$ is the lowest index column that is $(m-j-1)$-canonical for $0 \leq j \leq m-1$. The other columns are the *noncanonical* ones. Note that different $(m-j-1)$-canonical columns can coexist in a canonical matrix, but only the one of lowest index is called canonical. If M represents a set of points in a Boolean space, with column $\boldsymbol{c}_i$ corresponding to variable x_i, canonical and noncanonical columns correspond to *canonical* and *noncanonical variables*, respectively. For example, consider the matrix in Fig. 3. The canonical columns are $\mathbf{c}_0$, which is 2-canonical, $\mathbf{c}_2$, which is 1-canonical, and $\mathbf{c}_4$, which is 0-canonical. The corresponding canonical variables are x_0, x_2, and x_4, respectively. Observe that these variables assume all the possible combinations of values. We have

Definition 1 (From [19]): A pseudocube of degree m is a set of 2^m points whose matrix is canonical up to a row permutation. The matrix of Fig. 3 represents a pseudocube of 2^3 points in $\{0,1\}^6$. The function with value 1 in the points of a pseudocube P (i.e., the characteristic function of P) is called *pseudoproduct*, and can be expressed as a product of EXOR factors in several different forms, one of which is called the *canonical expression* (briefly *CEX*) of P. For the pseudocube P of Fig. 3 we have:

$$\mathrm{CEX}(P) = x_1 \cdot (x_0 \oplus x_2 \oplus \overline{x}_3) \cdot (x_0 \oplus x_4 \oplus x_5). \quad (1)$$

Refer to [19] for the nontrivial rule for generating $\mathrm{CEX}(P)$. Intuitively, in Fig. 3 the column $\mathbf{c}_3$ is the EXOR between columns $\mathbf{c}_0$ and $\mathbf{c}_2$, therefore, $(x_0 \oplus x_2 \oplus \overline{x}_3)$ is true. Analogously, $\mathbf{c}_5$ is always different from the EXOR between columns $\mathbf{c}_0$ and $\mathbf{c}_4$, therefore, $(x_0 \oplus x_4 \oplus x_5)$ is true.

We now simply recall that each EXOR factor of the expression contains exactly one noncanonical variable in directed or complemented form, namely the one with greatest index (x_1, x_3, and x_5 in the example), and each noncanonical variable appears in exactly one EXOR factor; all the other variables in the expression are canonical (x_0, x_2, and x_4 in the example) and appear in direct form; and some canonical variables may not appear in the expression. Note that the minimal SOP form for the above function (1) is

$$\begin{aligned}&\overline{x}_0x_1\overline{x}_2\overline{x}_3\overline{x}_4x_5 + \overline{x}_0x_1\overline{x}_2\overline{x}_3x_4\overline{x}_5 + \overline{x}_0x_1x_2x_3\overline{x}_4x_5\\ &+ \overline{x}_0x_1x_2x_3x_4\overline{x}_5 + x_0x_1\overline{x}_2x_3\overline{x}_4\overline{x}_5 + x_0x_1\overline{x}_2x_3x_4x_5\\ &+ x_0x_1x_2\overline{x}_3\overline{x}_4\overline{x}_5 + x_0x_1x_2\overline{x}_3x_4x_5\end{aligned}$$

much larger than $\mathrm{CEX}(P)$.

A cube in $\{0,1\}^n$ is a special case of pseudocube where the noncanonical columns are constant. In this case each EXOR factor in $\mathrm{CEX}(P)$ reduces to a single noncanonical variable, the canonical variables do not appear, and the whole expression reduces to the well known *product* expression, e.g., used for implicants in SOP forms.

A general property of the algebraic representation of pseudocubes is given in the following theorem.

Theorem 1: In a Boolean space $\{0,1\}^n$: a) the EXOR factor of any subset of variables (directed or complemented) represents a pseudocube of degree $n-1$; b) the product of $k \leq n$ arbitrary EXOR factors represents either an empty set or a pseudocube of degree $\geq n-k$.

Point a) of the theorem can be easily proved by induction on the number of variables in the EXOR factor. Point b) then follows from a theorem of [19] which states that the intersection of two pseudocubes of degrees p, r is either empty, or is a pseudocube of degree $\geq p + r - n$.

For the example of Fig. 3, the EXOR factors x_1, $x_0 \oplus x_2 \oplus \overline{x}_3$, and $x_0 \oplus x_4 \oplus x_5$ in (1) represent three pseudocubes of 2^5 points each, and their product represents the given pseudocube P of 2^3 points. Note now that an equality of the form $\mathrm{EXOR}_1 \cdot \mathrm{EXOR}_2 \cdot \cdots \cdot \mathrm{EXOR}_k = 1$ satisfied by all the points of a pseudocube can be equivalently written as a system of k linear equations: $\mathrm{EXOR}_i = 1$, $1 \leq i \leq k$ that is, an instance of a general linear system $A\boldsymbol{x} = \boldsymbol{b}$, with $\boldsymbol{x} = \{x_0, \ldots, x_{n-1}\}$, $\boldsymbol{b} \in \{0,1\}^k$, A is a $k \times n$ matrix of coefficients 0, 1, and the sum is substituted with EXOR. In fact, if EXOR_i contains a complemented variable $\overline{x}_j$, this variable can be changed to x_j and the new expression for EXOR_i be put equal to 0 instead of 1. As known [5], [1] the above system specifies an *affine subspace* of the linear space $\{0,1\}^n$.

Example 1: The CEX expression $\mathrm{CEX}(P) = x_1 \cdot (x_0 \oplus x_2 \oplus \overline{x}_3) \cdot (x_0 \oplus x_4 \oplus x_5)$ corresponds to the system:

$$\begin{cases} x_1 = 1 \\ x_0 \oplus x_2 \oplus \overline{x}_3 = 1 \\ x_0 \oplus x_4 \oplus x_5 = 1 \end{cases} \quad = \quad \begin{cases} x_1 = 1 \\ x_0 \oplus x_2 \oplus x_3 = 0 \\ x_0 \oplus x_4 \oplus x_5 = 1 \end{cases}$$

which represents the affine space composed of the points in the matrix of Fig. 3.

From the existence of $\mathrm{CEX}(P)$ for any pseudocube P, and from Theorem 1, we have Corollary 1.

Corollary 1: In a Boolean space $\{0,1\}^n$ there is a one-to-one correspondence between affine subspaces and pseudocubes.

This corollary allows to inherit all the properties of affine subspaces into pseudocube theory. In particular, a pseudocube containing the point (vector) $\mathbf{0}$ corresponds to a *linear* subspace. More details on this result can be found in [4].

The *structure* of a pseudocube P, denoted by $\mathrm{STR}(P)$, is $\mathrm{CEX}(P)$ without complementations [3]. For the pseudocube P of Fig. 3, we have: $\mathrm{STR}(P) = x_1 \cdot (x_0 \oplus x_2 \oplus x_3) \cdot (x_0 \oplus x_4 \oplus x_5)$. Let us now extend the symbol $\oplus$ to denote the element-wise EXOR between two vectors. Then $\boldsymbol{\alpha} \oplus \boldsymbol{\beta}$ is the vector obtained from $\boldsymbol{\beta}$ complementing in it the elements corresponding to the 1s of $\boldsymbol{\alpha}$. For a vector $\boldsymbol{\alpha} \in \{0,1\}^n$ and a subset $S \subseteq \{0,1\}^n$, let $\boldsymbol{\alpha} \oplus S = \{\boldsymbol{\alpha} \oplus \boldsymbol{s} \,|\, \boldsymbol{s} \in S\}$. For example, if $S = \{000,\ 010,\ 011,\ 001\}$ and $\boldsymbol{\alpha} = 100$, then $\boldsymbol{\alpha} \oplus S = \{100,\ 110,\ 111,\ 101\}$. We have Theorem 2.

Theorem 2: For any pseudocube $P \subseteq \{0,1\}^n$ and any vector $\boldsymbol{\alpha} \in \{0,1\}^n$, the subset $Q = \boldsymbol{\alpha} \oplus P$ is a pseudocube with $\mathrm{STR}(Q) = \mathrm{STR}(P)$.

Proof: Given a set S, the set $\boldsymbol{\alpha} \oplus S$ is the set of points obtained complementing in all the points of S the bits corresponding to the 1s of $\boldsymbol{\alpha}$. Furthermore, complementing a bit in all the points of a pseudocube P corresponds to substituting a literal with its complementation in the CEX expression. By definition of structure the thesis follows. ∎

Finally, recall that an arbitrary function can be expressed as an OR of pseudoproducts, giving rise to an SPP form [19]. For example, adding two rows (points) $\mathbf{r}_8 = 1\,1\,0\,1\,0\,1$ and $\mathbf{r}_9 = 1\,1\,0\,1\,1\,0$ to the matrix of Fig. 3 we have a new function f formed as the union of two partially overlapping pseudocubes: namely P (already studied), and Q associated to the rows $\mathbf{r}_4, \mathbf{r}_5, \mathbf{r}_8, \mathbf{r}_9$. Note that Q is in fact a cube, with $\mathrm{CEX}(Q) = x_0 x_1 \overline{x}_2 x_3$. In conclusion f can be expressed in SPP form as

$$\mathrm{CEX}(P) + \mathrm{CEX}(Q) = x_1(x_0 \oplus x_2 \oplus \overline{x}_3) \cdot (x_0 \oplus x_4 \oplus x_5) + x_0 x_1 \overline{x}_2 x_3.$$

The minimal SOP form for f contains 40 literals, while the SPP expression for $\mathrm{CEX}(P) + \mathrm{CEX}(Q)$ contains 11 literals. Passing from SOP to SPP, however, implies passing from a two-level to a three-level circuit. This fact has always to be taken into account and will not be further repeated.

IV. Autosymmetric Functions

A. Definitions and Characterization

The class of *autosymmetric functions* introduced in [19] seems to be particularly suitable for SPP minimization. The present work addresses these functions, for which we give an alternative definition.

Definition 2: A Boolean function f in $\{0,1\}^n$ is *closed under* $\boldsymbol{\alpha}$, with $\boldsymbol{\alpha} \in \{0,1\}^n$, if for each $\boldsymbol{w} \in \{0,1\}^n$, $\boldsymbol{w} \oplus \boldsymbol{\alpha} \in f$ if and only if $\boldsymbol{w} \in f$.

Example 2: The function $f = \{001, 010, 100, 111\}$ is closed under 011. Indeed $001 \oplus 011 = 010 \in f$, $010 \oplus 011 = 001 \in f$, $100 \oplus 011 = 111 \in f$ and $111 \oplus 011 = 100 \in f$.

Each function is obviously closed under the zero vector $\mathbf{0}$. Consider now the set of all the vectors $\boldsymbol{\beta}$ such that f is closed under $\boldsymbol{\beta}$. We have:

Proposition 1: The set $L_f = \{\boldsymbol{\beta}$: f is closed under $\boldsymbol{\beta}\}$ is a linear subspace of $\{0,1\}^n$.

Proof: We first observe that if a function f is closed under two different vectors $\boldsymbol{\alpha}_1, \boldsymbol{\alpha}_2 \in \{0,1\}^n$, it is also closed under $\boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2$. Indeed, $\boldsymbol{\omega} \oplus \boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2 \in f$ if and only if $\boldsymbol{\omega} \oplus \boldsymbol{\alpha}_1 \in f$ (since f is closed under $\boldsymbol{\alpha}_2$) if and only if $\boldsymbol{\omega} \in f$ (since f is closed under $\boldsymbol{\alpha}_1$). Combining in EXOR, in all possible ways, k linearly independent vectors $\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_k$, we form a subspace of 2^k vectors that is closed under $\oplus$, and contains the vector $\mathbf{0}$ generated as $\boldsymbol{\alpha}_i \oplus \boldsymbol{\alpha}_i$, see for example, [5]. Therefore, the set L_f is a linear subspace of $\{0,1\}^n$. ∎

The set L_f is called the *linear space* of f. Note that L_f has dimension $k = \log_2 |L_f|$. By Corollary 1, L_f is a pseudocube, and we shall refer to $\mathrm{CEX}(L_f)$ and $\mathrm{STR}(L_f)$.

Definition 3: A Boolean function f is *k-autosymmetric*, or equivalently f has *autosymmetry degree* k, $0 \leq k \leq n$, if its linear space L_f has dimension k. If $k \geq 1$, f will be simply called *autosymmetric*.

Example 3: The function f in Fig. 2 is 2-autosymmetric and its linear space L_f is {0000, 0011, 1001, 1010}. Indeed, f is closed under all the vectors in L_f.

Theorem 3: Let f be a k-autosymmetric function. There exist ℓ vectors $\boldsymbol{w}^1, \boldsymbol{w}^2, \ldots, \boldsymbol{w}^\ell \in f$, with $\ell = |f|/2^k$, such that

$$f = \bigcup_{i=1}^{\ell} (\boldsymbol{w}^i \oplus L_f) \tag{2}$$

and for each i, j, $i \neq j$, $(\boldsymbol{w}^i \oplus L_f) \cap (\boldsymbol{w}^j \oplus L_f) = \emptyset$.

Proof: Let $\boldsymbol{w}^1$ be any vector in f. By Definition 2, $\boldsymbol{w}^1 \oplus L_f \subseteq f$. Consider the set $f^1 = f \setminus (\boldsymbol{w}^1 \oplus L_f)$, where $\setminus$ denotes the set difference. If $f^1 = \emptyset$, then $f = \boldsymbol{w}^1 \oplus L_f$ and f is a pseudocube of degree k with $|f| = 2^k$. Otherwise, let $\boldsymbol{w}^2$ be any vector of f^1. Again by Definition 2, $\boldsymbol{w}^2 \oplus L_f \subseteq f$. Observe that $(\boldsymbol{w}^1 \oplus L_f) \cap (\boldsymbol{w}^2 \oplus L_f) = \emptyset$ (by contradiction: let $\boldsymbol{v} \in (\boldsymbol{w}^1 \oplus L_f) \cap (\boldsymbol{w}^2 \oplus L_f)$, then $\boldsymbol{v} = \boldsymbol{w}^1 \oplus \boldsymbol{\alpha} = \boldsymbol{w}^2 \oplus \boldsymbol{\beta}$, with $\boldsymbol{\alpha}, \boldsymbol{\beta} \in L_f$; then $\boldsymbol{w}^2 = \boldsymbol{w}^1 \oplus \boldsymbol{\alpha} \oplus \boldsymbol{\beta}$, that is $\boldsymbol{w}^2 \in (\boldsymbol{w}^1 \oplus L_f)$ which is a contradiction). Therefore, we have: $f = (\boldsymbol{w}^1 \oplus L_f) \cup (\boldsymbol{w}^2 \oplus L_f) \cup f \setminus ((\boldsymbol{w}^1 \oplus L_f) \cup (\boldsymbol{w}^2 \oplus L_f))$, and using the same argument on the set $f \setminus ((\boldsymbol{w}^1 \oplus L_f) \cup (\boldsymbol{w}^2 \oplus L_f))$ the theorem easily follows. ∎

From the proof above we see that the number of points of a k-autosymmetric function is a multiple of 2^k. Indeed, each affine subspace $\boldsymbol{w}^i \oplus L_f$ contains 2^k points. Recalling that L_f is a pseudocube, and by Theorems 2 and 3 we immediately have:

Corollary 2: A k-autosymmetric function f is a disjoint union of $|f|/2^k$ pseudocubes $\boldsymbol{w}^i \oplus L_f$ of degree k all having the same structure $\mathrm{STR}(L_f)$, and the same canonical variables of L_f.

This corollary has an immediate consequence. For any autosymmetric function f we can extend the definition of canonical and noncanonical variables from pseudocubes to the function f itself. Namely, the canonical (respectively: noncanonical) variables of L_f are designated as the *canonical* (respectively: *noncanonical*) *variables* of f.

Example 4: Consider the linear space L_f of function f in Fig. 2. We can arrange its vectors in the matrix:

	x_0	x_1	x_2	x_3
0	0	0	0	0
1	0	0	1	1
2	1	0	0	1
3	1	0	1	0

.

The canonical variables of L_f and f are x_0 and x_2.

An other important consequence of Theorem 3 is the following:

Corollary 3: The vectors $\boldsymbol{w}^1, \ldots, \boldsymbol{w}^\ell$ of Theorem 3 can be chosen as the points of f where all the canonical variables have value 0.

Proof: Note that the 2^k points of each pseudocube $\boldsymbol{w}^i \oplus L_f$ contain all the 2^k combinations of values of the canonical variables, hence in exactly one of these points all such values are 0. Therefore, in the proof of Theorem 3 we can choose the vectors $\boldsymbol{w}^i$ with all the canonical variables set to 0. ■

Note that the choice of $\boldsymbol{w}^1, \ldots, \boldsymbol{w}^\ell$ in Corollary 3 is not univocal.

Example 5: Consider the function

$$f = \{00001,\ 00100,\ 00110,\ 01000,\ 01010,\ 01101,\ 10001,\ 10011,\ 10100,\ 11000,\ 11101,\ 11111\}.$$

It can be easily verified that the linear space of f is $L_f = \{00000,\ 01100,\ 10101,\ 11001\}$, where each vector can be obtained as the EXOR of other two. f is then 2-autosymmetric. We have $\mathrm{STR}(L_f) = (x_0 \oplus x_1 \oplus x_2)x_3(x_0 \oplus x_4)$, where x_0, x_1, and x_2, x_3, x_4, are the canonical and noncanonical variables, respectively. From Corollary 3 we have $\boldsymbol{w}^1 = 00001$, $\boldsymbol{w}^2 = 00100$, $\boldsymbol{w}^3 = 00110$, hence: $f = (00001 \oplus L_f) \cup (00100 \oplus L_f) \cup (00110 \oplus L_f)$.

From the above properties of autosymmetric functions we observe that: 1) any function is at least 0-autosymmetric, since is closed under $\mathbf{0}$; 2) a function is $(n-1)$-autosymmetric if and only if it is a pseudocube of degree $n-1$; 3) a function is n-autosymmetric if and only if it is a constant; 4) pseudocubes of degree k are the only k-autosymmetric functions with only one term in the union of expression (2). We also have:

Theorem 4: The overall number of autosymmetric functions is $(2^n - 1)2^{2^n - 1}$.

Proof: We first count the number of functions that are at least k-autosymmetric, for a given k. Recall that a pseudocube is an affine space, and a k-autosymmetric function is a disjoint union of affine spaces over the same linear space L_f of dimension k (see Theorem 3). There are $\begin{bmatrix} n \\ k \end{bmatrix}$ ways of choosing a k-dimensional linear subspace (L_f) of $\{0,1\}^n$, where $\begin{bmatrix} n \\ k \end{bmatrix}$ denotes the Gaussian factor:

$$\begin{bmatrix} n \\ k \end{bmatrix} = \frac{\prod_{i=0}^{k-1}(2^{n-i} - 1)}{\prod_{i=0}^{k-1}(2^{k-i} - 1)}.$$

Once we have fixed the linear subspace L_f, we must choose a subset of different affine spaces over L_f, whose union defines a function that is at least k-autosymmetric. The different affine spaces over L_f are 2^{n-k} in number. Therefore, the overall number of functions that are at least k-autosymmetric is $\begin{bmatrix} n \\ k \end{bmatrix} 2^{2^{n-k}}$. For $k = 1$ the Gaussian factor is equal to $(2^n - 1)$, and the thesis follows. ■

B. The Restriction f_k of an Autosymmetric Function f

We now show how any k-autosymmetric function f can be studied through a simpler function f_k.

Definition 4: For a $(k \geq 1)$-autosymmetric function f, the *restriction* f_k consists of the $|f|/2^k$ points of f contained in the subspace $\{0,1\}^{n-k}$ where all the canonical variables of f have value 0.

Note that f_k depends only on the $n-k$ noncanonical variables of f. Once L_f has been computed (see next section), the canonical variables of f are known, and f_k can be immediately determined applying Definition 4. For instance, for the function f with $k = 2$ of Example 5, f_2 depends on the noncanonical variables x_2, x_3, x_4. To build f_2 we take the subset {00001, 00100, 00110} of the points of f for which the canonical variables x_0, x_1 have value 0, and then project these points onto the $\{0,1\}^3$ subspace relative to x_2, x_3, x_4, where we have $f_2 = \{001,\ 100,\ 110\}$; or, equivalently, f_2 is $\overline{x}_2\overline{x}_3x_4 + x_2\overline{x}_3\overline{x}_4 + x_2x_3\overline{x}_4$.

The importance of the restriction stems from the fact that the SPP-minimal form of any k-autosymmetric function f can be easily derived from the SPP-minimal form of f_k, and finding the latter is easier because it depends on less variables and contains less points. (In the example above f depends on five variables and has 12 points while f_2 depends on three variables and has only three points). An important result of [19] is extended as follows.

Lemma 1: A k-autosymmetric function f and its restriction f_k have the same number of pseudoproducts in their minimal SPP forms.

Proof: We show that there is a one-to-one correspondence between prime pseudoproducts of f and prime pseudoproducts of its restriction f_k. a) Each prime pseudoproduct of f corresponds to a prime pseudoproduct of f_k. In fact, each pseudocube associated to a prime pseudoproduct p of f is divided into 2^k equal pseudocubes lying in the subspaces where the canonical variables of f assume all the possible values. The pseudocube lying in the subspace where the canonical variables are set to zero corresponds to a prime pseudoproduct of f_k. Indeed, it is easy to verify by contradiction that the nonprimality of this pseudoproduct would imply the nonprimality of p. b) Each prime pseudoproduct of f_k corresponds to a prime pseudoproduct of f. In fact, f_k is the projection of f onto the subspace where all the canonical variables are set to zero, and for each prime pseudoproduct of f_k, there exists an equivalent pseudoproduct in all the other subspaces corresponding to all the other possible settings of the canonical variables of f. Since all these pseudoproducts have the same structure, they can be unified to form a prime pseudoproduct of f. The thesis follows immediately from this one-to-one correspondence. ■

Based on Lemma 1, we can prove a stronger property, namely a minimal form for f can be easily derived from a minimal form for f_k. Let $x_{z_0}, \ldots, x_{z_{n-k-1}}$ be the noncanonical variables of f, and let $\mathrm{STR}(L_f) = p_0 p_1 \ldots p_{n-k-1}$, where p_i is the EXOR factor containing x_{z_i}, $0 \leq i \leq n-k-1$ (recall that each noncanonical variable appears in exactly one EXOR factor, and each EXOR factor contains exactly one noncanonical variable). We have Theorem 5.

Theorem 5: If $\mathrm{SPP}(f_k)$ is a minimal SPP form for f_k, then the form $\mathrm{SPP}(f)$ obtained by substituting in $\mathrm{SPP}(f_k)$ each variable x_{z_i} with the EXOR factor p_i is a minimal SPP form for f.

Proof: By Lemma 1, the number of pseudoproducts in $\mathrm{SPP}(f)$ is minimum, then we have only to prove that this form covers exactly all the points of f. When we transform f into f_k, we select the vector $\boldsymbol{u}^i$ with all canonical variables set to

zero from each affine subspace $\boldsymbol{w}^i \oplus L_f$. Call $\pi(\boldsymbol{u}^i)$ the vector $\boldsymbol{u}^i$ without the canonical variables, i.e., its projection onto a subspace $\{0,1\}^{n-k}$. When we apply the linear substitutions $x_{z_0} \leftarrow p_0, \ldots, x_{z_{n-k-1}} \leftarrow p_{n-k-1}$, we force any pseudo-product that covers $\pi(\boldsymbol{u}^i)$ in f_k to cover all the points in $\boldsymbol{w}^i \oplus L_f$ in f, and the thesis immediately follows. ■

Note that the resulting expression may be reduced using some properties of EXOR, in particular $x \oplus x = 0$ and $0 \oplus x = x$.

Example 6: The function f in Fig. 2 can be reduced to the function f_2. Its linear space L_f has structure $x_1(x_0 \oplus x_2 \oplus x_3)$, which implies the substitutions $x_1 \leftarrow x_1$ and $x_3 \leftarrow x_0 \oplus x_2 \oplus x_3$. A minimal SPP expression of f_2 is $x_1 + x_3$. By Theorem 5, a minimal SPP expression for f is

$$(x_1 + x_3)[x_1 \leftarrow x_1,\ x_3 \leftarrow x_0 \oplus x_2 \oplus x_3] = x_1 + (x_0 \oplus x_2 \oplus x_3)\,.$$

Another example of minimization of an autosymmetric function f will be given in Section V.

C. Relation With Different Notions of Symmetry

A Boolean function is generally called symmetric if is invariant under any permutation of its variables (see for example [16]). This property is actually unrelated to autosymmetry. As known, the total number of symmetric functions is 2^{n+1}, much smaller than the one of autosymmetric functions (see Theorem 4); still symmetric functions do not form a subset of the autosymmetric ones. In fact, a symmetric function may be autosymmetric (e.g., the parity function), but there are symmetric functions that are not autosymmetric (e.g., any symmetric function with an odd number of points). The concept of symmetry has been extensively used for functions classification, and for easing the minimization process in some cases. However symmetric functions are not as common as autosymmetric ones in practical applications, and do not seem to yield as remarkable advantages as the latter ones in the synthesis process.

Another approach introduced in [18] under the name of *support-reducing decomposition* is worth mentioning here. On one hand, the underlying concept at the base of support-reducing decomposition can be seen as an extension of the one of autosymmetry already introduced in [19], because is aimed at specifying a function $f(x_1, \ldots, x_n)$ as $f = h(g_1(x_1, \ldots, x_s), \ldots, g_k(x_1, \ldots, x_s), x_{s+1}, \ldots, x_n)$, with $k < s$. On the other hand the work is directed at selecting the functions g_i from a predefined library set, so that finding a minimal form for f becomes library dependent.

Much more interesting, instead, seem to be the similarities between the class of autosymmetric functions and the well studied class of self–dual functions [25], [10]. A completely specified Boolean function f is *self–dual* if for all $\boldsymbol{w} \in \{0,1\}^n$, $f(\boldsymbol{w}) = \overline{f}(\overline{\boldsymbol{w}})$, where $\overline{f}$ is the function whose one-set is the off-set of f and $\overline{\boldsymbol{w}} = \boldsymbol{w} \oplus \mathbf{1}$. For example, the function $f = \{001,\ 010,\ 100,\ 111\}$ is self–dual, because $001 \oplus 111 = 110 \in \overline{f}$, $010 \oplus 111 = 101 \in \overline{f}$, $100 \oplus 111 = 011 \in \overline{f}$, and $111 \oplus 111 = 000 \in \overline{f}$. The relation between autosymmetric and self–dual functions is better understood generalizing the latter notion in the following.

Definition 5: A Boolean function f in $\{0,1\}^n$ is *dual-closed under* $\boldsymbol{\alpha}$, with $\boldsymbol{\alpha} \in \{0,1\}^n$, if for each $\boldsymbol{w} \in \{0,1\}^n$, $\boldsymbol{w} \oplus \boldsymbol{\alpha} \in f$ if and only if $\boldsymbol{w} \in \overline{f}$.

Note that a self–dual function is dual-closed under the vector $\mathbf{1}$. As the set L_f is a linear subspace of $\{0,1\}^n$ (Proposition 1), we have a similar property for self–dual functions in Proposition 2.

Proposition 2: The set $A_f = \{\boldsymbol{\beta}$: f is dual-closed under $\boldsymbol{\beta}\}$ is an *affine subspace* of $\{0,1\}^n$.

Proof: We will show that A_f is closed under the EXOR of an odd number of its vectors, as this implies that A_f is an affine subspace of $\{0,1\}^n$ by a classical linear algebra result. In fact, suppose that f is dual closed under $\boldsymbol{\alpha}$, $\boldsymbol{\beta}$ and $\boldsymbol{\gamma}$. Then we have: $f(\boldsymbol{w} \oplus \boldsymbol{\alpha} \oplus \boldsymbol{\beta} \oplus \boldsymbol{\gamma}) = \overline{f}(\boldsymbol{w} \oplus \boldsymbol{\alpha} \oplus \boldsymbol{\beta}) = f(\boldsymbol{w} \oplus \boldsymbol{\alpha}) = \overline{f}(\boldsymbol{w})$, and the thesis follows. ■

Definition 6: A Boolean function f is k*-self–dual*, $0 \leq k \leq n$, if its *affine space* A_f has dimension k.

Observe that a k-self–dual function has exactly 2^{n-1} points in its one-set.

Example 7: The function $f = \{000,\ 011,\ 101,\ 110\}$ is 2-self–dual and its affine space A_f is $\{001,\ 010,\ 100,\ 111\}$. For instance, $000 \oplus 001 = 001 \in \overline{f}$, $011 \oplus 001 = 010 \in \overline{f}$, $101 \oplus 001 = 100 \in \overline{f}$, and $110 \oplus 001 = 111 \in \overline{f}$.

The most relevant relation between k-autosymmetric functions and k-self–dual functions is given in the following.

Theorem 6: If f is k-self–dual, then f is k-autosymmetric.

Proof: Let $\boldsymbol{\alpha} \in A_f$. We show that f is closed under all vectors in $\boldsymbol{\alpha} \oplus A_f$. Indeed, for all $\boldsymbol{\beta} \in A_f$ and for all $\boldsymbol{w} \in \{0,1\}^n$, we have: $f(\boldsymbol{w} \oplus \boldsymbol{\alpha} \oplus \boldsymbol{\beta}) = \overline{f}(\boldsymbol{w} \oplus \boldsymbol{\alpha}) = f(\boldsymbol{w})$. Since A_f is an affine space and $\boldsymbol{\alpha} \in A_f$, the set $\boldsymbol{\alpha} \oplus A_f$ is a linear space. Therefore, $L_f = \boldsymbol{\alpha} \oplus A_f$, and the dimensions of L_f and A_f are equal to k. ■

Finally, note that the concept of self–duality has never been applied to express functions in reduced algebraic forms.

V. Minimization Algorithm

In the previous section, we have shown that each Boolean function f is k-autosymmetric, for $k \geq 0$. For minimization purposes we have an increasing advantage for increasing $k \geq 1$, as minimizing a k-autosymmetric function with n variables and ℓ points reduces to minimizing a different function with $n - k$ variables and $\ell/2^k$ points. Even for $k = 1$ we have to cover only one half of the original points.

Fortunately, for a given function f, finding the associated linear space L_f and computing the autosymmetry degree k is an easy task, because the required algorithm is polynomial in the number n of variables and in the number of points of f.

We now give some intuition behind the computation of L_f. By Definition 2, a function f is closed under $\boldsymbol{\alpha}$ if for any $\boldsymbol{u} \in f$ there exists $\boldsymbol{v} \in f$ such that $\boldsymbol{v} = \boldsymbol{u} \oplus \boldsymbol{\alpha}$. Thus, for all $\boldsymbol{u} \in f$, there exists a vector $\boldsymbol{v} \in f$ such that we can express $\boldsymbol{\alpha}$ as $\boldsymbol{\alpha} = \boldsymbol{u} \oplus \boldsymbol{v}$. In other words, $\boldsymbol{\alpha}$ must be searched within the vectors of the set $\{\boldsymbol{u} \oplus \boldsymbol{v} \mid \boldsymbol{u}, \boldsymbol{v} \in f\}$. More precisely, we have Theorem 7.

Theorem 7: Let f be a Boolean function. Then $L_f = \bigcap_{\boldsymbol{u} \in f} (\boldsymbol{u} \oplus f)$.

Proof: Let $\boldsymbol{\alpha} \in \bigcap_{\boldsymbol{u} \in f} (\boldsymbol{u} \oplus f)$. Then $\boldsymbol{\alpha} \in \bigcap_{\boldsymbol{u} \in f} (\boldsymbol{u} \oplus f) \Leftrightarrow \forall\, \boldsymbol{u} \in f, \boldsymbol{\alpha} \in \boldsymbol{u} \oplus f \Leftrightarrow \forall\, \boldsymbol{u} \in f, \exists\, \boldsymbol{v} \in f \mid \boldsymbol{\alpha} = \boldsymbol{u} \oplus \boldsymbol{v} \Leftrightarrow \boldsymbol{\alpha} \in L_f$. ■

Based on Theorem 7, we state Algorithm 1.

Algorithm 1: ***Construction of L_f*** *(build L_f and find the autosymmetry degree k of a given function f)*

1) *for all $\boldsymbol{u} \in f$ build the set $\boldsymbol{u} \oplus f$;*
2) *build the set $L_f = \bigcap_{\boldsymbol{u} \in f} (\boldsymbol{u} \oplus f)$;*
3) *compute $k = \log |L_f|$.*

The time complexity of Algorithm 1 is $\Theta(|f|^2 n)$, because we must build a set $\boldsymbol{u} \oplus f$ for all $\boldsymbol{u} \in f$, and the construction of each such a set requires $\Theta(|f|\, n)$ time.

Any SPP minimization algorithm can be easily extended for exploiting autosymmetry. For a given function f we first compute L_f and k with Algorithm 1. If $k = 0$ (i.e., f is not autosymmetric) we proceed with regular minimization, otherwise we compute the restriction f_k of f, minimize it, and finally derive a minimal form $\mathrm{SPP}(f)$ from $\mathrm{SPP}(f_k)$. We propose the following A (for autosymmetry) minimization algorithm.

Algorithm 2: ***A-Minimization*** *(build a minimum SPP form of a given function f)*

1) *build L_f and compute the value of k by Algorithm 1;*
2) **if** $k = 0$ **then**
   a) *minimize f with any SPP synthesis algorithm*
3) **else**
   a) *determine the canonical variables of L_f and compute the restriction f_k as indicated in Section IV;*
   b) *compute $STR(L_f) = p_0 p_1 \cdots p_{n-k-1}$;*
   c) *compute the minimal form $SPP(f_k)$ for f_k with any SPP synthesis algorithm;*
   d) *build SPP(f) by substituting in $SPP(f_k)$ each noncanonical variable x_{z_i} with the EXOR factor p_i of $STR(L_f)$.*

By the theory developed in the previous section, Algorithm 2 is correct. Note that the algorithm builds an SPP form minimal with respect to the number of pseudoproducts. To obtain the minimal SPP form with respect to the number of literals we must slightly rearrange steps *3(c)* and *3(d)*, executing the substitutions of all p_i for x_{z_i} in the prime pseudoproducts of f_k, before selecting such pseudoproducts in the set covering problem implicit in the minimization algorithm.

Example 8: Minimization of the function f of Example 5, using Algorithm 2.

- Derive L_f and k by Algorithm 1. For this purpose, for all $\boldsymbol{u} \in f$ compute the set $\boldsymbol{u} \oplus f$. For example, for the point 00 100 we obtain the set:

$$00\,100 \oplus f = \{00101,\ 00000,\ 00010,\ 01100,\ 01110,\ 01001,\ 10101,\ 10111,\ 10000,\ 11100,\ 11001,\ 11\,011\}.$$

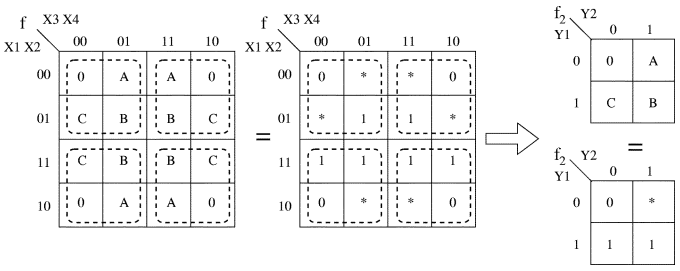

Fig. 4. In the Karnaugh map on the left, each point of $O_f \cup D_f$ is represented with a letter. Points in the same affine space are represented by the same letter. The central and the right maps represent f and its restriction f_k, respectively.

The intersection of all the sets $\boldsymbol{u} \oplus f$ gives the linear space $L_f = \{00000,\ 01100,\ 10101,\ 11\,001\}$. We then have $k = 2$.

- Since $k = 2$ we proceed with the **else** branch of Algorithm 2. L_f has noncanonical variables x_2, x_3, x_4, hence f_2 is restricted to these variables and we have: $f_2 = \{001,\ 100,\ 110\}$.
- The minimization problem now consists of finding a minimal SPP cover of the points of f_2. Applying the algorithm of [3] we have the minimal form $\mathrm{SPP}(f_2) = x_2\overline{x}_4 + \overline{x}_3(x_2 \oplus x_4)$.
- Compute: $\mathrm{STR}(L_f) = (x_0 \oplus x_1 \oplus x_2)x_3(x_0 \oplus x_4)$.
- Derive the minimal SPP form for f by substituting x_2, x_3 and x_4 in $\mathrm{SPP}(f_2)$ with the EXOR factors of $\mathrm{STR}(L_f)$, respectively $(x_0 \oplus x_1 \oplus x_2)$, x_3 and $(x_0 \oplus x_4)$. We obtain $\mathrm{SPP}(f) = (x_0 \oplus x_1 \oplus x_2)(x_0 \oplus \overline{x}_4) + \overline{x}_3(x_1 \oplus x_2 \oplus x_4)$, with some immediate algebraic simplifications, e.g., in the first term of $\mathrm{SPP}(f_k)$ we have: $\overline{x}_4[x_4 \leftarrow (x_0 \oplus x_4)] = \overline{(x_0 \oplus x_4)} = (x_0 \oplus \overline{x}_4)$. In the last term of $\mathrm{SPP}(f_k)$ we have:

$$\begin{aligned}(x_2 \oplus x_4)[x_2 \leftarrow &(x_0 \oplus x_1 \oplus x_2), x_4 \leftarrow (x_0 \oplus x_4)] \\ &= (x_0 \oplus x_1 \oplus x_2) \oplus (x_0 \oplus x_4) \\ &= (x_0 \oplus x_0 \oplus x_1 \oplus x_2 \oplus x_4) \\ &= (x_1 \oplus x_2 \oplus x_4).\end{aligned}$$

VI. Incompletely Specified Autosymmetric Functions

Let us now discuss how to extend the notion of autosymmetry to functions with don't care points (denoted by $*$). For an incompletely specified Boolean function $f : \{0,1\}^n \rightarrow \{0,1,*\}$, the sets of points on which f takes value 0, or 1, or $*$, are respectively called the *zero-set* Z_f, the *one-set* O_f, and the *don't-care-set* D_f.

Definition 7: An incompletely specified function f is *k-autosymmetric* if the completely specified function g with $O_g = O_f \cup D_f$ is k-autosymmetric.

The restriction f_k of an incompletely specified autosymmetric function f is in general incompletely specified. For defining the one-set and don't-care-set of f_k, let us first discuss an example.

Example 9: Consider the function f represented in the central map of Fig. 4. The map on the left shows the points of $O_f \cup D_f$ marked with letters, with the points in the same affine space marked with the same letter. If f contains an affine space

TABLE I

DISTRIBUTION OF k-AUTOSYMMETRIC FUNCTIONS IN THE ESPRESSO BENCHMARK SUITE. $\#\mathbf{funct}$ IS THE TOTAL NUMBER OF FUNCTIONS (SINGLE OUTPUTS) FOR ANY VALUE OF k, AND $\%\mathbf{funct}$ IS THE CORRESPONDING PERCENTAGE. THE NEXT THREE ROWS REPORT THE NUMBER OF k-AUTOSYMMETRIC FUNCTIONS: 1) SYNTHESIZED WITH THE NEW ALGORITHM *A* AND WITH THE PREVIOUS BEST ALGORITHM *C* ($\#\mathbf{AC}$); 2) SYNTHESIZED WITH *A* ONLY ($\#\mathbf{A}$), SINCE *C* DID NOT TERMINATE; 3) NOT SYNTHESIZED AT ALL SINCE BOTH ALGORITHMS DID NOT TERMINATE ($\#^*$). $\mathbf{T_A}$ AND $\mathbf{T_C}$ ARE THE AVERAGE CPU TIMES T_A, T_C IN SECONDS, REQUIRED BY THE TWO ALGORITHMS ON THE SAME FUNCTION ON A PENTIUM III 450 MACHINE. $\mathbf{T_A/T_C}$ IS THE AVERAGE OF THE RATIO T_A/T_C, FOR THE 546 FUNCTIONS OF ROW $\#\mathbf{AC}$ ($\mathbf{k} = 0$ TO 8). $\mathbf{k/n}$ IS THE AVERAGE OF THE RATIO k/n, WHERE n IS THE NUMBER OF INPUT VARIABLES

k=	**0**	**1**	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**	**Tot**
#funct	365	116	72	95	41	43	39	27	16	24	58	28	18	942
%funct	38.9	12.3	7.6	10.1	4.3	4.6	4.1	2.9	1.7	2.4	6.2	3.0	1.9	100
#AC	264	82	55	54	23	35	21	10	2	0	0	0	0	546
#A	0	5	4	5	9	7	16	9	14	24	58	28	18	197
#*	101	29	13	36	9	1	2	8	0	0	0	0	0	199
$\mathbf{T_A}$	101.94	6.42	1.37	0.19	0.05	0.09	0.10	0.08	0.41	-	-	-	-	-
$\mathbf{T_C}$	101.8	222.3	109.7	203.2	4.8	51.5	90.1	23.6	2880	-	-	-	-	-
$\mathbf{T_A/T_C}$	1.01	0.63	0.42	0.37	0.32	0.27	0.07	0.004	0.0001	-	-	-	-	-
k/n	0	0.12	0.23	0.31	0.43	0.55	0.58	0.66	0.62	0.59	0.63	0.73	0.75	0.4

whose points belong to O_f (e.g., the points marked *B*), the corresponding point of f_k belongs to O_{f_k}. If all the points of an affine space belong to D_f (e.g., the points marked A), the corresponding point of f_k belongs to D_{f_k}. For an affine space composed of points from both O_f and D_f (e.g., the points marked C), the corresponding point of f_k must be in O_{f_k}, since the points in its affine space that are in O_f must be covered in the final solution. Therefore, in Fig. 4 the affine space corresponding to C maps to an element of O_{f_k}.

Formally, let S_0 denote the set of points in $O_f \cup D_f$ projected onto the subspace $\{0,1\}^{n-k}$ where all the canonical variables of f have value 0. We pose the following definition.

Definition 8: For a ($k \geq 1$)-autosymmetric incompletely specified function f, the *restriction* f_k is the incompletely specified function such that: a) D_{f_k} is the set of points $\boldsymbol{u} \in S_0$ for which the affine space associated to $\boldsymbol{u}$ in f is contained in D_f and b) $O_{f_k} = S_0 \setminus D_{f_k}$.

We can now generalize Theorem 5 to incompletely specified autosymmetric functions.

Theorem 8: Let f and f_k be as in Definition 8. If $\mathrm{SPP}(f_k)$ is a minimal SPP form for f_k, the form $\mathrm{SPP}(f)$ obtained from $\mathrm{SPP}(f_k)$ by substituting each variable x_{z_i} with the EXOR factor p_i is a minimal SPP form for f.

Proof: For any vector $\boldsymbol{v} \in \{0,1\}^n$ we denote by $\pi(\boldsymbol{v})$ its projection in the space $\{0,1\}^{n-k}$ where all the canonical variables of f are set to 0. Let $\boldsymbol{v} \in D_f$ and $\pi(\boldsymbol{v}) \in O_{f_k}$. By contradiction, suppose that $\boldsymbol{v}$ is not covered by any minimal SPP form for f. Of course any minimal SPP form for f_k covers $\pi(\boldsymbol{v})$. By Definition 8 we know that there exists at least a point $\boldsymbol{u}$ in the affine space containing $\boldsymbol{v}$ (i.e., $\boldsymbol{u} \in \boldsymbol{v} \oplus L_f$) such that $\boldsymbol{u} \in O_f$. Let p be a pseudocube covering $\boldsymbol{u}$ in a minimal SPP form for f. We can replace p, in such form, with a pseudoproduct covering not only $\boldsymbol{u}$ but all $\boldsymbol{u} \oplus L_f$ (i.e., the affine space containing both $\boldsymbol{u}$ and $\boldsymbol{v}$). We have obtained a minimal SPP form for f covering $\boldsymbol{v}$, thereby contradicting the initial hypothesis. ■

Although the given generalization of autosymmetry to functions with don't cares may appear quite restrictive, our experimental results show that more than 40% of the outputs for the incompletely specified functions in the classical ESPRESSO benchmark suite are autosymmetric (see next section). Indeed, Definition 7 is just a possible one, as it takes *all* the don't cares of f as points of the function to be actually synthesized. This choice guarantees the minimality of the SPP form for f, given a minimal SPP form for f_k (Theorem 8).

VII. EXPERIMENTAL RESULTS

The new minimization Algorithm 2, also called Algorithm *A* (for autosymmetry), has been tested on a large set of functions taken from the ESPRESSO benchmark suite [27]. The different outputs for each function have been synthesized separately. The performance of Algorithm *A* has been compared with the performance of the best previous algorithm, that is the one proposed in [3], in the following indicated as Algorithm *C* (after Ciriani). In fact, the minimization of function f_k in Algorithm *A* [step 3(c)] has been implemented with Algorithm *C*. The input of both algorithms is the on-set and don't care-set of the functions.

For all the functions considered we have computed the values of the autosymmetry degree k with Algorithm 1, obtaining the results shown in the first two rows of Table I for completely specified functions, and in Table II for functions with don't cares. Surprisingly the overall percentage of autosymmetric completely specified functions ($k \geq 1$) is over the 61% (Table I); while more than 40% of the functions with don't care set are autosymmetric according to Definition 7 (Table II). We have then attempted to run Algorithms *A* and *C* for all the completely specified functions of our test set, recording the CPU times whenever the computation terminated in less than 172 800 seconds (2 days) on a Pentium III 450 machine. Results on program termination are given in rows $\mathbf{T_A}$, $\mathbf{T_C}$ of Table I.

We have found out that the autosymmetry property drastically reduces the minimization time, as reported in row $\mathbf{T_A/T_C}$ of Table I that shows the average reduction of computing time using Algorithm *A* (time $\mathbf{T_A}$) instead of *C* (time $\mathbf{T_C}$), for all the benchmark functions for which both algorithms terminated (i.e., for the 546 functions of row $\#\mathbf{AC}$). Note how the improvement introduced by the new algorithm drastically increases for increasing k. For $k = 0$, instead, we have $\mathrm{Avg}(T_A/T_C) > 1$, and actually the ratio T_A/T_C is slightly greater than 1 for each such a function. This is because Algorithm *A* computes L_f in any case, then calls Algorithm *C*. The resulting slowdown is however always negligible because L_f is computed in polynomial time by Algorithm 1 (see Section V), while Algorithm *C* is

TABLE II
DISTRIBUTION OF k-AUTOSYMMETRIC FUNCTIONS WITH A DON'T CARE SET, IN THE ESPRESSO BENCHMARK SUITE. $\#\mathbf{funct}$ IS THE TOTAL NUMBER OF FUNCTIONS (SINGLE OUTPUTS) FOR ANY VALUE OF k, AND $\#\mathbf{funct}$ IS THE CORRESPONDING PERCENTAGE

k	#funct	%funct
0	83	53.90
1	17	11.04
2	8	5.19
3	23	14.94
4	8	5.19
5	10	6.49
6	4	2.60
8	1	0.65

TABLE III
DETAILED RESULTS FOR A SUBSET OF AUTOSYMMETRIC FUNCTIONS. COLUMNS $|\mathbf{f}|$, $\mathbf{n}$, $\mathbf{k}$ AND $\mathbf{k/n}$ REPORT THE NUMBER OF POINTS OF THE FUNCTION, AND THE VALUES OF n, k, AND k/n, RESPECTIVELY. COLUMNS $\mathbf{T_A}$ AND $\mathbf{T_C}$ REPORT CPU TIMES AS IN TABLE I (A STAR INDICATES NON TERMINATION AFTER 172 800 SECONDS). THE LAST COLUMN REPORTS THE RATIO T_A/T_C. THE RESULTS ARE RELATIVE TO SINGLE OUTPUTS. $\#\mathbf{L}$ AND $\#\mathbf{PP}$ REPORT THE NUMBERS OF LITERALS, AND OF PRIME PSEUDOPRODUCTS, IN THE MINIMAL EXPRESSION

function	|f|	n	k	k/n	#L	#PP	T_A	T_C	T_A/T_C
max512(0)	258	9	1	0.11	8	2	20.82	2618.02	0.08
Z9sym(0)	420	9	1	0.11	102	17	310.17	*	*
newtpla2(2)	204	10	2	0.20	17	4	1.71	236.89	0.07
radd(2)	128	8	3	0.38	16	3	0.22	5.43	0.04
addm4(6)	224	9	4	0.44	18	4	0.39	79.86	0.005
intb(0)	13888	15	5	0.33	96	12	11420.80	*	*
addm4(7)	192	9	6	0.67	6	2	0.27	123.63	0.002
intb(5)	16384	15	7	0.46	38	8	431.60	*	*
newtpla(4)	256	15	8	0.57	7	1	0.48	2893.04	0.0002
opa(17)	33792	17	10	0.59	37	7	503.01	*	*
alcom(5)	14336	15	11	0.73	6	3	119.67	*	*

TABLE IV
DETAILED RESULTS FOR A SUBSET OF BENCHMARK CIRCUITS. COLUMNS $\mathbf{n}$, $\#\mathbf{O}$, $\mathbf{Avg(k)}$ AND $\mathbf{Avg(k)/n}$ REPORT THE NUMBER OF INPUT VARIABLES, THE NUMBER OF OUTPUTS, THE AVERAGE AUTOSYMMETRY DEGREE OF THE SINGLE OUTPUTS, AND ITS RATIO WITH n, RESPECTIVELY. COLUMNS $\mathbf{T_A}$, $\mathbf{T_C}$, AND $\mathbf{T_A/T_C}$, REPORT TOTAL CPU TIMES IN SECONDS, AND THEIR RATIO, FOR THE COMPUTATION OF ALL OUTPUTS (FOR *'S SEE TABLE III)

function	n	#O	Avg(k)	Avg(k)/n	T_A	T_C	T_A/T_C
Z9sym	9	1	1.0	0.11	310.17	*	*
adr4	8	5	3.2	0.40	88.22	145.29	0.61
max512	9	4	0.5	0.06	3042.22	6420.00	0.46
newapla1	12	6	5.3	0.44	0.38	2.81	0.14
risk	8	31	3.3	0.41	0.96	516.66	0.002
z4	7	4	3.0	0.43	6.75	9.44	0.72

exponential in nature. For all the functions in the table the forms obtained with Algorithms *A* and *C* coincide. Table III shows the CPU times for a small subset of the above functions with $k \geq 1$, and other relevant minimization parameters for them. Note that Tables I–III show results for single outputs of different benchmark circuits. Table IV, instead, shows experimental results for whole benchmark circuits, with reference to all their outputs.

We now compare the cost of the solutions generated with our algorithm with the once of two level SOP minimization and of the widely studied EX-SOP three-level logic synthesis [7], [12], [13]. To this end, we count the number of literals and gates (AND and EXOR) of an expression. In the multilevel contest the cost function is the number of literals in each different gate (see [14], [15]). For example, we represent the multilevel network of expression $e = ((xyz) + (z \oplus xyz) + x)y$ with the four equations

$$a = xyz, \quad b = z \oplus a, \quad c = a + b + x, \quad e = cy$$

(each equation corresponds to a gate) and we count the 10 literals on the right hand sides of the equations. The problem is that, in many technologies, EXOR and OR (or AND) gates have different costs. In [15] the authors consider a 2-input EXOR gate as $x \oplus y = xy + \overline{xy}$. Thus, the cost in literals of an 2-input EXOR gate is 4, while the cost of the 2-input OR and AND gates is 2. This corresponds also to the number of transistors used for the CMOS technology mapping (i.e., 4 transistors for AND/OR gates and 8 transistors for the EXOR gate). In general the associative property of the EXOR operator allows to see a k-input EXOR gate as the composition of $k-1$, 2-input EXOR gates. For example, $x_1 \oplus x_2 \oplus x_3 = (x_1 \oplus x_2) \oplus x_3$, and $x_1 \oplus x_2 \oplus x_3 \oplus x_4 \oplus x_5 = ((x_1 \oplus x_2) \oplus (x_3 \oplus x_4)) \oplus x_5$. Therefore, we can use a cost function μ_C, where a k-input EXOR gate costs $4(k-1)$, and a k-input OR/AND gate costs k. This function corresponds to the CMOS cost described in [14], where the expression e of the previous example has cost $\mu_C = 12$. In [14], a different cost function is also proposed for FPGA [23] realiza-

TABLE V
CMOS AND FPGA COSTS FOR SOME BENCHMARK FUNCTIONS IN THE SPP, SOP, AND EX-SOP SYNTHESIS. μ_C AND μ_F ARE THE CMOS AND FPGA COSTS FOR THE SPP NETWORK, RESPECTIVELY. μ'_C IS THE CMOS COST FOR THE SOP NETWORK, AND ITS FPGA COST IS $\mu'_F = \mu'_C$. μ''_C IS THE CMOS COST FOR THE EX-SOP NETWORK, AND ITS FPGA CIOST IS $\mu''_F = \mu''_C - 2$. #E IS THE NUMBER OF DIFFERENT EXOR GATES IN THE SPP FORM

	SPP			SOP	SPP vs SOP		EX-SOP	SPP vs EX-SOP	
function	μ_C	μ_F	#E	μ'_C	μ_C/μ'_C	μ_F/μ'_F	μ''_C	μ_C/μ''_C	μ_F/μ''_F
5xp1	64	60	2	341	0.20	0.18	146	0.44	0.42
adr4	117	79	10	415	0.28	0.19	108	1.08	0.74
life	180	148	16	746	0.24	0.20	509	0.35	0.29
mlp4	524	400	32	853	0.61	0.47	433	1.21	0.92
rd53	64	46	7	171	0.37	0.27	79	0.81	0.60
rd73	207	127	15	883	0.23	0.14	312	0.66	0.41
rd84	420	330	31	2029	0.20	0.17	650	0.65	0.51
z4	100	62	10	311	0.32	0.20	146	0.68	0.43

tion, where k-input EXOR gates and k-input AND/OR gates have the same cost k. We call this cost μ_F. The results of comparing SPP, SOP, and EX-SOP expressions with this measures are reported in Tables V and VI.

Note that minimization of multiple output circuits has been carried out individually for each output. Such outputs may not be function of all input variables (i.e., the corresponding functions are degenerated). In our method, such irrelevant variables are not eliminated beforehand, since they are discovered during the evaluation of autosymmetry at no additional cost. Doing this, the degenerated functions are brought into the class of autosymmetric ones. Note that this is not a disadvantage of our approach if compared with others, as none of the minimization methods in current use detects the irrelevant variables in a preliminary phase.

VIII. A DISCUSSION ON AUTOSYMMETRY

To understand the role of autosymmetric functions, we must compare them with the set of all possible functions. The total number of Boolean functions of n variables is $N_F = 2^{2^n}$, corresponding to all the ways a subset of points can be chosen in $\{0,1\}^n$. This is a huge number, however, due to the randomness of the above generating process very many of such functions do not correspond to any significant circuit. Autosymmetric functions are just a subset of the above. The autosymmetric functions are $N_A = (2^n - 1)2^{2^n - 1}$ in number (see Theorem 4). Therefore, for increasing n, autosymmetric functions constitute a vanishing fraction of all the functions, as N_A/N_F goes to zero for n going to infinity. Still the question remains on how many significant functions are autosymmetric.

A key observation is that most of the major benchmark functions are indeed autosymmetric, as shown in the previous section. The more so when n is small and the values of N_F and N_A are not too distant. The reason, we might argue, is that a function encoding a real life problem must exhibit a regular structure that can be reflected in some degree of autosymmetry. In fact, also "degenerated" functions that do not depend on all the variables are autosymmetric, although the converse is not true in general. Although some degenerated functions are encountered in the major benchmarks, this property is not immediately evident and has not been directly used in the standard minimization processes.

TABLE VI
MINIMIZATION TIMES (IN SECONDS) FOR SOME BENCHMARK FUNCTIONS IN THE SPP, SOP, AND EX-SOP FORMS. SPP EXPERIMENTS WERE PERFORMED ON A SINGLE 450 MHZ CPU WITH 128 MB MEMORY. EXPERIMENTAL TIMES FOR SOP AND EX-SOP WERE OBTAINED WITH A SUN ULTRA 60 OPERATING ON TWO 360 MHZ CPUS AND 768 MB MEMORY

function	Avg(k)	SPP	SOP	EX-SOP
f51m	3.6	201.30	0.90	55
life	0.0	122.59	0.42	200
radd	3.2	87,93	0.24	33
rd53	1.7	0.10	0.01	13
rd73	2.0	60.75	0.11	494
z4	3.0	6.75	0.11	9

Another important observation is that regularity may also allow to define an autosymmetric function f independently on the number of variables, and then to state a rule for deriving a minimal form for f valid for any n. Well known functions as, for example, the ones counting the parity of n bits, or giving the next-state values for an n bits Gray code, can be easily expressed in minimal form for an arbitrary number of variables just because they are autosymmetric (for the parity see [19]; for Gray codes elementary considerations suffice).

The relation between autosymmetric functions and functions which are simply symmetric might be better investigated (see Section IV-C). No interesting results seem to emerge from the analysis conducted as far. We simply note that the total number of symmetric functions is $2^{n+1} \ll N_A$, but symmetric functions do not form a subset of the autosymmetric ones, as already observed.

The introduction of the restriction f_k for a k-autosymmetric function f leads to consider the nature of Boolean functions under a new light. We can state that the *information content* of f is represented by f_k together with the linear transformation $x_{z_0} \leftarrow p_0, \ldots, x_{z_{n-k-1}} \leftarrow p_{n-k-1}$ (Section IV), so that the core of the synthesis problem is the minimization of f_k (Section V). This suggests a formal generalization. For a given function $g \in \{0,1\}^m \rightarrow \{0,1\}$ we can define the *autosymmetry class* $\mathcal{C}_g$ as the class of all the autosymmetric functions $f \in \{0,1\}^t \rightarrow \{0,1\}, t \geq m$, such that $f_k \equiv g$. Since the information content of any given function f can be easily found, and a minimal SPP form for f can then be derived from $\mathrm{SPP}(f_k)$, minimizing the function f_k corresponds to minimize the entire class $\mathcal{C}_g = \mathcal{C}_{f_k}$. Exploiting the full potential of such an approach is currently a matter of study.

IX. Conclusion

In this paper, we have introduced the autosymmetry degree k of a Boolean function f of n variables. The value of k is a measure of the regularity of f. This approach supplies a new tool for efficient minimization. For $k > 0$ a new function f_k of $n - k$ variables has been defined. f_k is called the restriction of f and can be built in polynomial time.

The importance of f_k has been demonstrated in connection with the construction of minimal three-level SPP expressions. In fact, it has been shown how a minimal SPP expression for f can be built in linear time from a minimal SPP expression for f_k, and how this induces a drastic reduction of the minimization time. This advantage increases with the value of k, since f_k depends on $n - k$ variables only, and has a number of input points equal to the number of points of f divided by 2^k. Our experiments have confirmed the foreseen time reduction, and have also shown that a great number of functions of practical importance are indeed autosymmetric, thus validating the overall interest of our approach.

Our minimization algorithm would probably be greatly improved if formulated on BDD's as its applicability is presently limited by the size of the input. This promising approach is currently under investigation, and constitutes a challenging open problem.

Also some more work is needed in the treatment of don't care conditions. In fact our present definition of autosymmetry for an incompletely specified function f is rather restrictive, as it takes all the don't cares of f as points of the function g to be actually synthesized. Another approach would be selecting *only a subset* of don't cares of f as points of the function g, in order to maximize the autosymmetry degree and reduce the size of the final circuit. Thus far we have been unable to propose a polynomial time algorithm for such a selection.

Acknowledgment

The authors would like to thank T. Villa and E. Bozzo for many important discussions on the subject of this paper.

References

[1] A. Aleksanyan, "Realization of boolean functions by disjunctions of products of linear forms" (in Russian), *Dokl. Akad. Nauk SSSR*, vol. 304, no. 4, pp. 781–784, 1989. Translation in *Soviet Math. Dokl.*, vol. 39(1), 1989.

[2] F. A. Aloul, A. Ramani, I. Markov, and K. Sakallah, "Solving difficult SAT instances in the presence of symmetry," in *ACM/IEEE 39th Design Automation Conf. (DAC)*, 2002, pp. 731–736.

[3] V. Ciriani, "Logic minimization using exclusive OR gates," in *ACM/IEEE 38th Design Automation Conf. (DAC)*, 2001, pp. 115–120.

[4] ——, "A New Approach to Three-Level Logic Synthesis," University of Pisa, Computer Science Dept., Rep. TR-02–03, 2002.

[5] P. Cohn, *Algebra*. New York: Wiley, 1981, vol. 1.

[6] O. Coudert, "Two-level logic minimization: An overview," *INTEGRATION*, vol. 17, pp. 97–140, 1994.

[7] D. Debnath and T. Sasao, "Minimization of AND-OR-EXOR three-level networks with AND gate sharing," *IEICE Trans. Inform. and Syst.*, vol. E80-D(10), pp. 1001–1008, 1997.

[8] ——, "A heuristic algorithm to design AND-OR-EXOR three-level networks," in *Asia and South Pacific Design Automation Conf.*, 1998, pp. 69–74.

[9] ——, "Multiple-valued minimization to optimize PLA's with output EXOR gates," in *IEEE Int. Symp. Multiple-Valued Logic*, 1999, pp. 99–104.

[10] C. Domingo, "Polynomial time algorithms for some self-duality problems," presented at the CIAC: Italian Conf. on Algorithms and Complexity, 1997.

[11] E. Dubrova and P. Ellervee, "A fast algorithm for three-level logic optimization," in *Int. Workshop on Logic Synthesis*, 1999, pp. 251–254.

[12] E. Dubrova, D. Miller, and J. Muzio, "AOXMIN: A three-level heuristic AND-OR-XOR minimizer for boolean functions," in *3rd Int. Workshop on the Applications of the Reed–Muller Expansion in Circuit Design*, 1997, pp. 209–218.

[13] ——, "AOXMIN-MV: A heuristic algorithm for AND-OR-XOR minimization," in *4th Int. Workshop on the Applications of the Reed Muller Expansion in Circuit Design*, 1999, pp. 37–54.

[14] M. Eggerstedt, N. Hendrich, and K. Von der Heide, "Minimization of parity-checked fault-secure AND/EXOR networks," in *IFIP WG 10.2 Workshop on Applications of the Reed–Muller Expansion in Circuit Design*, 1993, pp. 142–146.

[15] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. New York: Kluwer Academic, 1996.

[16] Z. Kohavi, *Switching and Finite Automata Theory*. New York: McGraw Hill, 1970.

[17] V. Kravets and K. Sakallah, "Generalized symmetries of boolean functions," in *International Conf. Computer-Aided Design (ICCAD)*, 2000, pp. 526–532.

[18] ——, "Constructive library-aware synthesis using symmetries," in *Design, Automation and Test in Europe Conf. and Exhibition (DATE)*, 2001, pp. 208–213.

[19] F. Luccio and L. Pagli, "On a new boolean function with applications," *IEEE Trans. Comput.*, vol. 48, pp. 296–310, Mar. 1999.

[20] A. Malik, D. Harrison, and R. Brayton, "Three-level decomposition with application to PLD's," in *IEEE Int. Conf. Computer Design: VLSI in Computer and Processors, ICCD '91*, 1991, pp. 628–633.

[21] P. McGeer, J. Sanghavi, R. Brayton, and A. Sangiovanni-Vincentelli, "ESPRESSO-SIGNATURE: A new exact minimizer for logic functions," *IEEE Trans. VLSI Syst.*, vol. 1, pp. 432–440, Aug. 1993.

[22] M. Perkowski, "A new representation of strongly unspecified switching functions and its application to multi-level AND/OR/EXOR synthesis," in *IFIP WG 10.5 Workshop on Applications of the Reed–Muller Expansion*, 1995, pp. 143–151.

[23] J. Rose, A. E. Gamal, and A. Sangiovanni-Vincentelli, "Architecture of field-programmable gate arrays," *Proc. IEEE*, vol. 81, pp. 1013–1029, July 1993.

[24] T. Sasao, "A design method for AND-OR-EXOR three level networks," in *Int. Workshop on Logic Synthesis*, 1995, pp. 8:11–8:20.

[25] ——, *Switching Theory for Logic Synthesis*. New York: Kluwer Academic, 1999.

[26] T. Villa, T. Kam, R. Brayton, and A. Sangiovanni-Vincentelli, *Synthesis of Finite State Machines: Logic Optimization*. New York: Kluwer Academic, 1997.

[27] S. Yang. (1991) Synthesis on Optimization Benchmarks. User guide, Microelectronic Center of North Carolina. [Online]. Available: ftp://ftp.sunsite.org.uk/computing/general/espresso.tar.Z

Anna Bernasconi received the Laurea degree in physics from University of Pavia, Italy, in 1992, and the Ph.D. in computer science from University of Pisa, Italy, in 1998.

She is currently an associated researcher with the Department of Informatics of the University of Pisa. Her interests include algorithms and complexity, Boolean function complexity, as well as logic synthesis and VLSI design.

Valentina Ciriani (S'01) received the Laurea degree in computer sciences from University of Pisa, Italy, in 1998. She is working toward the Ph.D. degree at the University of Pisa, where she is a recipient of the Italian Government Scholarship since November 1998.

Her interests include logic synthesis and VLSI design, as well as algorithms and data structures. She is a student member of the ACM.

Fabrizio Luccio (M'66–SM'78–F'83) received the Dr.Ing. degree in electrical engineering from the Politecnico di Milano, Italy, in 1962, and the Libera Docenza in electronic computers in 1968.

He started his research activity at Olivetti and the Politecnico di Milano. In 1966, he became Staff Member at M.I.T., Cambridge, MA, on Project MAC, then Professor at the University of Southern California, Los Angelges, CA, and later at New York University, New York, pursuing research in logical network synthesis. In 1971, he returned to Italy as Professor of Informatics at the University of Pisa. He spent sabbatical periods as a Visiting Professor in the U.S., Canada, and Singapore, and was a Visiting Scientist at IBM research in the U.S.., and at NTT Laboratories, Japan. His interests are in algorithm design, and in the relationship between computational models and realistic circuits.

Linda Pagli received the Laurea degree in information sciences from University of Pisa, Italy, in 1973.

She is currently a professor of computer science at the University of Pisa. After receiving her degree, she was associated researcher with the Department of Informatics of the University of Pisa, starting her research activity in data structures and, then, in VLSI computation. In 1987, she was appointed to Professor of Computer Science at the University of Salerno, Italy, returning to Pisa three years later. She has been Visiting Professor at Carleton University, Ottawa, Canada. Her current research interests are in the bases of computation and design and analisys of algorithms. She has pursued intense activities in higher education in favor of developing countries. In this framework, she has been a Professor at the National University of Somalia for an extensive period of time.