Composition of Switching Lattices and Autosymmetric Boolean Function Synthesis

Anna Bernasconi
Dipartimento di Informatica
Università di Pisa, Italy
anna.bernasconi@unipi.it

Valentina Ciriani Luca Frontini Gabriella Trucco
Dipartimento di Informatica
Università degli Studi di Milano, Italy
{valentina.ciriani, luca.frontini, gabriella.trucco}@unimi.it

***Abstract*—Multi-terminal switching lattices are typically exploited for modeling switching nano-crossbar arrays that lead to the design and construction of emerging nanocomputers. In this paper we propose a switching lattice optimization method for a special class of "regular" Boolean functions, called autosymmetric functions. Autosymmetry is a property that is frequent enough within Boolean functions to be interesting in the synthesis process. Each autosymmetric function can be synthesized through a new function (called *restriction*), depending on less variables and with a smaller on-set, which can be computed in polynomial time. In this paper we describe how to exploit the autosymmetry property of a Boolean function in oder to obtain a smaller lattice representation in a reduced minimization time. The original Boolean function can be constructed through a composition of the restriction with some EXORs of subsets of the input variables. Similarly, the lattice implementation of the function can be constructed using some external lattices for the EXORs, whose outputs will input the lattice implementing the restriction. Finally, the output of the restriction lattice corresponds to the output of the original function. Experimental results show that the total area of the obtained lattices is often significantly reduced. Moreover, in many cases, the computational time necessary to minimize the restriction is smaller than the time necessary to perform the lattice synthesis of the entire function.**

I. Introduction

Behind standard CMOS networks representing Boolean functions, the interest on new circuit models has been grown in the recent literature. In this framework, the logic optimization of switching lattices for emerging nanoscale technologies have been proposed and discussed in [4], [23].

A switching lattice is a two-dimensional network of four-terminal switches. Each switch is linked to the four neighbors of a lattice cell, so that these are either all connected or disconnected. A Boolean function can be represented using a switching lattice associating each four-terminal switch to a Boolean literal: if the literal has value 1 the corresponding switch is connected to its four neighbors, otherwise it is not connected. In this model, the Boolean function evaluates to 1 if and only if there exists a connected path between two opposing edges of the lattice, e.g., the top and the bottom edges (see Figure 2 for an example). The synthesis problem on a lattice thus consists in finding an assignment of literals to switches in order to implement a given target function with a lattice of minimal size. The idea of using regular two-dimensional arrays of switches, to implement Boolean functions, is first proposed in a paper by Akers in 1972 [1]. Recently, with the advent of a variety of emerging nanoscale technologies, synthesis methods targeting lattices of multi-terminal switches have found a renewed interest [2], [3], [4], [20], [23].

Previous studies on this subject [8], [9] have shown how the cost of implementing a four-terminal switching lattice could be mitigated by exploiting Boolean function decomposition techniques. The basic idea of this approach is to first decompose a function into some subfunctions, according to a given functional decomposition scheme, and then to implement the decomposed blocks with physically separated regions in a single lattice. Since the decomposed blocks usually correspond to functions depending on fewer variables and/or with a smaller on-set, their synthesis should be more feasible and should produce lattice implementations of smaller size. In the framework of switching lattice synthesis, where the available minimization tools are not yet as developed and mature as those available for CMOS technology, we are interested in reducing the size of the function to be minimized with a preprocessing phase. A smaller input function to a minimization algorithm can imply a smaller area circuit and a reduced synthesis time.

In this paper we study the lattice synthesis of a special class of *regular* Boolean functions, called *Autosymmetric* functions. Thus, the regularity of a function f of n variables is expressed by an *autosymmetry degree* k (with $0 \leq k \leq n$), computed in polynomial time. While the extreme value $k = 0$ means no regularity, 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, depends on $n-k$ variables $(y_1, \ldots, y_{n-k})$ only, and the number of points of f_k is equal to the one of f divided by 2^k. Therefore, the minimization of f_k is naturally easier than that of f. The new variables $y_1, \ldots, y_{n-k}$ are built as EXOR combinations of the original variables, that is $y_i = EXOR(X_i)$, with $X_i \subseteq \{x_1, \ldots, x_n\}$. These EXOR equations are called *reduction equations*.

Although autosymmetric functions form a subset of all possible Boolean functions, a great amount of standard functions of practical interest fall in this class. For istance, the 24% of the functions from a classical collection of benchmarks [25] have at least one non-degenerated autosymmetric output, and their minimization time is critically reduced in the frameworks of SOP and SPP synthesis [10], [11], [12]. While the total number of Boolean functions of n variables is 2^{2^n}, the number of autosymmetric functions is $(2^n-1)2^{2^{n-1}}$, which is much larger than the number of the classical symmetric functions, i.e., the ones invariant under any permutation of their variables, that is 2^{n+1} [10]. Note that an autosymmetric function f depends

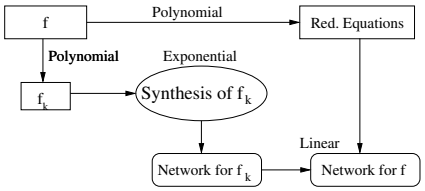

Fig. 1. Synthesis of an autosymmetric function f through the synthesis of its restriction f_k.

in general on all the n input variables, however we are 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 [9] we studied a different kind of lattice decomposition, based on the concept of "D-reducibility". D-reducible functions, similarly to autosymmetric functions, exhibit a regular structure that can be described using the EXOR operation. However, D-reducibility and autosymmetry are different regularities: autosymmetric functions can be studied in a new space whose variables are EXOR combinations of the original ones, while D-reducible functions are studied in a projection space producing an expression where the EXOR gates are in AND with a SOP form. There are examples of autosymmetric functions that are not D-reducible, and of D-reducible functions that are not autosymmetric.

The autosymmetry of a function f can be exploited in the minimization process, according to the strategy shown in Figure 1. First detect the autosymmetry degree k of f. If $k > 0$, derive the restriction f_k of f and the corresponding reduction equations. Second, minimize f_k with any standard method: two level logic as SOP [17], Reed Muller [24]; three-level logic as SPP [5], [14], [21] (OR of ANDs of EXORs), EXSOP [18], [19] (EXOR of ORs of ANDs), or switching lattices, as proposed in this paper. We note that, in the worst case, the lattice minimization requires time exponential in the number of points of the function, however, this number is strongly reduced for f_k if compared to f. We can finally construct a lattice for f from the one of f_k and from the reduction equations, whose computation requires some EXORs. This approach is convenient because: (i) f_k can be computed more efficiently; (ii) the number of points of f_k is $|f|/2^k$ and f_k depends only on $n-k$ variables; (iii) the lattice minimization of f_k is naturally easier than that of f; and (iv) the number of EXORs that we add to the logic is at most $2(n-k)$. On the other hand, we require a second lattice containing the EXORs whose outputs are the input variables (i.e., $y_1, \ldots, y_{n-k}$) of the lattice for f_k. However, the reduction equations are in general EXORs of a very reduced number of variables and their lattices implementations have limited size. Experimental results show that by applying the proposed method we obtain more compact lattices and, in many cases, the computational time necessary to minimize the restriction is smaller than the time necessary to perform the lattice synthesis of the entire function.

The paper is organized as follows. Preliminaries on switching lattices and autosymmetric Boolean functions are described in Section II. Section III discusses the problem of lattice composition, while Section IV discusses the lattice implementation of autosymmetric functions. Section V provides the experimental results and Section VI concludes the paper.

II. Preliminaries

In this section we briefly review some basic notions and results on switching lattices [1], [4], [20] and autosymmetric Boolean functions [6], [7], [10], [11], [12], [13], [21].

A. Switching Lattices

A switching lattice is a two-dimensional array of four-terminal switches. The four terminals of the switch link to the four neightbours of a lattice cell, so that these are either all connected (when the switch is ON), or disconnected (when the switch is OFF).

A Boolean function can be implemented by a lattice in terms of connectivity across it:

- each four-terminal switch is controlled by a Boolean literal;
- each switch may be also labelled with the constant 0, or 1;
- if the literal takes the value 1, the corresponding switch is connected to its four neightbours, else it is not connected;
- the function evaluates to 1 if and only if there exists a connected path between two opposing edges of the lattice, e.g., the top and the bottom edges;
- input assignments that leave the edges unconnected correspond to output 0.

For instance, the 3×3 network of switches in Figure 2 (a) corresponds to the lattice form depicted in Figure 2 (b), which implements the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_2 + x_2x_3$. If we assign the values 1, 1, 0 to the variables x_1, x_2, x_3, respectively, we obtain paths of gray square connecting the top and the bottom edges of the lattices (Figure 2 (c)), indeed on this assignment f evaluates to 1. On the contrary, the assignment $x_1 = 0, x_2 = 0, x_3 = 1$, on which f evaluates to 0, does not produce any path from the top to the bottom edge (Figure 2 (d)).

The synthesis problem on a lattice consists in finding an assignment of literals to switches in order to implement a given target function with a lattice of minimal size. The size is measured in terms of the number of switches in the lattice.

A switching lattice can similarly be equipped with left edge to right edge connectivity, so that a single lattice can implement two different functions. This fact is exploited in [3], [4] where the authors propose a synthesis method for switching lattices simultaneously implementing a function f according to the connectivity between the top and the bottom plates, and its dual function f^D according to the connectivity between the left and the right plates. Recall that the dual of a Boolean function f depending on n binary variables is the function f^D such that $f(x_1, x_2, \ldots, x_n) = \overline{f^D}(\overline{x}_1, \overline{x}_2, \ldots, \overline{x}_n)$. This method produces lattices with a size that grows linearly with the number of products in an irredundant sum of product (SOP) representation of f, and consists of the following steps:

1) find an irredundant, or a minimal, SOP representation for f and f^D: $SOP(f) = p_1 + p_2 + \ldots p_s$ and $SOP(f^D) = q_1 + q_2 + \ldots q_r$;

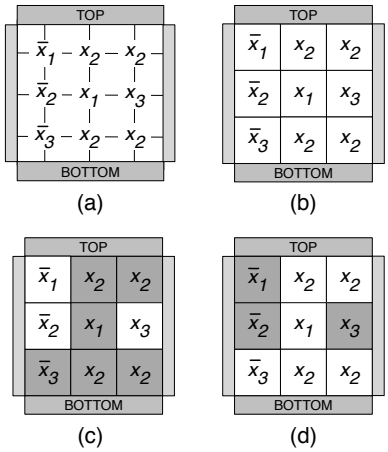

Fig. 2. A four terminal switching network implementing the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_2 + x_2x_3$ (a); its corresponding lattice form (b); the lattice evaluated on the assignments 1,1,0 (c) and 0, 0, 1 (d), with grey and white squares representing ON and OFF switches, respectively.

2) form a $r \times s$ switching lattice and assign each product p_j $(1 \leq j \leq s)$ of $SOP(f)$ to a column and each product q_i $(1 \leq i \leq r)$ of $SOP(f^D)$ to a row;
3) for all $1 \leq i \leq r$ and all $1 \leq j \leq s$, assign to the switch on the lattice site (i, j) one literal which is shared by q_i and p_j (the fact that f and f^D are duals guarantees that such a shared literal exists for all i and j).

This synthesis algorithm thus produces a lattice for f whose size depends on the number of products in the irredundant SOP representations of f and f^D, and it comes with the dual function implemented for free. For instance, the lattice depicted in Figure 2 has been built according to this algorithm, and it implements both the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_2 + x_2x_3$ and its dual $f^D = x_1\overline{x}_2x_3 + \overline{x}_1x_2 + x_2\overline{x}_3$.

The time complexity of the algorithm is polynomial in the number of products. However, the method does not always build lattices of minimal size for every target function, since it ties the dimensions of the lattices to the number of products in the SOP forms. In particular this method is not effective for Boolean functions whose duals have a a very large number of products. Another reason that could explain the non-minimality of the lattices produced in this way is that the algorithm does not use Boolean constants as input, i.e., each switch in the lattice is always controlled by a Boolean literal.

In [20], the authors have proposed a different approach to the synthesis of minimal-sized lattices, which is formulated as a satisfiability problem in quantified Boolean logic and solved by quantified Boolean formula solvers. This method uses the previous algorithm to find an upper bound on the dimensions of the lattice. It then searches for successively better implementations until either an optimal solution is found, or a preset time limit has been exhausted. Experimental results show how this alternative method can decrease lattice sizes considerably. In this approach the use of fixed inputs (i.e., constant values 0 and 1) is allowed.

B. Autosymmetric Boolean functions

In this section we briefly review autosymmetric functions that are introduced in [21] and further studied in [6], [7], [10], [11], [12], [13]. For the description of these particular regular functions we need to summarize several concepts of Boolean algebra [16].

Given two binary vectors $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, let $\boldsymbol{\alpha} \oplus \boldsymbol{\beta}$ be the elementwise EXOR between $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$, for example $11010 \oplus 11000 = 00010$. We recall that $(\{0,1\}^n, \oplus)$ is a vector space, and that a *vector subspace* V is a subset of $\{0,1\}^n$ containing the zero vector $\mathbf{0}$, such that for each $\boldsymbol{v}_1$ and $\boldsymbol{v}_2$ in V we have that $\boldsymbol{v}_1 \oplus \boldsymbol{v}_2 \in V$. The vector subspace V contains 2^k vectors, where k is the *dimension* of V, and it is generated by a basis B containing k vectors. Indeed B is a minimal set of vectors of V such that each point of V is an EXOR combination of some vectors in B.

Let us consider a completely specified Boolean function $f : \{0,1\}^n \rightarrow \{0,1\}$, recalling that f can be described as the set of binary vectors in $\{0,1\}^n$ for which f takes the value 1 (i.e., the ON-set of f). Using this notation we can give the following definition. The function f is *closed under* a vector $\boldsymbol{\alpha} \in \{0,1\}^n$, if for each vector $\boldsymbol{w} \in \{0,1\}^n$, $\boldsymbol{w} \oplus \boldsymbol{\alpha} \in f$ if and only if $\boldsymbol{w} \in f$.

For example, the function $f = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1011, 1101, 1110\}$ is closed under $\boldsymbol{\alpha} = 0011$, as it can be easily verified.

It is easy to observe that any function f is closed under the zero vector $\mathbf{0}$. Moreover, 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 $\boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2$. Therefore, the set $L_f = \{\boldsymbol{\beta}$: f is closed under $\boldsymbol{\beta}\}$ is a vector subspace of $(\{0,1\}^n, \oplus)$. The set L_f is called the *vector space* of f. For instance, the function f of our previous example is closed under the vectors in the vector space $L_f = \{0000, 0011, 0101, 0110\}$.

For an arbitrary function f, the vector space L_f provides the essential information for the definition of the autosymmetry property:

Definition 1 ([11]): A completely specified Boolean function f is *k-autosymmetric*, or equivalently f has *autosymmetry degree* k, $0 \leq k \leq n$, if its vector space L_f has dimension k.

In general, f is *autosymmetric* if its autosymmetry degree is $k \geq 1$. For instance, the function f of our running example is 2-autosymmetric since its vector space L_f has dimension 2.

We now define a special basis, called canonical, to represent L_f. Consider a $2^k \times n$ matrix M whose rows correspond to the points of a vector space V of dimension k, and whose columns correspond to the variables $x_1, x_2, \ldots, x_n$. Let the row indices of M be numbered from 0 to $2^k - 1$. We say that V is in *binary order* if the rows of M are sorted as increasing binary numbers. We have:

Definition 2 ([11]): Let V be a vector space of dimension k in binary order. The *canonical basis* B_V of V is the set of points corresponding to the rows of M with indices $2^0, 2^1, \ldots, 2^{k-1}$. The variables corresponding to the first 1 from the left of each row of the canonical basis are the *canonical variables* of V, while the other variables are *non-canonical*.

It can be easily proved that the canonical basis is indeed a vector basis [15]. The canonical variables of L_f are also called canonical variables of f.

Example 1: Consider the vector space L_f of the function f of our running example. We can arrange its vectors in a matrix in binary order:

	x_1	x_2	x_3	x_4
0	0	0	0	0
1	0	0	1	1
2	0	1	0	1
3	0	1	1	0

The canonical basis is composed of the vectors in position **1** and **2**, that are the vectors 0011 and 0101. The canonical variables of f are x_2 (corresponding to the first 1 in 0101) and x_3 (corresponding to the first 1 in 0011). The remaining variables x_1 and x_4 are non-canonical.

For a vector $\boldsymbol{\alpha} \in \{0,1\}^n$ and a subset $S \subseteq \{0,1\}^n$, consider the set $\boldsymbol{\alpha} \oplus S = \{\boldsymbol{\alpha} \oplus \boldsymbol{s} \mid \boldsymbol{s} \in S\}$. In a sense, the vector $\boldsymbol{\alpha}$ is used to "translate" the subset S. If the set S is a vector space, then its "translations" are called *affine spaces*:

Definition 3: Let V be a vector subspace of $(\{0,1\}^n, \oplus)$. The set $A = \boldsymbol{\alpha} \oplus V$, $\boldsymbol{\alpha} \in \{0,1\}^n$, is an *affine space* over V with *translation point* $\boldsymbol{\alpha}$.

Note that $\boldsymbol{\alpha} \in A$, because S contains the zero vector $\mathbf{0}$, hence $\boldsymbol{\alpha} = \boldsymbol{\alpha} \oplus \mathbf{0} \in A$. Moreover, any other vector of A could be chosen as translation point $\boldsymbol{\alpha}$, thus generating the same affine space.

There is a simple formula that characterizes the vector space associated to a given affine space A, namely [16]:

$$V = \boldsymbol{\alpha} \oplus A, \text{ with } \boldsymbol{\alpha} \text{ any point in } A.$$

That is, given an affine space A there exists a unique vector space V such that $A = \boldsymbol{\alpha} \oplus V$, where $\boldsymbol{\alpha}$ is any point of A.

As proved in [6], the points of a k-autosymmetric function f can be partitioned into $\ell = |f|/2^k$ disjoint sets, where $|f|$ denotes the number of points of f; all these sets are affine spaces over L_f. I.e., $S = \boldsymbol{\alpha} \oplus L_f$, where S is any such a space and $\boldsymbol{\alpha} \in f$. Thus:

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

and for each i, j, $i \neq j$, $(\boldsymbol{\alpha}^i \oplus L_f) \cap (\boldsymbol{\alpha}^j \oplus L_f) = \emptyset$. The vectors $\boldsymbol{\alpha}^1, \ldots, \boldsymbol{\alpha}^\ell$ are chosen as all the points of f where all the canonical variables have value 0.

Example 2: Consider the function $f = \{0000, 0001, 0010, 0011, 0100, 0101, 0110, 0111, 1000, 1011, 1101, 1110\}$ of our running example. By Example 1 the canonical variables of f are x_2 and x_3. Thus, if we take the points of f with all canonical variables set to 0, i.e., $\boldsymbol{\alpha}^1 = 0000$, $\boldsymbol{\alpha}^2 = 0001$, and $\boldsymbol{\alpha}^{\,1} = 1000$, we have

$$f = (0000 \oplus L_f) \cup (0001 \oplus L_f) \cup (1000 \oplus L_f),$$

where $L_f = \{0000, 0011, 0101, 0110\}$.

Autosymmetric functions can be reduced to "equivalent, but smaller" functions; in fact, if a function f is k-autosymmetric, then there exists a function f_k over $n-k$ variables, y_1, y_2, …, y_{n-k}, such that

$$f(x_1, \ldots, x_n) = f_k(y_1, \ldots, y_{n-k}),$$

where each y_i is an EXOR combination of a subset of x_i's. These combinations are denoted $EXOR(X_i)$, where $X_i \subseteq X$, and the equations $y_i = EXOR(X_i)$, $i = 1, \ldots, n-k$, are called *reduction equations*. The function f_k is called a *restriction* of f; indeed f_k is "equivalent" to, but smaller than f, and has $|f|/2^k$ points only.

The restriction f_k can be computed from f and its vector space L_f by first identifying the canonical variables, and then deriving the cofactor of f where all the canonical variables are set to 0 (see [6] and [11] for more details). The reduction equations correspond to the homogeneous system of linear equations whose solutions define the vector space L_f, and they can be derived applying standard linear algebra techniques as shown in [6], [11].

Example 3: Consider the 2-autosymmetric function f in our running example, with $L_f = \{0000, 0011, 0101, 0110\}$ and canonical variables x_2 and x_3. We can build f_2 by taking the cofactor $f_{x_2=0, x_3=0} = \{00, 01, 10\}$, that contains only 3 points and corresponds to the function $f_2(y_1, y_2) = \overline{y_1 y_2}$. The homogeneous system whose solutions are $\{0000, 0011, 0101, 0110\}$ is:

$$\begin{cases} x_1 = 0 \\ x_2 \oplus x_3 \oplus x_4 = 0 \end{cases}$$

Thus the reduction equations are given by

$$y_1 = x_1 \quad (1)$$

$$y_2 = x_2 \oplus x_3 \oplus x_4\,. \quad (2)$$

Finally, the function f can be represented as:

$$f(x_1, x_2, x_3, x_4) = f_2(y_1, y_2) = \overline{y_1 y_2} = \overline{x}_1(\overline{x_2 \oplus x_3 \oplus x_4})\,.$$

We can note that f is indeed a composition of f_2 and the reduction equations (1) and (2).

III. Lattice composition

First of all, we recall from [20] that given the switching lattices implementing two functions f and g, we can easily construct the lattices representing their disjunction $f + g$ and their conjunction $f \cdot g$ composing the two lattices for f and g and using a padding column of 0s and a padding row of 1s, respectively, as shown in Figure 3. In particular, for the disjunction, the column of 0s separates all top-to-bottom paths in the lattices for f and g, so that the accepting paths for the two functions never intersect; this, in turn, implies that there exists a top-to-bottom connected path in the lattice for $f + g$ if and only if there is at least one connected path for f or for g. If the lattices for f and g have a different number of rows, we add some rows of 1s to the lattice with fewer rows, so that each accepting path can reach the bottom edge. Similarly, for the conjunction the padding row of 1s allows to join any top-to-bottom accepting path for the function f with any top-to-bottom accepting path for g, so that the overall lattice evaluates to 1 if and only if both f and g evaluate to 1.

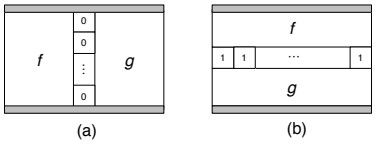

Fig. 3. Lattice implementation of $f \vee g$ (a) and of $f \wedge g$ (b).

As before, if the lattices for f and g have a different number of columns, we add some columns of 0s to the lattice with fewer columns, so that an accepting path for one of the two functions can never reach the opposite edge of the lattice if the other function evaluates to 0.

More in general, these simple composition rules can be used to implement a switching lattice for a function f starting from a decomposition of f into subfunctions. The basic idea of this approach is to first decompose f according to a given functional decomposition scheme, then generate the lattices for each component function, and finally implement the original function by a single composed lattice obtained by gluing together appropriately the lattices of the component functions. Previous studies on this subject [8], [9] demonstrated that lattice synthesis benefits from this decomposition-based approach: since the decomposed blocks usually correspond to functions depending on fewer variables and/or with a smaller on-set, their synthesis should produce lattice implementations of smaller size, yielding an overall lattice of smaller dimension in an affordable computation time.

In all these examples, from the simple cases of $f + g$ and $f \cdot g$, to the decomposition schemes described in [8], [9], the lattice for the original function has been obtained implementing the decomposed blocks with physically separated regions in a single overall lattice. We will refer to this approach as *internal composition*.

However, there are situations where this kind of internal composition cannot be directly applied. For instance, consider a function f depending on n binary variables defined as

$$f(x_1, \ldots, x_n) = g(y_1, \ldots, y_k)\,,$$

where *(i)* g is a Boolean function depending on $k < n$ variables; *(ii)* for any i, $y_i = h_i(S_i)$, $S_i \subseteq \{x_1, \ldots, x_n\}$, and h_i is a Boolean function depending on $|S_i|$ variables. Ideally, we would like to derive a lattice implementation for f substituting in a lattice implementation for g each occurrence of a variable y_j with a lattice implementation of the corresponding function h_j. This task, however, requires some care to be performed correctly.

Consider a very simple case: $f(x_1, x_2, x_3, x_4) = (x_1 \oplus x_2)(x_3 \oplus x_4)$, that can be seen as a functional composition $f = g(y_1, y_2)$ where g is simply a conjunction of two variables, and y_1 and y_2 are EXORs of two variables. Then, we can build a lattice for f (Figure 4(c)) starting from the very simple 2×1 lattice representation of g (Figure 4(a)), and substituting y_1 and y_2 with the lattice representations of $(x_1 \oplus x_2)$ and $(x_3 \oplus x_4)$, which are shown in Figure 4(b). Note that we need to insert a row of 1s between the two sublattices, so that we can extend any accepting path in the sublattice on top, with any accepting path in the bottom sublattice. The overall lattice for f has size

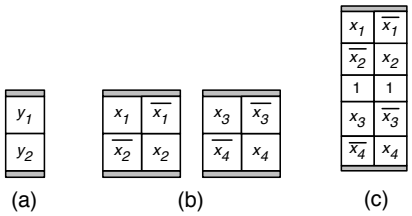

Fig. 4. (a) Lattice implementation of $g = y_1y_2$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$; (c) final lattice for $f = (x_1 \oplus x_2)(x_3 \oplus x_4)$

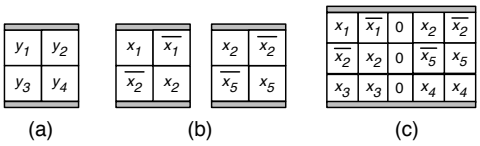

Fig. 5. (a) Lattice implementation of $g = y_1y_3 + y_2y_4$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_2 \oplus x_5$; (c) final lattice for $f(x_1, x_2, x_3, x_4, x_5) = (x_1 \oplus x_2)x_3 + (x_2 \oplus x_5)x_4$.

5×2. Notice that using the lattice synthesis method presented in [4] directly on f, we would get a lattice of size 4×4.

Now, consider the function $f(x_1, x_2, x_3, x_4, x_5) = (x_1 \oplus x_2)x_3 + (x_2 \oplus x_5)x_4$. Given a lattice for the function $g = y_1y_3 + y_2y_4$, we could try to build a lattice for f by simply substituting the occurrences of y_1 and y_2 with sublattices for $(x_1 \oplus x_2)$ and $(x_2 \oplus x_5)$, and the occurrences of y_3 and y_4 with x_3 and x_4, respectively. Note that we need to duplicate some variables in order to get a rectangular lattice, besides inserting a padding column of 0, as shown in Figure 5. Indeed, without the padding column, the lattice would contain a top-to-bottom path on the assignment 11100, whereas $f(1, 1, 1, 0, 0) = 0$.

As a final example, let us consider the parity function of 4 variables, $f(x_1, x_2, x_3, x_4) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$, that can be interpreted as $f = g(y_1, y_2) = y_1 \oplus y_2$, where $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$. If we derive a lattice for f using this decomposition, we need to appropriately insert padding rows and columns as depicted in Figure 6: the padding rows of 1s are needed to join the accepting paths in the sublattices on top, implementing y_1 and $\overline{y}_1$ with the accepting paths in the bottom sublattices for y_2 and $\overline{y}_2$; while the column of 0s is needed to avoid intersections between accepting paths on the left and on the right side of the overall lattices. Without the column of 0s, the lattice in Figura 6 would contain a top-to-bottom path e.g., on the input assignment 0110. With the padding rows and columns, the size of the overall lattice becomes 5×5, that is not competitive with the size of an optimal lattice for the parity of 4 variables, which is 3×5 [22].

A possible strategy to overcome some of these problems could be a different lattice composition technique[1], that we could call *external composition*. The idea is simply to use multiple nanoarrays, i.e., multiple lattices and to connect the output of a lattice with one or more literals occurring in another lattice as depicted in Figure 7. Observe that the overall lattice

[1]M. Altun and M. C. Morgul, *personal communication*, 2017

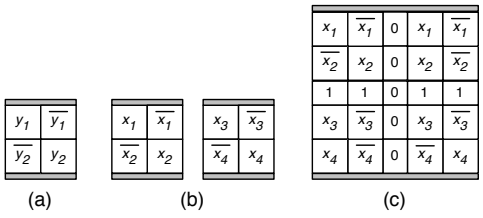

Fig. 6. (a) Lattice implementation of $g = y_1 \oplus y_2$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$; (c) final lattice for $f(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

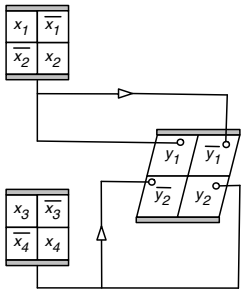

Fig. 7. Multiple lattice implementation of $f(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

composition in this picture implements the parity function of 4 variables as a 2×2 lattice representing $g = y_1 \oplus y_2$, connected to two external lattice implementations for $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$. In this way, we get a multiple lattice implementation of overall size 12, smaller than an optimal standard lattice for the parity of four variables, whose size is 15 [22]. As this simple example clearly shows, multiple lattices allow to reduce the number of switches and thus the overall dimension of the lattice. However, the gain in the dimension comes at the expense of an increase in the interconnection cost.

IV. Lattice representation of autosymmetric functions

The lattice implementation of autosymmetric functions can be derived exploiting the external lattice composition discussed in the previous Session III. Recall from Section II-B, that a k-autosymmetric function f can be decomposed as

$$f(x_1, \ldots, x_n) = f_k(y_1, \ldots, y_{n-k}),$$

where *(i)* the *restriction* f_k depends on $n-k$ binary variables, and has $|f|/2^k$ points only; and *(ii)* each y_i is defined by a *reduction equation*, i.e., an EXOR of a subset of the original variables x_i's: $y_i = EXOR(X_i)$, $X_i \subseteq \{x_1, \ldots, x_n\}$. Therefore, we can build a multiple lattice implementing f composing a lattice $L(f_k)$ for the restriction f_k with $n-k$ sublattices representing the reduction equations: for each i, $1 \leq i \leq n-k$, the output of the sublattice $L(y_i)$ implementing y_i is connected, possible through an inverter, to all occurrences of the literal y_i in $L(f_k)$ (see Figure 8). Of course, for all variables y_j whose associated reduction equation is a single variable, e.g., x_t, there is no need to connect the switch to an external lattice, but just to x_t.

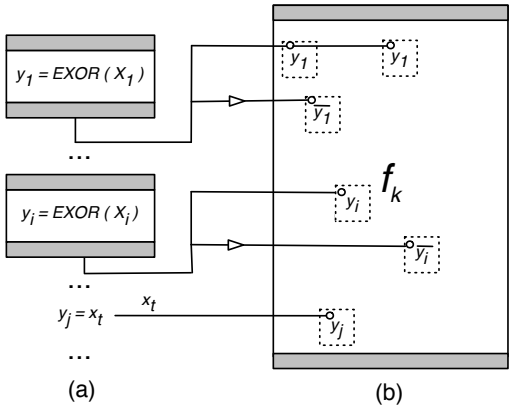

Fig. 8. Multiple lattice implementation of an autosymmetric function: (a) lattice implementation of the reduction equations; (b) lattice implementation of the restriction f_k.

Since f_k depends on fewer variables, and has a smaller on-set with respect to f, its lattice synthesis should be an easier task, and should produce a lattice of reduced size. Notice that, to further reduce the total area, one could also apply the decomposition methods presented in [8] and [9] to the lattice for f_k. Moreover, the reduction equations are in general EXORs of a very reduced number of variables and their lattices implementations have limited size. For these reasons, the overall multiple lattice representing f should be smaller that a standard lattice for f, derived with the synthesis methods presented in [4] and in [20], and possibly even smaller than an optimal size lattice for f. This expectation has been confirmed by our experimental results.

V. Experimental results

In this section we report the experimental results obtained applying the multiple lattice implementation of autosymmetric functions described in Section IV. Since a k-autosymmetric function $f_k(y_1, \ldots, y_{n-k})$ depends on fewer variables w.r.t. the corresponding original function $f(x_1, \ldots, x_n)$, our aim is to obtain lattices of reduced size.

The algorithms have been implemented in C. The experiments have been run on a machine with two AMD Opteron 4274HE for a total of 16 CPUs at 2.5 GHz and 128 GByte of main memory, running Linux CentOS 6.6. The benchmarks are taken from LGSynth93 [25]. We considered each output as a separate Boolean function, for a total of 607 functions, including 53 autosymmetric functions on which we applied the lattice implementation described in the previous section.

To evaluate the utility of our approach, in Table I we compare the lattice synthesis results obtained applying the decomposition scheme based on autosymmetry, with the results obtained with the standard synthesis methods presented in [4] and in [20], without exploiting the autosymmetry property. To simulate the results reported in [20], we used a collection of Python scripts for computing minimum-area switching lattices, by transformation to a series of SAT problems.

Each row of the table lists the results for each separate autosymmetric output function of the benchmark circuit. More

TABLE I. PROPOSED LATTICE SIZES FOR STANDARD BENCHMARK CIRCUITS: A COMPARISON OF THE PROPOSED METHOD WITH THE RESULTS PRESENTED IN [4] AND IN [20]. WHEN THE SYNTHESIS OF A LATTICE IS STOPPED, THERE IS NO LATTICE (−). RESULTS ARE MARKED WITH ⋆ WHEN SAT IS STOPPED.

	# y_i	[4]							[20]									
		Std. Synthesis		Decomposed Synthesis					Std. Synthesis			Decomposed Synthesis						
		col×row	Area	f_k col×row	f_k Area	XOR Area	tot. Area	inv.	col×row	Area	time	f_k col×row	f_k Area	f_k Time	XOR Area	XOR Time	tot. Area	tot. time	inv.
add6(0)	1	2×2	**4**	1×1	1	4	5	0	2×2	**4**	**0.021**	1×1	1	0.028	4	0.024	5	0.052	0
add6(1)	1	6×6	36	3×3	9	4	**14**	1	5×3	15	2.561	3×3	9	0.019	4	0.028	**14**	**0.047**	1
add6(2)	1	16×16	256	8×8	64	4	**69**	1	-	-	-	5×3	15	124	4	0.026	**20**	0.026	1
add6(3)	1	36×36	1296	18×18	324	4	**329**	1	-	-	-	-	-	-	4	0.029	**329**⋆	0.029	1
add6(4)	1	76×76	5776	38×38	1444	4	**1449**	1	-	-	-	-	-	-	4	0.035	**1449**⋆	0.035	1
add6(5)	1	156×156	24336	78×78	6084	4	**6089**	1	-	-	-	-	-	-	4	0.026	**6089**⋆	0.26	1
adr4(1)	1	15×23	345	18×18	324	4	**329**	1	-	-	-	-	-	-	4	0.025	**329**⋆	0.025	1
adr4(2)	1	36×36	1296	8×8	64	4	**69**	1	-	-	-	-	-	-	4	0.027	**20**⋆	0.027	1
adr4(3)	1	16×16	256	3×3	9	4	**14**	1	5×3	15	2.57	3×3	9	0.019	4	0.026	**14**	**0.045**	1
adr4(4)	1	6×6	36	1×1	1	4	**5**	0	-	-	-	1×1	1	0.026	4	0.031	**5**	0.026	0
al2(11)	1	5×25	125	3×20	60	4	**64**	0	-	-	-	-	-	-	4	0.021	**64**⋆	0.057	0
alcom(5)	1	4×3	12	3×2	6	4	**10**	0	4×3	12	**0.028**	3×2	6	0.025	4	0.3	**10**	0.325	0
b11(5)	1	2×3	**6**	1×2	2	4	**6**	0	2×3	**6**	**0.023**	1×2	2	0.025	4	0.025	**6**	0.05	0
b12(6)	1	9×6	54	7×5	35	4	**39**	0	4×5	**20**	**918**	4×5	20	1350	4	0.027	24	1350	0
dekoder(0)	1	4×2	8	3×1	3	4	**7**	0	4×2	8	**0.027**	3×1	3	0.024	4	0.022	**7**	0.046	0
dekoder(1)	1	3×2	**6**	2×1	2	4	**6**	0	3×2	**6**	**0.024**	2×1	2	0.023	4	0.023	**6**	0.046	0
dk27(8)	1	1×1	**1**	1×1	1	4	5	0	1×1	**1**	**0.021**	1×1	1	0.026	4	0.024	5	0.05	0
exps(18)	3	2×8	**16**	1×7	7	12	19	0	2×6	**12**	22.2	1×7	7	0.027	12	0.067	19	**0.094**	0
exps(19)	6	2×8	**16**	1×7	7	24	31	0	2×8	**16**	**0.03**	1×7	7	0.024	24	0.137	31	0.161	0
f51m(6)	1	2×2	**4**	1×1	1	4	5	0	2×2	**4**	**0.03**	1×1	1	0.025	4	0.026	5	0.051	0
luc(3)	1	4×4	16	3×3	9	4	**13**	0	4×3	**12**	0.861	3×3	9	0.024	4	0.031	13	**0.055**	0
m1(8)	1	3×5	15	2×4	8	4	**12**	0	3×4	**12**	1.65	2×4	8	0.333	4	0.024	**12**	**0.357**	0
max1024(0)	8	4×14	56	2×7	14	32	**46**	0	-	-	-	2×7	14	1.16	32	0.202	**46**	116	0
max1024(1)	8	18×18	324	9×9	81	32	**119**	6	18×18	324	**0.034**	6×4	24	2543	32	0.198	**63**	2543	7
max1024(2)	9	38×32	1216	19×16	304	36	**348**	8	-	-	-	19×16	304	0.036	36	0.23	**348**	**0.266**	8
max1024(3)	9	64×48	3072	32×25	800	36	**844**	8	64×48	3072	**0.133**	-	-	-	36	0.22	**844**⋆	0.22	8
max1024(4)	9	82×83	6806	43×46	1978	36	**2023**	9	82×83	6806	**0.23**	43×46	1978	0.07	36	0.218	**2023**	0.288	9
max1024(5)	9	122×117	14274	62×64	3968	36	**4013**	9	-	-	-	-	-	-	36	0.221	**4013**⋆	0.221	9
newcond(1)	2	2×4	**8**	1×3	3	8	11	0	2×4	**8**	0.244	1×3	3	0.023	8	0.051	11	**0.074**	0
newcwp(0)	2	5×4	20	3×2	6	8	**15**	1	4×3	**12**	0.521	3×2	6	0.023	8	0.053	15	**0.076**	1
newcwp(1)	1	4×4	16	1×1	1	12	**13**	0	3×3	**9**	**0.303**	1×1	1	0.025	9	0.281	10	0.306	0
newcwp(3)	2	2×2	**4**	1×1	1	4	5	0	2×2	**4**	**0.022**	1×1	1	0.021	4	0.027	5	0.048	0
p82(10)	2	2×5	10	1×4	4	4	**8**	0	2×4	**8**	0.106	1×4	4	0.021	4	0.025	**8**	**0.046**	0
pope.rom(18)	3	2×6	**12**	1×5	5	12	17	0	2×5	**10**	1.45	1×5	5	0.025	12	0.07	17	**0.095**	0
pope.rom(32)	1	2×4	8	1×3	3	4	**7**	0	2×3	**6**	0.086	1×3	3	0.027	4	0.028	7	**0.055**	0
pope.rom(34)	1	4×2	8	3×1	3	4	**7**	0	1×4	8	1806	3×1	3	0.023	4	0.019	**7**	**0.042**	0
pope.rom(35)	1	2×3	**6**	1×2	2	8	10	0	2×3	**6**	**0.077**	1×2	2	0.025	8	0.05	10	**0.075**	0
pope.rom(41)	1	5×2	10	4×1	4	4	**8**	0	5×2	10	**0.022**	4×1	4	0.031	4	0.023	**8**	0.054	0
radd(0)	1	2×2	**4**	1×1	1	4	5	0	2×2	**4**	**0.025**	1×1	1	0.021	4	0.026	5	0.047	0
radd(1)	1	6×6	36	3×3	9	4	**14**	1	3×5	15	2.45	3×3	9	0.026	4	0.027	**14**	**0.053**	1
radd(2)	1	16×16	256	8×8	64	4	**69**	1	-	-	-	5×3	15	122.1	4	0.025	**20**	122	1
radd(3)	1	36×36	1296	18×18	324	4	**329**	1	-	-	-	-	-	-	4	0.024	**329**⋆	0.024	1
rd53(1)	4	10×10	100	6×5	30	16	**50**	4	-	-	-	4×3	12	0.676	16	0.118	**32**	0.794	4
rd53(2)	1	16×16	256	1×1	1	80	**81**	0	-	-	-	1×1	1	0.022	-	-	**81**⋆	0.022	0
rd73(2)	1	35×35	1225	1×1	1	448	**449**	0	-	-	-	1×1	1	0.023	-	-	**449**⋆	0.023	0
risc(4)	1	2×3	**6**	1×2	2	4	**6**	0	2×3	**6**	**0.026**	1×2	2	0.027	4	0.02	**6**	0.047	0
sqn(0)	2	17×16	272	7×7	49	8	**58**	1	-	-	-	3×5	15	11	8	0.049	**24**	11.5	1
wim(2)	1	3×2	**6**	2×1	2	4	**6**	0	3×2	**6**	**0.026**	2×1	2	0.023	4	0.019	**6**	0.042	0
z4(1)	5	28×28	784	8×8	64	28	**97**	5	28×28	784	**11.34**	5×3	15	115	25	0.412	**45**	115	5
z4(2)	3	12×12	144	3×3	9	20	**32**	3	-	-	-	3×3	9	0.021	17	0.341	**29**	0.362	3
z4(3)	1	4×4	16	1×1	1	12	**13**	0	3×3	**9**	**0.29**	1×1	1	0.027	9	0.286	10	0.313	0
z5xp1(8)	1	2×2	**4**	1×1	1	4	5	0	2×2	**4**	**0.027**	1×1	1	0.019	4	0.026	5	0.045	0
z9sym(0)	8	86×72	6192	56×36	2016	32	**2056**	8	-	-	-	-	-	-	32	0.217	**2056**⋆	0.276	8

precisely, the first column reports the name and the number of the considered output of each instance; the second column reports the number of EXOR lattices used to implement the reduction equations (y_j) when the decomposition method is applied. The following seven columns refer to the synthesis of lattices as described in [4], with (columns 5-9) and without (columns 3-4) the multiple lattice decomposition based on autosymmetry. In particular, columns 3 and 4 show dimension ($X \times Y$) and area ($Area = X \cdot Y$) of lattices derived applying the standard synthesis method (i.e., without exploiting the autosymmetry property), columns 5 and 6 show the dimension and the area of the lattice for the restriction f_k, column 7 shows the total area of the lattices for the EXOR terms y_i, column 8 shows the total area occupied by lattices ($TotalArea = A_{f_k} + \sum_i A_{y_i} + num.inv$), and column 9 indicates the number $num.inv$ of inverter necessary to make the signal routing. The synthesis in [4] is performed using ESPRESSO, and in all cases it takes less than 0.01 s, that is the minimum time resolution of the synthesizer; for this reason the synthesis time is omitted.

Columns 10 to 20 refer to lattices synthesized using the methodology presented in [20], with and without decomposition on multiple lattices. In particular columns 10, 11, and 12 report the dimension, the area and the synthesis time of lattices obtained with standard synthesis; columns 13, 14 and 15 report the dimension, the area and the synthesis time of the lattice for the restriction f_k, column 16 and 17 report the total area of the lattices for the EXOR terms y_i and their synthesis time; columns 18 and 19 show the total area occupied by lattices ($TotalArea = A_{f_k} + \sum_i A_{y_i} + num.inv$) and the total synthesis time; finally, column 20 indicates the number

$num.inv$ of inverter necessary for signal routing.

For each function, we bolded the best areas (col. 4 vs col. 8 and col. 11 vs col. 18) and the best simulation time. In some cases the method proposed in [20] fails in computing a result in reasonable run time. For this reason, we set a time limit (equal to ten minutes) for each SAT execution; if we do not find a solution within the time limit, the synthesis is stopped. We marked with $-$ all cases where the synthesis of lattices has been stopped. In the synthesis of sublattices, whenever [20] is stopped, we use the sublattices synthesized with [4], because without a sublattice it would be impossible to complete the synthesis of the overall decomposed lattice. We marked these cases with $\star$. Note that, for many benchmarks, the method in [20] did not find a solution within the fixed time limit for at least one sublattice, and had to be replaced with [4].

The results are promising. Considering the methodology presented in [4], we obtain a smaller total area w.r.t. the standard synthesized lattices in 58% of the benchmarks, with an average gain of 53%. Considering the methodology presented in [20], we obtain a smaller total area in 48% of the benchmarks, with an average gain of 60%. Note that in many cases the synthesis time necessary to decompose the function as described in this paper (column 19) is smaller than the time necessary to perform the standard synthesis (column 12).

Comparing the results of different types of decomposition w.r.t. standard synthesized lattices, [8] obtains smaller area in about 33% of cases, with an average gain of 24%, [9] obtains smaller area in about 15% of cases, with an average gain of 24%. Note that the decomposition in [8] can be applied to all the boolean functions and the decomposition in [9] only to the subset of D-reducible boolean functions.

VI. Conclusions

In this paper we have shown a lattice minimization strategy for autosymmetric functions. We have described how to exploit an external composition for autosymmetric functions in order to get compact area representation with switching lattices. The experimental results have validated the approach.

As future work, it would be interesting to study other classes of regular functions. Another interesting future direction is the study of a different strategy to compose the switching lattices in order to obtain more compact solutions.

VII. Acknowledgments

This project has received funding from the European Union Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie grant agreement No 691178.

References

[1] S. B. Akers, "A Rectangular Logic Array," *IEEE Trans. Comput.*, vol. 21, no. 8, pp. 848–857, Aug. 1972.

[2] M. Altun, "Computing with Emerging Nanotechnologies," in *Low-Dimensional and Nanostructured Materials and Devices: Properties, Synthesis, Characterization, Modelling and Applications*, H. Ünlü, N. J. M. Horing, and J. Dabowski, Eds. Springer International Publishing, 2016, pp. 635–660.

[3] M. Altun and M. D. Riedel, "Lattice-Based Computation of Boolean Functions," in *Proceedings of the 47th Design Automation Conference, DAC 2010, Anaheim, California, USA, July 13-18, 2010*, 2010, pp. 609–612.

[4] ——, "Logic Synthesis for Switching Lattices," *IEEE Trans. Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.

[5] A. Bernasconi, V. Ciriani, R. Drechsler, and T. Villa, "Logic Minimization and Testability of 2-SPP Networks," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 27, no. 7, pp. 1190–1202, 2008.

[6] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Fast Three-Level Logic Minimization Based on Autosymmetry," in *ACM/IEEE 39th Design Automation Conference (DAC)*, 2002, pp. 425–430.

[7] ——, "Implicit Test of Regularity for Not Completely Specified Boolean Functions," in *IEEE/ACM 11th International Workshop on Logic & Synthesis (IWLS)*, 2002, pp. 345–350.

[8] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, and T. Villa, "Logic Synthesis for Switching Lattices by Decomposition with P-Circuits," in *2016 Euromicro Conference on Digital System Design, DSD 2016, Limassol, Cyprus, August 31 - September 2, 2016*, 2016, pp. 423–430.

[9] A. Bernasconi, V. Ciriani, L. Frontini, and G. Trucco, "Synthesis on switching lattices of dimension-reducible boolean functions," in *IFIP/IEEE International Conference on Very Large Scale Integration (VLSI-SoC)*, 2016.

[10] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli, "Three-Level Logic Minimization Based on Function Regularities," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 8, pp. 1005–1016, 2003.

[11] ——, "Exploiting regularities for Boolean function synthesis," *Theory Comput. Syst.*, vol. 39, no. 4, pp. 485–501, 2006.

[12] ——, "Synthesis of autosymmetric functions in a new three-level form," *Theory Comput. Syst.*, vol. 42, no. 4, pp. 450–464, 2008.

[13] A. Bernasconi, V. Ciriani, and G. Trucco, "Biconditional-bdd ordering for autosymmetric functions," in *2015 Euromicro Conference on Digital System Design, DSD 2015, Madeira, Portugal, August 26-28, 2015*, 2015, pp. 211–217.

[14] V. Ciriani, "Logic Minimization Using Exclusive OR Gates," in *ACM/IEEE 38th Design Automation Conference (DAC)*, 2001, pp. 115–120.

[15] ——, "A New Approach to Three-Level Logic Synthesis," Computer Science Department, University of Pisa, Technical Report TR-02-03, 2002, submitted.

[16] P. Cohn, *Algebra Vol. 1*. John Wiley & Sons, 1981.

[17] O. Coudert, "Two-Level Logic Minimization: an Overview," *INTEGRATION*, vol. 17, pp. 97–140, 1994.

[18] D. Debnath and T. Sasao, "Multiple–Valued Minimization to Optimize PLAs with Output EXOR Gates," in *IEEE International Symposium on Multiple-Valued Logic*, 1999, pp. 99–104.

[19] E. Dubrova, D. Miller, and J. Muzio, "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.

[20] G. Gange, H. Søndergaard, and P. J. Stuckey, "Synthesizing Optimal Switching Lattices," *ACM Trans. Design Autom. Electr. Syst.*, vol. 20, no. 1, pp. 6:1–6:14, 2014.

[21] F. Luccio and L. Pagli, "On a New Boolean Function with Applications," *IEEE Transactions on Computers*, vol. 48, no. 3, pp. 296–310, 1999.

[22] M. C. Morgul and M. Altun, "Logic Circuit Design with Switching Nano Arrays and Area Optimization (in Turkish)," in *Elektrik, Elektronik, Bilgisayar ve Biyomedikal Mühendisligi Sempozyumu (ELECO)*, 2014.

[23] ——, "Synthesis and optimization of switching nanoarrays," in *Design and Diagnostics of Electronic Circuits and Systems (DDECS), 2015 IEEE International Symposium on*. IEEE, 2015, pp. 161–164.

[24] T. Sasao, "AND-EXOR Expressions and their Optimization," in *Logic Synthesis and Optimization*, T. Sasao, Ed. Kluwer Academic Publisher, 1993.

[25] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide Version 3.0," Microelectronic Center, User Guide, 1991.