Composition of Switching Lattices for Regular and for Decomposed Functions

Anna Bernasconi[a], Valentina Ciriani[b], Luca Frontini[b], Gabriella Trucco[b]

[a]*Dipartimento di Informatica, Università di Pisa, Italy, anna.bernasconi@unipi.it*
[b]*Dipartimento di Informatica, Università degli Studi di Milano, Italy,*
{*valentina.ciriani, luca.frontini, gabriella.trucco*}*@unimi.it*

Abstract

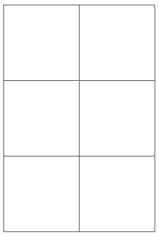
Multi-terminal switching lattices are typically exploited for modeling switching nano-crossbar arrays that lead to the design and construction of emerging nanocomputers. Typically, the circuit is represented on a single lattice composed by four-terminal switches. In this paper, we propose a two-layer model in order to further minimize the area of regular functions, such as *autosymmetric* and *D-reducible* functions, and of decomposed functions. In particular, 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 order 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 be inputs to 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. Finally, we propose the application of this particular lattice composition technique, based on connected multiple lattices, to the synthesis on switching lattices of D-reducible Boolean functions, and to the more general framework of

Preprint submitted to Embedded Hardware Design (Microprocessors and Microsystems)July 12, 2018

lattice synthesis based on logic function decomposition.

Keywords: Logic synthesis for emerging technologies, Switching lattices

---



1. Introduction

Going beyond standard CMOS networks representing Boolean functions, the interest on new circuit models has grown in the recent literature. In this framework, the logic optimization of switching lattices for emerging nanoscale technologies have been proposed and discussed in [6, 41].

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, was 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, 5, 6, 37, 41].

Previous studies on this subject [18, 19, 20] 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, the restriction 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 [43] have at least one non-degenerated autosymmetric output, and their minimization time is critically reduced in the frameworks of SOP and SPP synthesis [24, 25, 26]. 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} [24]. Note that an autosymmetric function f depends 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 [20] it is described a different 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 [34], Reed Muller [42]; three-level logic as SPP [8, 31, 38] (OR of ANDs of EXORs), EXSOP [35, 36] (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) the number of points of f_k is $|f|/2^k$ and f_k depends only on $n - k$ variables; (ii) the lattice minimization of f_k is naturally easier than that of f; and (iii) 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 lattice implementations have limited size.

Finally, we consider the application of this lattice composition technique, based on multiple lattices connected to each other, to the synthesis on switching lattices of D-reducible functions [13, 14, 15, 16], and to the more general framework of lattice synthesis based on logic function decomposition, focusing in particular on the P-circuit decomposition model [11, 12, 23, 29].

Experimental results show that by applying the proposed method to autosymmetric functions, 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. Moreover, our experimentation has confirmed that the application of this method turns out to be convenient also in other contexts of lattice synthesis based on functional decomposition.

This paper is an extended version of the conference paper in [21], where only the composition method for autosymmetric functions is described. The paper is organized as follows. Preliminaries on switching lattices and autosymmetric Boolean functions are described in Section 2. Section 3 discusses the problem

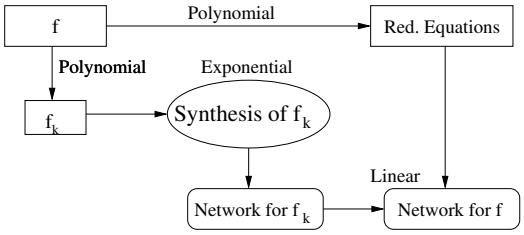


Figure 1: Synthesis of an autosymmetric function f through the synthesis of its restriction f_k.

of lattice composition, while Section 4 discusses the lattice implementation of autosymmetric functions. In Section 5, we show how to apply the proposed multiple-lattice composition approach both to the class of D-reducible functions, and in the general framework of lattice synthesis based on functional decomposition. Section 6 provides the experimental results and Section 7 concludes the paper.

2. Preliminaries

In this section we briefly review some basic notions and results on switching lattices [1, 6, 37] and autosymmetric Boolean functions [9, 10, 24, 25, 26, 27, 38].

2.1. 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.

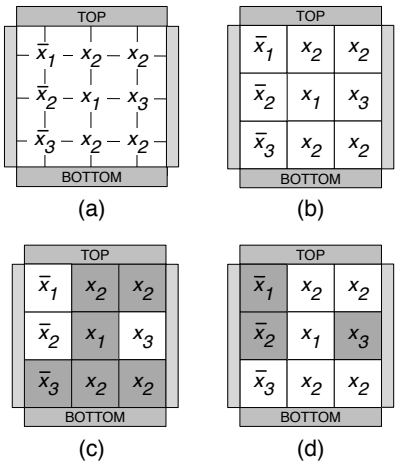


Figure 2: A four terminal switching network implementing the function $f = \overline{x}_1\overline{x}_2\overline{x}_3 + x_1x_2 + x_2x_3$ (a); its corresponding lattice form (b); the lattice evaluated on the assignments 1,1,0 (c) and 0, 0, 1 (d), with grey and white squares representing ON and OFF switches,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

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 [5, 6] 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:

$$SOP(f) = p_1 + p_2 + \ldots p_s \quad \text{and} \quad SOP(f^D) = q_1 + q_2 + \ldots q_r\,;$$

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 [37], 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.

2.2. Autosymmetric Boolean functions

In this section we briefly review autosymmetric functions that are introduced in [38] and further studied in [9, 10, 24, 25, 26, 27]. For the description of these particular regular functions we need to summarize several concepts of Boolean algebra [33].

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 ([25]). *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 ([25]). *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 [32]. 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 [33]:

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

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 [9], 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 [9] and [25] 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 [9, 25].

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

$$y_1 = x_1 \tag{1}$$
$$y_2 = x_2 \oplus x_3 \oplus x_4 \,. \tag{2}$$

Finally, the function f can be represented as:

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

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

3. Lattice composition

First of all, we recall from [37] 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. 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.

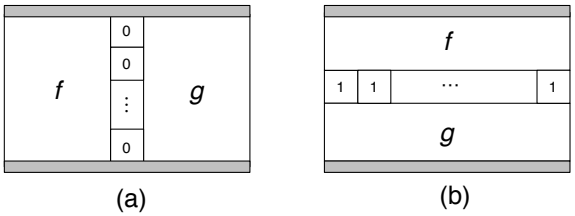


Figure 3: Lattice implementation of $f \vee g$ (a) and of $f \wedge g$ (b).

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 [18, 20] 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 [18, 20], 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

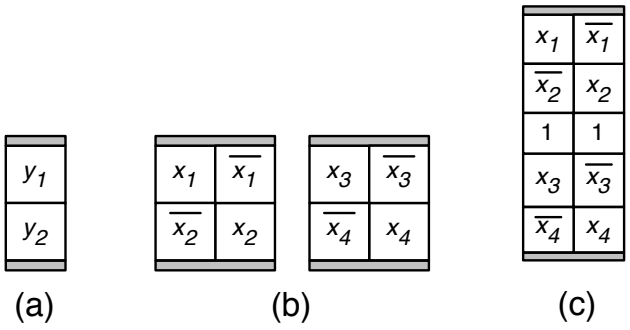


Figure 4: (a) Lattice implementation of $g = y_1y_2$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$; (c) final lattice for $f = (x_1 \oplus x_2)(x_3 \oplus x_4)$

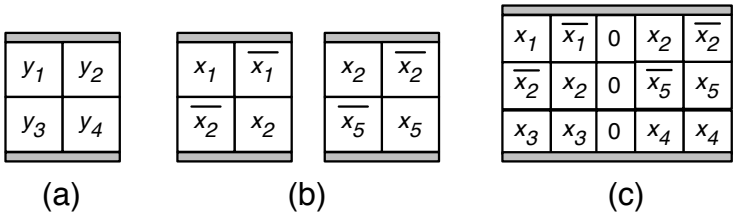


Figure 5: (a) Lattice implementation of $g = y_1y_3 + y_2y_4$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_2 \oplus x_5$; (c) final lattice for $f(x_1, x_2, x_3, x_4, x_5) = (x_1 \oplus x_2)x_3 + (x_2 \oplus x_5)x_4$.

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 5×2. Notice that using the lattice synthesis method presented in [6] 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 $(x_1 \oplus x_2)$

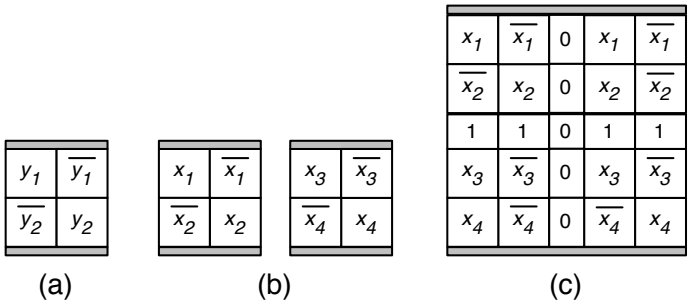


Figure 6: (a) Lattice implementation of $g = y_1 \oplus y_2$; (b) lattices for $y_1 = x_1 \oplus x_2$ and $y_2 = x_3 \oplus x_4$; (c) final lattice for $f(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

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 [40].

A possible strategy to overcome some of these problems could be a different lattice composition technique[1], that we could call *external composition*. The

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

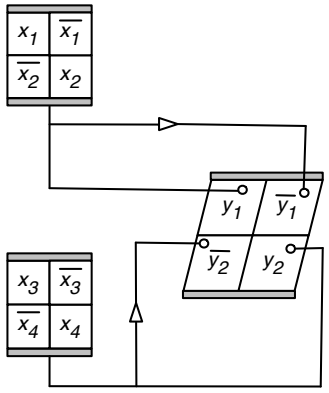


Figure 7: Multiple lattice implementation of $f(x_1, x_2, x_3, x_4, x_5) = x_1 \oplus x_2 \oplus x_3 \oplus x_4$.

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 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 [40]. 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.

4. Lattice representation of autosymmetric functions

The lattice implementation of autosymmetric functions can be derived exploiting the external lattice composition discussed in the previous Session 3. Recall from Section 2.2, 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_is: $y_i = EXOR(X_i)$, $X_i \subseteq \{x_1, \ldots, x_n\}$.

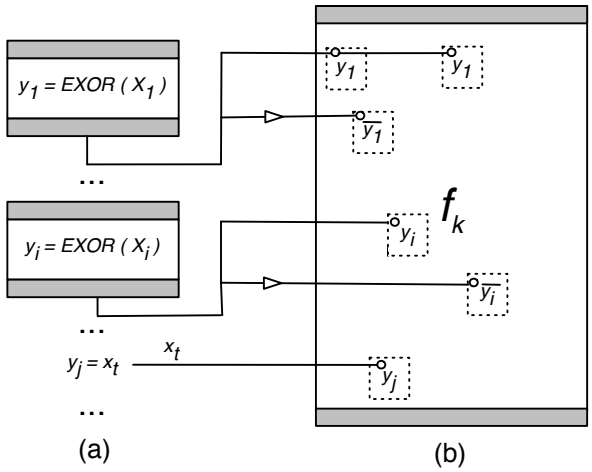


Figure 8: Multiple lattice implementation of an autosymmetric function: (a) lattice implementation of the reduction equations; (b) lattice implementation of the restriction f_k.

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.

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 [18] and [20] 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 [6] and in [37], and possibly even smaller than an optimal size lattice for f. This expectation has been confirmed by our experimental results.

5. Multiple lattice implementation of D-reducible functions and P-circuits

In this section we discuss the idea of exploiting the external lattice composition scheme in other contexts, with the aim of deriving multiple lattice implementations of reduced overall size.

We consider two approaches that have already been studied in previous works on switching lattice synthesis [18, 20].

5.1. D-reducible functions

First of all, we analyze the lattice synthesis of *D-reducible* functions. Recall from [13, 14, 15] that

- a D-reducible function is a function whose on-set minterms are completely contained in an affine space A strictly smaller than the whole Boolean cube $\{0,1\}^n$;
- a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where A is the smallest affine space that contains the on-set of f, χ_A is the characteristic function of A, and f_A is the projection of f onto A;
- f and f_A have the same number of points, but the points of the projection f_A are now compacted in a smaller space.

The D-reducibility property of a function f can be exploited in the lattice synthesis process as proposed in [20]: the idea is to independently find lattice implementations for the projection f_A and for the characteristic function χ_A of A, and then to compose them in order to construct the lattice for f, as shown in Figure 9 (a). Since the two functions f_A and χ_A depend on fewer variables than the original function f, their synthesis is more feasible and should produce lattice implementations of more compact area. The experimental results presented in [20] have validated this approach, demonstrating that *(i)* the lattice synthesis based on the D-reducibility property allows to obtain a more compact area in 15% of the considered cases, with an average gain of about 24%; and *(ii)* the synthesis time of the lattices can be reduced of about 50%, with respect to the time needed for the synthesis of plain lattices.

From Figure 9 (a) we can note that the overall lattice contains the padding row of 1s, used to join the accepting paths of χ_A and f_A, and some extra columns of 0s added to the lattice with fewer columns so that the final lattice has a rectangular shape and the accepting paths for one of the two functions never reach the opposite edge of the lattice if the other function evaluates to 0. This extra row and columns could be avoided using the external composition scheme as depicted in Figure 9

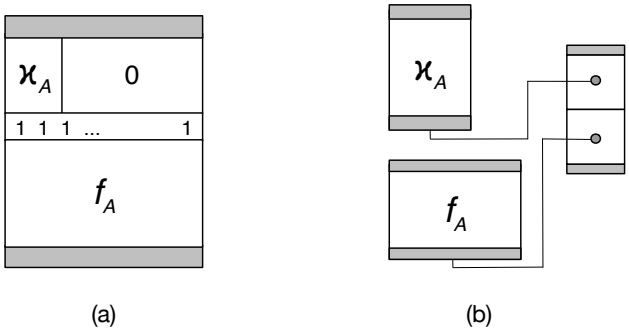


Figure 9: Lattice implementation of a D-reducible function: (a) standard lattice implementation based on internal composition; (b) multiple lattice implementation.

(b). The main lattice now consists in only one column of two switches, one switch is connected to the output of an external lattice implementing χ_A and the other switch is connected to the output of an external lattice for f_A. Since the main lattice consists in just two switches, we do not need any padding switch between them, and the lattice evaluates to 1 if and only if both external sublattices evaluate to 1. The size of the overall lattice is therefore given by 2 plus the sum of the sizes of the two sublattices, and it is smaller than the size of the lattice in Figure 9 (a) obtained by standard, internal composition. Only in a few cases, e.g., when the two sublattices for χ_A and f_A have the same number of columns and can be joined without the row of padding 1s, the internal composition produces a lattice of smaller area, but just for an additive factor 2.

5.2. P-circuits

In a very similar way, we can use external lattice composition in the lattice implementation of the bounded-level logic networks called ***P-circuits*** [11, 23, 29]. P-circuits are extended forms of Shannon cofactoring, where the expansion is with respect to an orthogonal basis $\overline{x}_i \oplus p$ (i.e., $x_i = p$), and $x_i \oplus p$ (i.e., $x_i \neq p$), where p is a function defined over all variables except for a critical variable x_i (e.g., the variable with more switching activity or with higher delay that should be projected away from the rest of the circuit). They can be defined as follows:

$$\text{P-circuit}(f) = (\overline{x}_i \oplus p)\, f^{=} + (x_i \oplus p)\, f^{\neq} + f^{I}$$

where I is the intersection of the projections of f onto the two sets $x_i = p$ and $x_i \neq p$, and

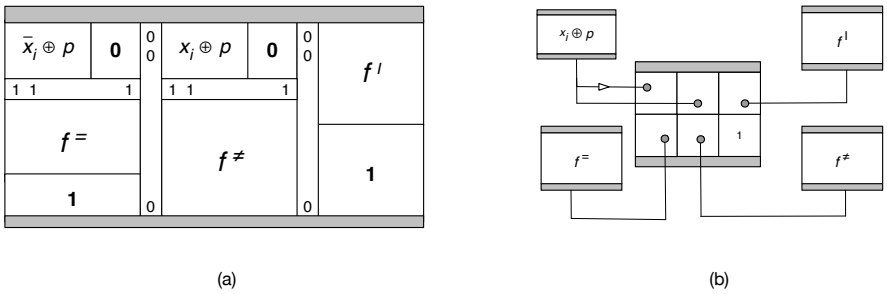


Figure 10: Lattice implementation of P-circuits: (a) standard lattice implementation based on internal composition; (b) multiple lattice implementation.

1. $(f|_{x_i=p} \setminus I) \subseteq f^= \subseteq f|_{x_i=p}$
2. $(f|_{x_i \neq p} \setminus I) \subseteq f^{\neq} \subseteq f|_{x_i \neq p}$
3. $\emptyset \subseteq f^I \subseteq I$.

Thus, the synthesis idea of P-circuits is to construct a network for f by appropriately choosing the sets $f^=$, $f^{\neq}$, and f^I as building blocks. The same idea can be exploited in the switching lattice framework: the subfunctions $f^=$, $f^{\neq}$, and f^I depend on $n-1$ variables instead of n, they have a smaller on-set than f, and their lattice synthesis should produce lattices of reduced area. Therefore, the overall lattice for f, derived composing minimal lattices for $f^=$, $f^{\neq}$, and f^I as shown in Figure 10 (a), could be smaller than the one derived for f without exploiting its P-circuits decomposition. This expectation has been confirmed by a set of experimental results, (see [18]) showing that in 30% of the analyzed cases the synthesis of switching lattices based on the P-circuit decomposition of the logic function allows to obtain a more compact area in the final resulting lattice, with an average gain of at least 20%. As before, the lattice obtained by internal composition of the sublattices for $f^=$, $f^{\neq}$, and f^I, contains padding rows and columns of 1s and 0s, that could be in part avoided applying the external decomposition scheme depicted in Figure 10 (b). The main lattice is composed of three columns of two switches. The first two columns contains switches connected to the output of external lattices implementing the projection function $(x_i \oplus p)$ and the cofactors $f^=$ and $f^{\neq}$; the last column contains one switch connected to the output of the external lattice implementing f^I, and one switch with constant value 1 used to connect the accepting path in the lattice for f^I to the bottom edge of the main lattice. Observe that the lattice correctly computes the disjunctions between f^I and the projections of $f^=$ and $f^{\neq}$, without the need for padding columns of 0s. This is due to the

fact that the lattice contains only two rows, and that each accepting path must contain at least two switches in the same column, as switches are not connected diagonally. Also note that the switch with constant value 1 can be replaced with a second switch connected to f^I, at the expense of the interconnection cost, so that the lattice does not contain switches labelled with constant values.

The overall lattice is therefore composed by a main lattice of size 6, four external sublattices, and one inverter, with an evident gain in the overall area.

6. Experimental results

In this section we report the experimental results obtained applying the multiple lattice implementation of autosymmetric functions described in Section 4. 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, using the CUDD library for OBDDs [17, 22, 30, 39] to represent Boolean functions, and BREL [7] to solve Boolean relations, as detailed in [28, 29]. The experiments have been run on a machine with two AMD Opteron 4274HE for a total of 16 CPUs at $2.5\,\mathrm{GHz}$ and $128\,\mathrm{GByte}$ of main memory, running Linux CentOS 6.6. The benchmarks are taken from LGSynth93 [43]. 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 sections.

To evaluate the utility of our approach, in Table 1 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 [6] and in [37], without exploiting the autosymmetry property. In Table 2 and Table 3 we compare the lattice synthesis results obtained applying the external decomposition scheme to D-reducible functions and P-circuits, with the corresponding internal decomposition scheme and the results obtained with the standard synthesis methods presented in [6] and in [37], without exploiting the decomposition property.

To simulate the results reported in [37], we used a collection of Python scripts for computing minimum-area switching lattices, by transformation to a series of SAT problems.

Each row of Table 1 lists the results for each separate autosymmetric output function of the benchmark circuit. More precisely, the first column reports the

Table 1: Proposed lattice sizes for autosymmetric benchmark circuits: a comparison of the proposed method with the results presented in [6] and in [37]. When the synthesis of a lattice is stopped, there is no lattice (−). Results are marked with ⋆ when SAT is stopped.

		[6]					[37]							
	$\#y_i$	Std. Synth.	Decomposed Synthesis				Std. Synth.		Decomposed Synthesis					
		Area	f_k Area	XOR Area	tot. Area	inv.	Area	time	f_k Area	f_k Time	XOR Area	XOR Time	tot. Area	tot. time	inv.
add6(0)	1	**4**	1	4	5	0	**4**	**0.021**	1	0.028	4	0.024	5	0.052	0
add6(1)	1	36	9	4	**14**	1	15	2.561	9	0.019	4	0.028	**14**	**0.047**	1
add6(2)	1	256	64	4	**69**	1	-	-	15	124	4	0.026	**20**	0.026	1
add6(3)	1	1296	324	4	**329**	1	-	-	-	-	4	0.029	**329**⋆	0.029	1
add6(4)	1	5776	1444	4	**1449**	1	-	-	-	-	4	0.035	**1449**⋆	0.035	1
add6(5)	1	24336	6084	4	**6089**	1	-	-	-	-	4	0.026	**6089**⋆	0.26	1
adr4(1)	1	345	324	4	**329**	1	-	-	-	-	4	0.025	**329**⋆	0.025	1
adr4(2)	1	1296	64	4	**69**	1	-	-	-	-	4	0.027	**20**⋆	0.027	1
adr4(3)	1	256	9	4	**14**	1	15	2.57	9	0.019	4	0.026	**14**	**0.045**	1
adr4(4)	1	36	1	4	**5**	0	-	-	1	0.026	4	0.031	**5**	0.026	0
al2(11)	1	125	60	4	**64**	0	-	-	-	-	4	0.021	**64**⋆	0.057	0
alcom(5)	1	12	6	4	**10**	0	12	**0.028**	6	0.025	4	0.3	**10**	0.325	0
b11(5)	1	**6**	2	4	**6**	0	**6**	**0.023**	2	0.025	4	0.025	**6**	0.05	0
b12(6)	1	54	35	4	**39**	0	**20**	**918**	20	1350	4	0.027	24	1350	0
dekoder(0)	1	8	3	4	**7**	0	8	**0.027**	3	0.024	4	0.022	**7**	0.046	0
dekoder(1)	1	**6**	2	4	**6**	0	**6**	**0.024**	2	0.023	4	0.023	**6**	0.046	0
dk27(8)	1	**1**	1	4	5	0	**1**	**0.021**	1	0.026	4	0.024	5	0.05	0
exps(18)	3	**16**	7	12	19	0	**12**	22.2	7	0.027	12	0.067	19	**0.094**	0
exps(19)	6	**16**	7	24	31	0	**16**	**0.03**	7	0.024	24	0.137	31	0.161	0
f51m(6)	1	**4**	1	4	5	0	**4**	**0.03**	1	0.025	4	0.026	5	0.051	0
luc(3)	1	16	9	4	**13**	0	**12**	0.861	9	0.024	4	0.031	13	**0.055**	0
m1(8)	1	15	8	4	**12**	0	**12**	1.65	8	0.333	4	0.024	**12**	**0.357**	0
max1024(0)	8	56	14	32	**46**	0	-	-	14	1.16	32	0.202	**46**	116	0
max1024(1)	8	324	81	32	**119**	6	324	**0.034**	24	2543	32	0.198	**63**	2543	7
max1024(2)	9	1216	304	36	**348**	8	-	-	304	0.036	36	0.23	**348**	0.266	8
max1024(3)	9	3072	800	36	**844**	8	3072	**0.133**	-	-	36	0.22	**844**⋆	0.22	8
max1024(4)	9	6806	1978	36	**2023**	9	6806	**0.23**	1978	0.07	36	0.218	**2023**	0.288	9
max1024(5)	9	14274	3968	36	**4013**	9	-	-	-	-	36	0.221	**4013**⋆	0.221	9
newcond(1)	2	**8**	3	8	11	0	**8**	0.244	3	0.023	8	0.051	11	**0.074**	0
newcwp(0)	2	20	6	8	**15**	1	**12**	0.521	6	0.023	8	0.053	15	**0.076**	1
newcwp(1)	1	16	1	12	**13**	0	**9**	**0.303**	1	0.025	9	0.281	10	0.306	0
newcwp(3)	2	**4**	1	4	5	0	**4**	**0.022**	1	0.021	4	0.027	5	0.048	0
p82(10)	2	10	4	4	**8**	0	**8**	0.106	4	0.021	4	0.025	**8**	**0.046**	0
pope.rom(18)	3	**12**	5	12	17	0	**10**	1.45	5	0.025	12	0.07	17	**0.095**	0
pope.rom(32)	1	8	3	4	**7**	0	**6**	0.086	3	0.027	4	0.028	7	**0.055**	0
pope.rom(34)	1	8	3	4	**7**	0	8	1806	3	0.023	4	0.019	**7**	**0.042**	0
pope.rom(35)	1	**6**	2	8	10	0	**6**	0.077	2	0.025	8	0.05	10	**0.075**	0
pope.rom(41)	1	10	4	4	**8**	0	10	**0.022**	4	0.031	4	0.023	**8**	0.054	0
radd(0)	1	**4**	1	4	5	0	**4**	**0.025**	1	0.021	4	0.026	5	0.047	0
radd(1)	1	36	9	4	**14**	1	15	2.45	9	0.026	4	0.027	**14**	**0.053**	1
radd(2)	1	256	64	4	**69**	1	-	-	15	122.1	4	0.025	**20**	122	1
radd(3)	1	1296	324	4	**329**	1	-	-	-	-	4	0.024	**329**⋆	0.024	1
rd53(1)	4	100	30	16	**50**	4	-	-	12	0.676	16	0.118	**32**	0.794	4
rd53(2)	1	256	1	80	**81**	0	-	-	1	0.022	-	-	**81**⋆	0.022	0
rd73(2)	1	1225	1	448	**449**	0	-	-	1	0.023	-	-	**449**⋆	0.023	0
risc(4)	1	**6**	2	4	**6**	0	**6**	**0.026**	2	0.027	4	0.02	**6**	0.047	0
sqn(0)	2	272	49	8	**58**	1	-	-	15	11	8	0.049	**24**	11.5	1
wim(2)	1	**6**	2	4	**6**	0	**6**	**0.026**	2	0.023	4	0.019	**6**	0.042	0
z4(1)	5	784	64	28	**97**	5	784	**11.34**	15	115	25	0.412	**45**	115	5
z4(2)	3	144	9	20	**32**	3	-	-	9	0.021	17	0.341	**29**	0.362	3
z4(3)	1	16	1	12	**13**	0	**9**	**0.29**	1	0.027	9	0.286	10	0.313	0
z5xp1(8)	1	**4**	1	4	5	0	**4**	**0.027**	1	0.019	4	0.026	5	0.045	0
z9sym(0)	8	6192	2016	32	**2056**	8	-	-	-	-	32	0.217	**2056**⋆	0.276	8

Table 2: Proposed lattice sizes for standard benchmark circuits: a comparison of the proposed external D-reducible decomposition method with the results presented in [20], [6] and [37]. When the synthesis of a lattice is stopped, there is no lattice (−). Results are marked with ⋆ when SAT is stopped.

	[6]								[37]							
	Std. synthesis		Internal Decomp.		External Decomp.				Std. synthesis		Internal Decomp.		External Decomp.			
	X×Y	Area	X×Y	Area	χ	f_A	cost	Area	X×Y	Area	X×Y	Area	χ	f_A	cost	Area
addm4(0)	9×12	180	9×15	135	1×3	9×12	2	**113**	-	-	3×8	24	1×3	3×5	2	**20**
addm4(1)	22×23	506	22×23	506	1×1	22×22	1	**486**	-	-	22×23	506⋆	1×1	22×22	1	**486**⋆
addm4(2)	33×36	1180	33×36	1188	1×1	33×35	1	**1157**	-	-	33×36	1188⋆	1×1	33×35	1	**1157**⋆
adr4(4)	2×2	**4**	2×2	**4**	2×3	1×0	0	6	2×2	**4**	2×5	10	2×5	0×0	0	10
alu2(6)	4×4	**16**	4×5	20	1×1	4×4	1	18	3×4	**12**	4×4	16	1×1	4×3	1	14
amd(3)	6×8	**48**	6×9	54	1×1	6×8	1	50	4×5	**20**	3×7	21	1×1	3×6	1	**20**
amd(4)	10×14	140	10×14	140	1×1	10×13	1	**132**	-	-	10×14	140⋆	1×1	10×13	1	**132**⋆
amd(5)	2×8	**16**	2×9	18	2×6	2×2	2	18	6×2	**12**	2×8	16	2×6	2×2	2	18
amd(6)	3×9	**27**	3×14	42	1×5	3×9	2	34	6×3	18	3×8	24	1×5	3×3	2	**16**
amd(7)	6×7	**42**	6×8	48	1×1	6×7	1	44	4×5	20	3×6	18	1×1	3×5	1	**17**
apla(0)	4×13	52	4×13	52	1×3	4×10	2	**45**	6×3	30	5×6	30	1×3	5×3	2	**20**
apla(1)	4×12	48	4×12	48	1×3	4×9	2	**41**	6×3	18	4×6	24	1×3	4×3	2	**17**
apla(2)	3×6	**18**	5×18	90	1×2	5×16	2	84	5×2	**10**	4×4	16	1×2	4×2	2	12
apla(7)	5×10	**50**	6×12	72	1×2	6×10	2	64	-	-	4×6	24	1×2	4×4	2	**20**
apla(9)	5×15	**75**	6×22	132	1×2	6×20	2	124	-	-	5×6	30	1×2	5×4	2	**24**
b10(2)	10×14	**140**	10×19	190	1×5	10×14	2	147	-	-	10×14	140⋆	1×5	10×9	2	**97**⋆
br1(3)	2×12	**24**	2×13	26	2×13	2×1	2	30	-	-	2×12	**24**	2×11	2×1	2	26
br1(4)	6×15	**90**	6×20	120	1×5	6×15	2	97	-	-	5×9	45	1×5	5×4	2	**27**
br2(4)	8×18	**144**	8×20	160	1×2	8×18	2	148	-	-	8×18	144⋆	1×2	8×16	2	**132**⋆
br2(5)	4×14	56	8×16	128	2×9	4×6	2	**44**	-	-	4×21	84⋆	2×17	4×3	2	**48**⋆
br2(6)	5×16	80	5×17	85	2×7	5×9	2	**61**	-	-	4×12	48⋆	2×7	4×4	2	**32**⋆
dk48(2)	2×13	**26**	2×22	44	1×9	2×13	2	37	-	-	2×13	26⋆	1×9	2×4	2	**19**⋆
dk48(3)	2×13	**26**	4×17	68	2×15	1×0	0	30	-	-	2×15	**30**⋆	2×15	0×0	0	**30**⋆
exam(4)	9×25	225	11×22	242	1×2	11×20	2	**224**	-	-	9×20	180⋆	1×2	9×18	2	**166**⋆
exp(6)	6×7	**42**	8×12	96	1×2	8×10	2	84	5×4	20	3×7	21	1×2	3×5	2	**19**
exp(10)	6×12	**72**	6×15	90	1×2	6×13	2	82	-	-	5×6	30	1×2	5×4	2	**24**
exp(11)	6×12	72	5×12	60	2×4	5×7	2	**45**	-	-	5×8	40	2×4	5×3	2	**25**
gary(2)	12×14	168	12×15	180	2×3	12×12	2	**152**	-	-	12×18	216⋆	2×5	12×12	2	**156**⋆
gary(3)	5×12	**60**	5×14	70	1×2	5×12	2	64	-	-	5×12	60	1×2	5×10	2	**54**
in2(6)	39×36	**1404**	39×38	1482	1×2	39×36	2	1408	-	-	39×35	1365⋆	1×2	39×33	2	**1291**⋆
in2(7)	17×26	**442**	17×27	459	1×1	17×26	1	444	-	-	17×26	442⋆	1×1	17×25	1	**427**⋆
in2(8)	27×31	**837**	27×32	864	1×1	27×31	1	839	-	-	27×31	837⋆	1×1	27×30	1	**812**⋆
in2(9)	40×36	**1440**	40×37	1480	1×1	40×36	1	1442	-	-	40×36	1440⋆	1×1	40×35	1	**1402**⋆
in7(6)	11×18	**198**	11×21	231	1×3	11×18	2	203	-	-	11×18	198⋆	1×3	11×15	2	**170**⋆
m2(6)	10×13	**130**	10×14	140	1×1	10×13	1	132	-	-	10×13	130⋆	1×1	10×12	1	**122**⋆
m2(7)	14×14	**196**	14×15	210	1×1	14×14	1	198	-	-	14×14	196⋆	1×1	14×13	1	**184**⋆
m2(12)	6×11	**66**	6×13	78	1×2	6×11	2	70	5×4	**20**	4×6	24	1×2	4×4	2	**20**
m2(13)	9×12	**108**	9×14	126	1×2	9×12	2	112	-	-	4×8	32	1×2	4×6	2	**28**
m2(15)	16×18	**288**	17×19	323	1×1	17×18	1	308	-	-	16×18	288⋆	1×1	16×17	1	**274**⋆
m4(9)	4×7	**28**	4×10	40	1×3	4×7	2	33	5×3	**15**	4×6	24	1×3	4×3	2	17
m4(10)	7×7	**49**	7×9	63	1×2	7×7	2	53	7×7	49	3×8	24	1×2	3×6	2	**22**
max128(3)	6×6	36	5×7	35	2×3	5×4	2	**28**	3×6	**18**	3×11	33	2×5	3×5	2	27
newapla1(6)	5×6	**30**	5×11	55	1×5	5×6	2	37	-	-	5×6	30	1×5	5×1	2	**12**
newapla(0)	4×6	**24**	4×7	28	1×1	4×6	1	26	3×6	**18**	3×7	21	1×1	3×6	1	20
newcpla1(6)	6×11	**66**	6×13	78	1×2	6×11	2	70	-	-	4×6	24	1×2	4×4	2	**20**
newcpla1(7)	7×6	**42**	7×7	49	1×1	7×6	1	44	4×5	20	3×6	18	1×1	3×5	1	**17**
newcpla1(8)	7×12	**84**	7×15	105	1×3	7×12	2	89	-	-	5×7	35	1×3	5×4	2	**25**
newtpla1(1)	2×9	**18**	2×15	30	1×6	2×9	2	26	2×9	18	2×9	18	1×6	2×3	2	**14**
newtpla2(2)	2×9	**18**	2×17	34	1×8	2×9	2	28	2×9	18	2×9	18	1×8	2×1	2	**12**
newtpla(6)	3×12	**36**	3×16	48	1×4	3×12	2	42	-	-	6×7	42	1×4	6×3	2	**24**
newtpla(9)	2×7	**14**	2×13	26	1×6	2×7	2	22	6×2	12	2×7	14	1×6	2×1	2	**10**
newxcpla1(1)	7×6	**42**	7×7	49	1×1	7×6	1	44	3×6	18	3×6	18	1×1	3×5	1	**17**
p1(2)	3×6	**18**	7×10	70	2×6	7×4	2	42	4×3	**12**	2×9	18	2×5	2×3	2	18
p1(11)	6×9	54	5×12	60	2×4	5×7	2	**45**	3×5	**15**	5×8	40	2×4	5×3	2	25

Table 3: Proposed lattice sizes for standard benchmark circuits: a comparison of the proposed external P-circuit decomposition method with the results presented in [18], [6] and [37]. When the synthesis of a lattice is stopped, there is no lattice (−). Results are marked with ⋆ when SAT is stopped.

	[6]									[37]								
	Std. synthesis		Internal Decomp.		External Decomp.					Std. synthesis		Internal Decomp.		External Decomp.				
	X×Y	Area	X×Y	Area	$A_{f=}$	$A_{f\neq}$	A_{fI}	cost	Area	X×Y	Area	X×Y	Area	$A_{f=}$	$A_{f\neq}$	A_{fI}	cost	Area
adr4(1)	36×36	1296	37×19	703	324	324	0	5	**653**	–		37×19	703⋆	324⋆	324	0	5	**653**⋆
alu2(2)	11×10	110	13×7	91	6	5	56	7	**74**	7 × 3	**21**	10×6	60	6	5	15	7	33
alu2(5)	14×13	182	16×10	160	8	6	110	7	**131**	–		16×10	160⋆	8	6	110	7	**131**⋆
alu3(0)	5×4	20	7×4	28	2	3	4	7	**16**	3 × 3	**9**	7×4	28	2	3	4	7	16
alu3(1)	8×7	56	10×5	50	4	4	20	7	**35**	5 × 3	**15**	8×5	40	4	4	9	7	24
alu3(2)	10×11	110	12×8	96	4	8	56	7	**75**	6 × 4	**24**	11×5	55⋆	4	8	18	7	37⋆
b12(0)	4×6	24	4×6	24	20	0	0	3	**23**	3 × 4	**12**	3×4	**12**⋆	9	0	0	3	**12**⋆
b12(1)	7×5	35	7×5	35	28	0	0	3	**31**	4 × 4	16	4×4	16⋆	12	0	0	3	**15**⋆
bcc(5)	9×27	243	9×26	234	225	0	0	3	**228**	–		9×26	234⋆	225	0	0	3	**228**⋆
bcc(7)	11×31	341	12×29	348	280	10	0	5	**295**	–		12×29	348⋆	280	10	0	5	**295**⋆
bcc(8)	12×31	372	13×29	377	308	10	0	5	**323**	–		13×29	377⋆	308	10	0	5	**323**⋆
bcc(12)	11×31	341	11×30	330	319	0	0	3	**322**	–		11×30	330⋆	319	0	0	3	**322**⋆
bcc(27)	19×39	741	20×33	660	416	90	0	5	**511**	–		20×33	660⋆	416	90	0	5	**511**⋆
bcc(43)	10×20	200	11×16	176	78	60	0	5	**143**	–		11×16	176⋆	78	60	0	5	**143**⋆
bcd.div3(1)	3×4	**12**	5×4	20	3	1	2	7	13	3 × 3	**9**	5×4	20	3	1	2	7	13
bcd.div3(2)	3×4	**12**	5×4	20	3	1	3	7	14	3 × 3	**9**	5×4	20	3	1	3	7	14
bcd.div3(3)	3×5	15	4×4	16	6	0	3	5	**14**	3 × 4	**12**	4×4	16⋆	6	0	3	5	14⋆
bench1(2)	24×45	1080	33×29	957	350	504	0	5	**859**	–		33×29	957⋆	350	504	0	5	**859**⋆
bench1(3)	16×31	496	20×18	360	104	136	14	7	**261**	–		21×18	378⋆	104	136	12	7	**259**⋆
bench1(5)	27×50	1350	32×28	896	128	78	448	7	**661**	–		32×28	896⋆	128	78	448	7	**661**⋆
bench1(6)	21×35	735	26×24	624	160	345	0	5	**510**	–		26×24	624⋆	160	345	0	5	**510**⋆
bench1(7)	21×43	903	27×20	540	247	190	14	7	**458**	–		27×20	540⋆	247	190	10	7	**454**⋆
bench1(8)	24×44	1056	31×26	806	375	345	0	5	**725**	–		31×26	806⋆	375	345	0	5	**725**⋆
bench(6)	4×8	32	6×3	18	2	2	6	7	**17**	3 × 4	**12**	6×3	18	2	2	6	7	17
br2(4)	8×18	144	8×18	144	0	136	0	2	**138**	–		8×18	144⋆	0	136	0	2	**138**⋆
br2(5)	4×14	56	4×14	56	0	52	0	2	**54**	–		4×14	56⋆	0	52	0	2	**54**⋆
br2(6)	5×16	80	5×16	80	0	75	0	2	**77**	–		5×16	80⋆	0	75	0	2	**77**⋆
clpl(2)	2×2	**4**	3×2	6	0	1	4	4	6	2 × 2	**4**	3×2	6⋆	0	1	1	4	6⋆
clpl(3)	6×6	**36**	9×6	54	1	10	20	6	37	6 × 3	**18**	9×6	54⋆	1	10	12	7	30⋆
clpl(4)	5×5	**25**	8×5	40	1	8	12	6	27	5 × 3	**15**	8×5	40⋆	1	8	9	7	25⋆
co14(0)	14×92	1288	15×80	1200	1027	13	0	5	**1045**	–		15×80	1200⋆	1027	13	0	5	**1045**
dc1(0)	4×4	**16**	5×4	20	9	2	0	5	**16**	3 × 3	**9**	4×4	16⋆	6	2	0	5	13⋆
dc1(1)	2×3	**6**	3×3	9	2	0	3	5	10	2 × 3	**6**	3×3	9⋆	2	0	3	5	10⋆
dc1(4)	4×5	20	5×4	20	9	0	3	5	**17**	3 × 4	**12**	5×4	20⋆	9	0	3	5	17⋆
dc1(6)	3×3	9	4×2	**8**	2	0	2	5	9	3 × 2	**6**	4×2	8⋆	2	0	2	5	9⋆
dc2(4)	9×10	90	10×9	90	48	18	0	5	**71**	4 × 5	**20**	8×5	40⋆	16	12	0	5	33⋆
dc2(5)	6×6	36	7×7	49	12	18	0	5	**35**	2 × 6	**12**	5×6	30⋆	8	10	0	5	23⋆
dk17(0)	2×8	16	4×4	16	6	2	0	5	**13**	2 × 6	**12**	4×4	16⋆	6	2	0	5	13⋆
dk17(1)	2×8	16	4×4	16	6	2	0	5	**13**	2 × 6	**12**	4×4	16⋆	6	2	0	5	13⋆
dk17(3)	3×11	33	4×7	**28**	0	0	28	0	**28**	2 × 7	**14**	6×3	18⋆	0	0	18	0	18⋆
dk17(4)	3×9	27	6×4	24	3	2	6	7	**18**	2 × 7	**14**	6×4	24	3	2	6	0	18
dk17(6)	1×3	**3**	1×3	**3**	0	0	3	0	**3**	1 × 3	**3**	1×3	**3**⋆	0	0	3	0	**3**⋆
exp(4)	6×17	**102**	6×17	**102**	0	0	102	0	104	–		6×17	**102**⋆	0	0	102	0	**102**⋆
exp(5)	45×35	**1575**	45×35	**1575**	0	0	1575	0	1577	–		45×35	**1575**⋆	0	0	1575	0	**1575**⋆
exp(32)	10×4	40	13×3	39	10	8	4	7	**29**	6 × 4	**24**	13×3	39	10	8	4	7	29
exp(33)	7×3	21	7×3	21	0	1	15	4	**20**	–		7×3	21⋆	0	1	15	4	**20**⋆
exp(34)	10×4	**40**	12×5	60	2	4	30	7	43	6 × 4	**24**	11×3	33	2	4	15	7	28
exp(36)	8×2	**16**	10×2	20	1	1	12	6	20	8 × 2	**16**	10×2	20	1	1	12	7	21
exp(38)	9×4	36	13×3	39	12	8	1	6	**28**	7 × 4	**24**	13×3	39	12	8	1	7	28
exp(39)	8×2	**16**	11×3	33	1	8	8	6	23	–		11×3	33	1	8	8	7	**24**
exp(40)	12×6	72	15×5	75	16	18	12	7	**53**	–		13×4	52	8	12	9	7	**36**
exp(43)	14×8	112	17×6	102	36	35	0	5	**76**	–		13×4	52⋆	18	18	0	5	**41**⋆
exam(5)	6×11	66	7×6	42	4	0	30	5	**39**	–		6×5	30⋆	4	0	16	5	**25**⋆
exam(9)	30×59	1770	38×30	1140	754	143	0	5	**902**	–		33×30	990⋆	754	24	0	5	**783**⋆
max128(5)	17×14	238	19×9	171	16	80	21	7	**124**	–		14×5	70	9	24	12	7	**52**
max128(8)	10×5	50	11×4	44	18	8	0	5	**31**	–		10×4	40⋆	15	8	0	5	**28**⋆
max128(17)	25×26	650	26×15	390	144	182	0	5	**331**	–		26×15	390⋆	144	182	0	5	**331**⋆
mp2d(6)	6×10	60	6×10	60	0	54	0	2	**56**	–		3×7	21⋆	0	18	0	2	**20**⋆
mp2d(9)	8×6	48	9×6	54	0	15	5	4	**24**	–		9×4	36⋆	0	9	5	4	**18**⋆
mp2d(10)	3×6	18	4×5	20	8	4	0	5	**17**	3 × 4	**12**	4×5	20⋆	8	4	0	5	17⋆
z4(0)	15×15	225	16×11	176	0	24	77	4	**105**	4 × 5	**20**	6×6	36⋆	0	10	12	4	26⋆
z4(1)	28×28	784	30×16	480	32	32	192	7	**263**	–		10×7	70	12	12	24	7	**55**
Z5×p1(2)	11×12	132	13×7	91	12	36	8	7	**63**	–		11×5	55⋆	12	16	8	7	**43**⋆
Z5×p1(3)	18×18	324	19×11	209	80	80	0	5	**165**	–		10×6	60⋆	20	20	0	5	**45**⋆

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 five columns refer to the synthesis of lattices as described in [6], with (columns 4-7) and without (column 3) the multiple lattice decomposition based on autosymmetry. In particular, column 3 shows the area of lattices derived applying the standard synthesis method (i.e., without exploiting the autosymmetry property), column 4 shows the area of the lattice for the restriction f_k, column 5 shows the total area of the lattices for the EXOR terms y_i, column 6 shows the total area occupied by lattices ($TotalArea = A_{f_k} + \sum_i A_{y_i} + num.inv$), and column 7 indicates the number $num.inv$ of inverters necessary to make the signal routing. The synthesis in [6] 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 8 to 16 refer to lattices synthesized using the methodology presented in [37], with and without decomposition on multiple lattices. In particular columns 8 and 9 report the area and the synthesis time of lattices obtained with standard synthesis; columns 10 and 11 report the area and the synthesis time of the lattice for the restriction f_k, column 12 and 13 report the total area of the lattices for the EXOR terms y_i and their synthesis time; columns 14 and 15 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 16 indicates the number $num.inv$ of inverter necessary for signal routing.

For each function, we bolded the best areas (col. 3 vs col. 5 vs col. 6 and col. 8 vs col. 14) and the best total time (col. 9 vs col.15).

Each row of Table 2 lists the results for each separate D-reducible output function of the benchmark circuits. More precisely, the first column reports the name and the number of the considered output of each instance; The following two columns refer to the synthesis of lattices as described in [6], without lattice decomposition (columns 2-3), with internal decomposition (column 4-5), and external decomposition (column 6-9). In particular, columns 2-3 show the dimension and the area of lattices derived applying the standard synthesis method (i.e., without exploiting the D-reducibility property), columns 4-5 show the dimension and the area of the lattice obtained applying the internal decomposition method, column 6 shows the dimension of χ, column 7 shows the dimension of the lattice of f_A, column 8 shows the cost in term of lattice area due to external decomposition, column 9 shows the total area occupied by lattices ($TotalArea = A_\chi + A_{f_A} + cost$).

Columns 10 to 17 of Table 2 refer to lattices synthesized using the methodol-

ogy presented in [37], the content refers to the same lattice as columns 2-9.

For each function, we bolded the best areas (col. 3 vs col. 5 vs col. 9 and col. 11 vs col. 13 vs col. 17).

Each row of Table 3 lists the results for each separate output function, represented as a P-circuit, of the benchmark circuits. More precisely, the first column reports the name and the number of the considered output of each instance. The following two columns refer to the synthesis of lattices as described in [6], without lattice decomposition (columns 2-3), with internal decomposition (column 4-5), and external decomposition (column 6-10). In particular, columns 2-3 show the dimension and the area of lattices derived applying the standard synthesis method (i.e., without exploiting the P-circuit decomposition), columns 4-5 show the dimension and the area of the lattice obtained applying the internal decomposition method, column 6 shows the dimension of the lattice of the projection $f^{=}$, column 7 shows the dimension of the lattice of the projection $f^{\neq}$, column 8 shows the dimension of the lattice of the intersection f^{I}, column 9 shows the cost in term of lattice area due to external decomposition, and column 10 reports the total area occupied by lattices ($TotalArea = A_{f^{=}} + A_{f^{\neq}} + A_{f^{I}} + cost$).

Columns 11 to 19 of Table 3 refer to lattices synthesized using the methodology presented in [37], the content refers to the same lattice as columns 2-10.

For each function, we bolded the best areas (col. 3 vs col. 5 vs col. 10 and col. 12 vs col. 14 vs col. 19).

In some cases the method proposed in [37] 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 [37] is stopped, we use the sublattices synthesized with [6], 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 [37] did not find a solution within the fixed time limit for at least one sublattice, and had to be replaced with [6].

The results are promising. Considering the methodology presented in [6], for the class of autosymmetric functions (see Table 1) 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 [37], 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 15 in Table 1) is smaller than the time necessary to perform

Table 4: Comparison between external decomposition method with not decomposed lattice and internal decomposed lattice

Decomposition method		Not-decomposed lattices		Lattice with internal decomposition	
		less area	area gain	less area	area gain
D-reducible	[6]	16%	9%	78%	9%
	[37]	54%	9%	75%	6%
P-circuit	[6]	54%	36%	93%	15%
	[37]	60%	39%	93%	17%
Autosymmetric	[6]	58%	53%	–	–
	[37]	48%	60%	–	–

the standard synthesis (column 9 in Table 1).

As for external vs internal decomposition, we report in Table 4 the overall results concerning the external decomposition applied to all different decomposition methods with respect to standard synthesized lattices and internal decomposition. In this table, each row is referred to a different decomposition method. Column 2 shows the synthesis method used for the experiment, columns 3 and 4 show the percentage of lattices with less area and how much area is gained with respect to not-decomposed lattice, columns 5 and 6 show the percentage of lattices with less area and how much area is gained with respect to internal decomposed lattices. These results clearly show how the use of multiple lattices often allows to reduce the number of switches and thus the overall dimension of the lattice, even if the gain in the dimension comes at the expense of an increase in the interconnection cost.

7. Conclusions

In this paper we have shown a lattice minimization strategy for autosymmetric function. We have described how to exploit an external composition for autosymmetric functions in order to get compact area representation with switching lattices. We have also proposed the application of the external composition technique to the synthesis on switching lattices of D-reducible Boolean functions, and to the more general framework of lattice synthesis based on logic function decomposition, focusing in particular on the P-circuit decomposition model. The experimental results have validated the approach.

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

8. 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] D. Alexandrescu, M. Altun, L. Anghel, A. Bernasconi, V. Ciriani, L. Frontini, and M. B. Tahoori, "Logic synthesis and testing techniques for switching nano-crossbar arrays," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 54, pp. 14–25, 2017. [Online]. Available: https://doi.org/10.1016/j.micpro.2017.08.004

[3] 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.

[4] M. Altun, V. Ciriani, and M. B. Tahoori, "Computing with nano-crossbar arrays: Logic synthesis and fault tolerance," in *Design, Automation & Test in Europe Conference & Exhibition, DATE 2017, Lausanne, Switzerland, March 27-31, 2017*, 2017, pp. 278–281.

[5] 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.

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

[7] D. Bañeres, J. Cortadella, and M. Kishinevsky, "A Recursive Paradigm to Solve Boolean Relations," *IEEE Transactions on Computers*, vol. 58, no. 4, pp. 512–527, 2009.

[8] 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.

[9] 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.

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

[11] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "On Decomposing Boolean Functions via Extended Cofactoring," in *Design Automation and Test in Europe (DATE)*, 2009, pp. 1464–1469.

[12] ——, "Logic Synthesis by Signal-Driven Decomposition," in *Advanced Techniques in Logic Synthesis, Optimizations and Applications*, K. Gulati, Ed. Springer New York, 2011, pp. 9–29.

[13] A. Bernasconi and V. Ciriani, "DRedSOP: Synthesis of a New Class of Regular Functions." in *Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools (DSD)*, 2006, pp. 377–384.

[14] ——, "Logic synthesis and testability of d-reducible functions," in *IFIP/IEEE VLSI-SoC 2010 - International Conference on Very Large Scale Integration of System-on-Chip,*, 2010, pp. 280–285.

[15] ——, "Dimension-reducible Boolean functions based on affine spaces," *ACM Trans. Design Autom. Electr. Syst.*, vol. 16, no. 2, p. 13, 2011.

[16] ——, "Autosymmetric and dimension reducible multiple-valued functions," *Multiple-Valued Logic and Soft Computing*, vol. 23, no. 3-4, pp. 265–292, 2014.

[17] ——, "Index-resilient zero-suppressed bdds: Definition and operations," *ACM Trans. Design Autom. Electr. Syst.*, vol. 21, no. 4, pp. 72:1–72:27, 2016.

[18] 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.

[19] ——, "Enhancing logic synthesis of switching lattices by generalized shannon decomposition methods," *Microprocessors and Microsystems - Embedded Hardware Design*, vol. 56, pp. 193–203, 2018.

[20] 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.

[21] ——, "Composition of Switching Lattices and Autosymmetric Boolean Function Synthesis," in *Euromicro Conference on Digital System Design, DSD 2017, Vienna, Austria, August 30 - Sept. 1, 2017*, 2017, pp. 137–144.

[22] A. Bernasconi, V. Ciriani, and L. Lago, "On the error resilience of ordered binary decision diagrams," *Theor. Comput. Sci.*, vol. 595, pp. 11–33, 2015.

[23] A. Bernasconi, V. Ciriani, V. Liberali, G. Trucco, and T. Villa, "Synthesis of P-Circuits for Logic Restructuring," *Integration*, vol. 45, no. 3, pp. 282–293, 2012.

[24] 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.

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

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

[27] 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.

[28] A. Bernasconi, V. Ciriani, G. Trucco, and T. Villa, "Minimization of EP-SOPs via Boolean relations," in *IFIP/IEEE VLSI-SoC 2013 - International Conference on Very Large Scale Integration of System-on-Chip,*, 2013, pp. 112–117.

[29] ——, "Using Flexibility in P-Circuits by Boolean Relations," *IEEE Trans. Computers*, vol. 64, no. 12, pp. 3605–3618, 2015.

[30] R. Bryant, "Graph Based Algorithm for Boolean Function Manipulation," *IEEE Transactions on Computers*, vol. 35, no. 9, pp. 667–691, 1986.

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

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

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

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

[35] 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.

[36] 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.

[37] 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.

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

[39] S. Minato, "Zero-Suppressed BDDs for Set Manipulation in Combinatorial Problems," in *ACM/IEEE 30th Design Automation Conference (DAC)*, 1993, pp. 272–277.

[40] 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.

[41] ——, "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.

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

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