Enhancing Logic Synthesis of Switching Lattices by Generalized Shannon Decomposition Methods

Anna Bernasconi[a], Valentina Ciriani[b], Luca Frontini[b], Valentino Liberali[c], Gabriella Trucco[b], Tiziano Villa[d]

[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
[c]*Dipartimento di Fisica Università degli Studi di Milano, Italy, valentino.liberali@unimi.it*
[d]*Dipartimento di Informatica, Università degli Studi di Verona, Italy, tiziano.villa@univr.it*

Abstract

In this paper we propose a novel approach to the synthesis of minimal-sized lattices, based on the decomposition of logic functions. Since the decomposition allows to obtain circuits with a smaller area, our idea is to decompose the Boolean functions according to generalizations of the classical Shannon decomposition, then generate the lattices for each component function, and finally implement the original function by a single composed lattice obtained by glueing together appropriately the lattices of the component functions. In particular we study the two decomposition schemes defining the bounded-level logic networks called P-circuits and EXOR-Projected Sums of Products (EP-SOPs). Experimental results show that about 34% of our benchmarks achieve a smaller area when implemented using the P-circuit decomposition for switching lattices, with an average gain of at least 25%, and about 27% of our benchmarks achieve a smaller area when implemented using the EP-SOP decomposition, with an average gain of at least 22%.

Keywords: Logic synthesis for emerging technologies, Switching lattices, Generalized Shannon decomposition

1. Introduction

A switching lattice is a two-dimensional lattice of four-terminal switches linked to the four neighbors of a lattice cell, so that these are either all connected, or disconnected. A Boolean function can be implemented by a lattice associating each four-terminal switch to a Boolean literal, so that if the literal takes the value 1 the corresponding switch is ON and connected to its four neighbors, otherwise 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 (see Figure 1 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 old and dates back to a seminal paper by Akers in 1972 [1]. Recently, with the advent of a variety of emerging nanoscale technologies based on regular arrays of switches, synthesis methods targeting lattices of multi-terminal switches have found a renewed interest [2, 3, 4]. Consider for instance a nanowire array, where each crosspoint is controlled by an input voltage. In this paper, we consider crosspoints that behave like four-terminal switches controlled by an input signal and therefore the proposed nanowire crossbar array can be modeled as a lattice of four-terminal switches. Note that, in general, crossbars can also be modeled by programmable contacts (see for instance [5]). Nanowire crossbar arrays may offer substantial advantages over conventional CMOS when used to implement programmable architectures. Conventional implementations typically employ SRAMs for programming crosspoints; other techniques have been suggested for implementing programmable crosspoints such as bistable switches that form memory cores, molecular switches and solid-electrolyte nanoswitches (see [3] for more details and bibliographic references).

In this paper we show how the cost of implementing a switching lattice could be mitigated by applying Boolean function decomposition techniques in lattice-based implementations. Decomposition of logic functions is a widely studied field in multi-level logic; here we focus on two particular decomposition methods that are based on different generalizations of the classical Shannon decomposition and that give rise to the families of bounded-level logic networks called *P-circuits* [6, 7, 8, 9] and *EXOR-Projected Sums of Products* (EP-SOPs) [10, 11, 12, 13]. These two methods have been selected, among other known decomposition techniques, mainly because Shannon-based decomposition methods allow to keep the number of logic levels bounded, in addition to the fact that they have been already exploited with good results in CMOS technology.

In the framework of switching lattices synthesis, where the available minimization tools are not yet as developed and mature as those available for CMOS technology, reducing the synthesis of a target Boolean function to the synthesis of smaller functions could represent a very beneficial approach. This expectation has been confirmed by our experimental results, which demonstrate that in about 35%

of the analyzed cases the synthesis of switching lattices based on a decomposition of the logic function into smaller sub-functions allows to obtain a smaller area in the final resulting lattice.

This paper is an extended version of the conference paper in [14], where only the decomposition with P-circuits was described, and is organized as follows. Preliminaries on switching lattices are reviewed in Section 2, while P-circuits and EP-SOP forms are described in Section 3. Section 4 shows how the proposed decomposition schemes can be exploited for the synthesis of switching lattices. Section 5 provides the experimental results and Section 6 concludes the paper.

2. Switching Lattices

In this section we briefly review some basic notions and results on switching lattices [1, 3, 4]

A switching lattice is a two-dimensional array of four-terminal switches. The four terminals of the switch link to the four neighbours 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 neighbours, 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 1 (a) corresponds to the lattice form depicted in Figure 1 (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 1 (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 define any path from the top to the bottom edge (Figure 1 (d)).

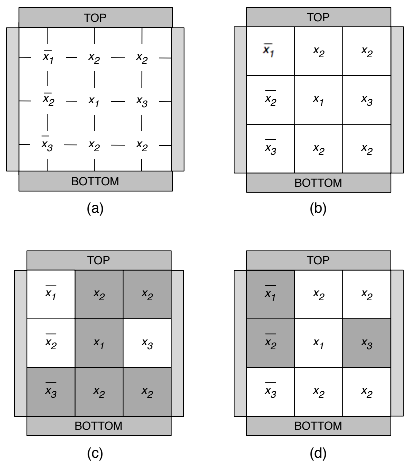

TOP
BOTTOM
$\overline{x}_1$ x_2 x_2
$\overline{x}_2$ x_1 x_3
$\overline{x}_3$ x_2 x_2
(a)
(b)
(c)
(d)

Figure 1: 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.

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 [2, 3] 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: $SOP(f) = p_1 + p_2 + \cdots + p_s$ and $SOP(f^D) = q_1 + q_2 + \cdots + q_r$;
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 1 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 [4], the authors 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 else a preset time limit has been exceeded. Experimental results show how this alternative method can decrease lattice sizes considerably. In this approach the use of fixed inputs is allowed, moreover the lattice considers only the top-to-bottom paths and implements the function f, but not its dual.

3. P-circuits and EP-SOP forms

We now review two slightly different bounded-level logic networks called *P-circuits* and *EXOR-Projected Sums of Products forms* (EP-SOP). Both networks are based on generalizations of the standard Shannon decomposition, and can be seen as special logic architectures which realize a Boolean function by projecting it onto overlapping subsets. They were introduced in [6, 7, 8] and in [10, 11, 12], and further studied in [9, 13].

We first give some preliminary definitions.

A *completely specified Boolean function* f is a function $f : \{0,1\}^n \to \{0,1\}$. A completely specified Boolean function can also be interpreted as the set of points $x \in \{0,1\}^n$ such that $f(x) = 1$.

An *incompletely specified Boolean function* is a function $f : \{0,1\}^n \to \{0,1,-\}$, where $-$ is called the don't care value of the function. An incompletely specified function can be described by three sets of points: the *on-set*, the *off-set* and the *don't care set*, which characterize the points in $\{0,1\}^n$ with images 0, 1, and $-$, respectively.

Given the Boolean space $\{0,1\}^n$ described by the set $\{x_1,\ldots,x_n\}$ of n binary variables, a *literal* is a variable or its complement; a *cube* is conjunction (or product) of a set of literals, and a *minterm* is a cube when it represents only one point, i.e., when it is a conjunction of n distinct literals. Finally, a *multiple-output Boolean function* f is a function $f : \{0,1\}^n \to \{0,1,-\}^m$; it can be considered also as a vector of Boolean functions $\{f_1, f_2, \ldots, f_m\}$.

P-circuits and EP-SOPs 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 the 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).

More precisely, let f be a completely specified Boolean function depending on the set $\{x_1,\ldots,x_n\}$ of n binary variables. Consider the classical Shannon decomposition

$$f = \overline{x}_i f|_{\overline{x}_i} + x_i f|_{x_i}\,,$$

and the more general EXOR-based decomposition [15] [16]

$$f = (\overline{x}_i \oplus p) f|_{x_i=p} + (x_i \oplus p) f|_{x_i \neq p}\,,$$

that corresponds to the classical one when $p = 0$ (as before, p is a function non-depending on x_i). This decomposition partitions the Boolean space $\{0,1\}^n$ into two subsets: the subset of points where $x_i = p$ and the subset of points where $x_i \neq p$. The characteristic functions of these two subsets are $(\overline{x}_i \oplus p)$ and $(x_i \oplus p)$, respectively.

Note that these two subsets have always the same cardinality, for any function p non-depending on x_i. This is due to the presence of the EXOR operator in their characteristic functions: indeed, for any minterm $x \in \{0,1\}^n$, x belongs to the subset where $x_i = p$, i.e., the subset where $(\overline{x}_i \oplus p) = 1$, if and only if the minterm obtained complementing the i-th bit of x belongs to the subset where $x_i \neq p$, i.e., where $(x_i \oplus p) = 1$. The cofactors $f|_{x_i=p}$ and $f|_{x_i \neq p}$ correspond to the projections of f onto the two subsets with $x_i = p$ and $x_i \neq p$, respectively. Observe that this decomposition is suitable for keeping x_i disjoint from the rest of the circuit, but is not oriented to area minimization. In fact, $f|_{x_i=p}$, and $f|_{x_i \neq p}$ do not depend on the variable x_i, but the cubes of f intersecting both subsets $x_i = p$ and $x_i \neq p$ may be split into two smaller subcubes when they are projected onto $f|_{x_i=p}$, and $f|_{x_i \neq p}$, respectively.

P-circuits and EP-SOPs try to overcome this problem in different ways.

3.1. P-circuits

The main idea in P-circuits synthesis is to keep unprojected some of the points of the original function. For this purpose, let $I = f|_{x_i=p} \cap f|_{x_i \neq p}$ be the intersection of the two cofactors $f|_{x_i=p}$ and $f|_{x_i \neq p}$. Note that the intersection I contains the cubes whose products do not contain x_i and that cross the two sets. In order to overcome the splitting of these crossing cubes, we could keep I unprojected, and project only the minterms in $f|_{x_i=p} \setminus I$ and $f|_{x_i \neq p} \setminus I$, obtaining the expression

$$f = (\overline{x}_i \oplus p)(f|_{x_i=p} \setminus I) + (x_i \oplus p)(f|_{x_i \neq p} \setminus I) + I\,.$$

Note that p, $f|_{x_i=p} \setminus I$, $f|_{x_i \neq p} \setminus I$ and I do not depend on x_i. However, the points that are in I could be exploited to form bigger cubes in the projected sets. Therefore, if a point is in I and it is useful for a better minimization of the projected parts, it can be kept both in the projection and in the intersection. Moreover, if a point is covered in both the projected sets, it is not necessary to cover it in the intersection. From these observations, we can infer that the projected sub-circuits should cover at least $f|_{x_i=p} \setminus I$ and $f|_{x_i \neq p} \setminus I$, and must be contained in $f|_{x_i=p}$ and $f|_{x_i \neq p}$, respectively. Moreover, the part of the circuit that is not projected should be contained in the intersection I.

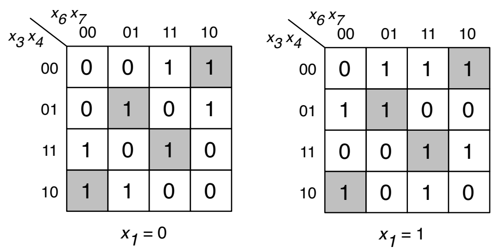
In summary, we can define a P-circuit as follows, where $S(f)$ indicates a SOP circuit implementing a Boolean function f.

Definition 1 ([9]). *A* P-circuit *of a completely specified function f is the circuit $P(f)$ denoted by the expression:*

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

where

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



$x_3x_4 \backslash x_6x_7$	00	01	11	10
00	0	0	1	▨ 1
01	0	▨ 1	0	1
11	1	0	▨ 1	0
10	▨ 1	1	0	0

$x_1 = 0$

$x_3x_4 \backslash x_6x_7$	00	01	11	10
00	0	1	1	▨ 1
01	1	▨ 1	0	0
11	0	0	▨ 1	1
10	▨ 1	0	1	0

$x_1 = 1$

Figure 2: Karnaugh map of the benchmark $z = z4(2)$. The cells in grey show the minterms that belong to the intersection between the cofactors $z|_{x_1=0}$ and $z|_{x_1\neq 0}$.

3. $\emptyset \subseteq f^I \subseteq I$
4. $P(f) = f$.

This definition can be easily generalized to incompletely specified Boolean functions $f = \{f^{on}, f^{dc}\}$. For the sake of simplicity, suppose that $f^{on} \cap f^{dc} = \emptyset$; otherwise, following the usual semantics, we consider $f^{on} \setminus f^{dc}$ as the on-set of f. Let I be the intersection of the projections of f onto the two sets $x_i = p$ and $x_i \neq p$:

$$I = (f^{on}|_{x_i=p} \cup f^{dc}|_{x_i=p}) \cap (f^{on}|_{x_i\neq p} \cup f^{dc}|_{x_i\neq p}).$$

Definition 2 ([9]). *A* P-circuit *of an incompletely specified function $f = \{f^{on}, f^{dc}\}$ is the circuit $P(f)$ denoted by the expression:*

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

where

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

As an example of P-circuit decomposition, let us consider the benchmark $z4$ taken from LGSynth93 [17], and in particular its third output $z4(2)$, simply denoted by z. This function, represented by the Karnaugh map in Figure 2, depends on seven variables $(x_1, x_2, \ldots, x_7)$, two of which (x_2 and x_5) are nonessential. Consider the P-circuit decomposition with respect to the first variable x_1 and to

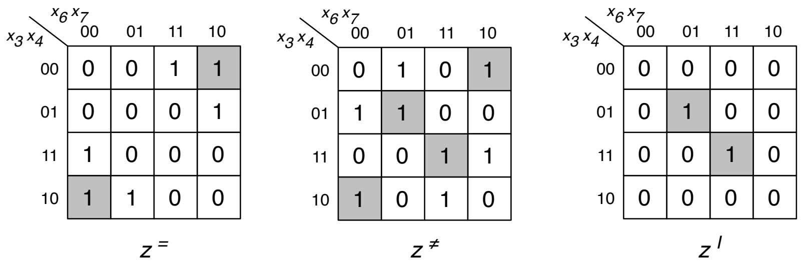


Figure 3: Karnaugh maps of the sets $z^=$, $z^{\neq}$, and z^I defining an optimal P-circuit for $z = z4(2)$ with respect to x_1 and to the function $p = 0$. The minterms in the intersection between $z|_{x_1=0}$ and $z|_{x_1\neq 0}$, now distributed among $z^=$, $z^{\neq}$, and z^I, are highlighted in grey.

the projection function $p = 0$. The three sets $z^=$, $z^{\neq}$, and z^I that define an optimal P-circuit representation of $z = z4(2)$, with respect to x_1 and to the function $p = 0$, are depicted in Figure 3. Observe that the two minterms 0010 and 1000 in the intersection between the cofactors $z|_{x_1=0}$ and $z|_{x_1\neq 0}$ have been removed from z^I since they have been kept in both sets $z^=$ and $z^{\neq}$ in order to get smaller SOP forms. Moreover, the other two points of the intersection, 0101 and 1111, are kept both in z^I and in the projection $z^{\neq}$, where they are used to form bigger cubes. The P-circuit representation of $z = z4(2)$ is then given by the expression

$$P(z) = \overline{x}_1\, S(z^=) + x_1\, S(z^{\neq}) + S(z^I)$$

where $S(z^=) = \overline{x}_3\overline{x}_4x_6 + x_3\overline{x}_4\overline{x}_6 + \overline{x}_3x_6\overline{x}_7 + x_3\overline{x}_6\overline{x}_7$, $S(z^{\neq}) = x_3x_4x_6 + \overline{x}_3x_4\overline{x}_6 + x_3x_6x_7 + \overline{x}_3\overline{x}_6x_7 + \overline{x}_3\overline{x}_4x_6\overline{x}_7 + x_3\overline{x}_4\overline{x}_6\overline{x}_7$, and $S(z^I) = x_3x_4x_6x_7 + \overline{x}_3x_4\overline{x}_6x_7$ are the SOP representations of $z^=$, $z^{\neq}$, and z^I, respectively.

As we have seen, the idea for synthesis of P-circuits is to construct a network for f by appropriately choosing the sets $f^=$, $f^{\neq}$, and f^I as building blocks. Several algorithms for performing such a choice have been studied and proposed in the literature [6, 7, 8, 9]. In particular, in [9], it is shown how the structural flexibility of P-circuits can be completely characterized by using Boolean relations, and how the associated optimal P-circuit decomposition problem, with respect to a given variable x_i and a function p, can be efficiently solved using a Boolean relation minimizer.

3.2. EP-SOP forms

Let us now consider the alternative decomposition methods leading to the definition of *EXOR-Projected Sums of Products* (EP-SOPs). These forms are derived as a special case of the generalized Shannon decomposition with *remainder*

$$f = (\overline{x}_i \oplus p)(f|_{x_i=p} \setminus R) + (x_i \oplus p)(f|_{x_i \neq p} \setminus R) + R$$

that restructures a logic function into subsets of points defined by the generalized cofactors with a remainder R containing the cubes that we do not want to split. In particular, EP-SOP forms correspond to the special case $p = x_j$, with $i \neq j$:

$$f = (\overline{x}_i \oplus x_j)(f|_{x_i=x_j} \setminus R) + (x_i \oplus x_j)(f|_{x_i \neq x_j} \setminus R) + R\,.$$

Note that, while the intersection set I in the P-circuit decomposition scheme does not depend on the variable x_i, the remainder R depends in general on all the input variables and is defined as the set of all minterms of f that could form a *crossing cube*, i.e., a cube of f intersecting both subsets $x_i = x_j$ and $x_i \neq x_j$. In the particular case $p = x_j$, crossing cubes can be precisely identified as those cubes described by products that do not depend on both x_i and x_j; thus the remainder R contains all points x in the on-set of f such that f takes the value 1 on at least one of the two points obtained complementing in x the i-th or the j-th variable. In other words, R contains all cubes described by products of literals

1. that depend on x_i, but not on x_j;
2. that depend on x_j, but not on x_i;
3. that depend neither on x_i, nor on x_j.

If we denote with $x^{(k)}$ the point obtained complementing the k-th bit of x, we get

$$R = \{x \mid f(x) = 1 \wedge (f(x^{(i)}) = 1 \vee f(x^{(j)}) = 1)\}\,.$$

Moreover, the two cofactors ($f|_{x_i=x_j}$ and $f|_{x_i \neq x_j}$) can be equivalently defined as incompletely specified Boolean functions depending on all input variables, in the following way: 1) $f^{on}|_{x_i=x_j}$ ($f^{on}|_{x_i \neq x_j}$) is the on-set of the original function f such that $x_i = x_j$ (resp. $x_i \neq x_j$); 2) $f^{dc}|_{x_i=x_j}$ ($f^{dc}|_{x_i \neq x_j}$) is the set of points such that $x_i \neq x_j$ (resp. $x_i = x_j$). These don't cares can be inserted in the cofactor $f|_{x_i=x_j}$ since, in the decomposition, this function is multiplied by $(\overline{x}_i \oplus x_j)$, which evaluates to 0 when $x_i \neq x_j$. A symmetric observation holds for $f|_{x_i \neq x_j}$. In the present paper, we will adopt this alternative definition, so that all three functions occurring in the decomposition may depend on all input variables (x_i included).

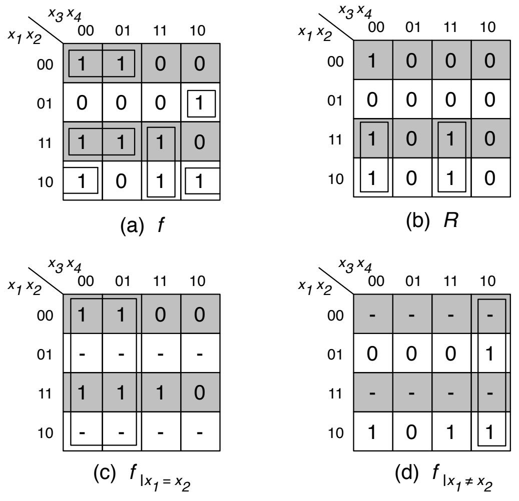


Figure 4: (a): A minimal SOP form for the function f. (b), (c) and (d): the remainder R, and the cofactors $f|_{x_1=x_2}$ and $f|_{x_1\neq x_2}$, together with the corresponding covers $f^R = x_1\overline{x}_3\overline{x}_4 + x_1x_3x_4$, $f^= = \overline{x}_3$, and $f^{\neq} = x_3\overline{x}_4$, used in a minimal EP-SOP expression for f with respect to the pair of variables x_1 and x_2.

For example, consider the function f in Figure 4(a), $i = 1$ and $j = 2$. In the figure, the subset of the Boolean space where $x_1 = x_2$ ($x_1 \neq x_2$, resp.) is depicted in gray (white, resp.). Figure 4(b) shows the remainder for f. Finally, the (non-projected) cofactors $f|_{x_1=x_2}$ and $f|_{x_1\neq x_2}$ are represented in Figures 4(c) and 4(d), respectively. Note that $f|_{x_1=x_2}$ corresponds to f for the points where $x_1 = x_2$ and contains don't care conditions where $x_1 \neq x_2$. These don't cares can be inserted in $f|_{x_1=x_2}$ since, in the decomposition, this function is multiplied by $(\overline{x}_1 \oplus x_2)$, which evaluates to 0 when $x_1 \neq x_2$. A symmetric observation holds for $f|_{x_1\neq x_2}$.

A clear advantage of this representation is that don't care points can be used to form bigger cubes. Consider, for instance, the two distinct cubes $\overline{x}_1\overline{x}_2\overline{x}_3$ and $x_1x_2\overline{x}_3$ in Figure 4(a). In the function $f|_{x_1=x_2}$, the corresponding cubes are merged together in a bigger cube, i.e., $\overline{x}_3$, using don't cares, as shown in Figure 4(c). Instead, the cube $x_1x_3x_4$ in Figure 4(a) is an example of crossing cube: it is split into two minterms: $x_1x_2x_3x_4$ in Figure 4(c) and $x_1\overline{x}_2x_3x_4$ in Figure 4(d) that are covered by two different cubes (x_1x_4 and x_1x_3, resp.). Observe that, if we keep and cover this crossing cube only in the remainder R, then we do not need to cover its minterms in the two cofactors; thus, the cubes x_1x_4 and x_1x_3 will not appear in the final EP-SOP expression.

As for the P-circuit scheme, we can observe that the minterms in the remainder R could be exploited to form bigger cubes in the projected sets, i.e., if a point is in R and is useful for a better minimization of one of the cofactors, it can be kept both in the projected cofactor and in the remainder. Therefore, in order to cover any point of the function at least once and get a minimal decomposition form, with respect to the variables x_i and x_j, we can decide to cover any minterm x in the remainder: 1) only in the remainder, 2) only in the corresponding cofactor (i.e., $f|_{x_i=x_j}$ if $x_i = x_j$ or $f|_{x_i\neq x_j}$ if $x_i \neq x_j$), 3) both in the remainder and in the corresponding cofactor. The last choice can be convenient when x is useful for forming bigger cubes for both the remainder and the cofactor. For instance, in the running example, the point 1100 can be used in $f|_{x_1=x_2}$ to form the cube $\overline{x}_3$ and in the remainder R to form the cube $x_1\overline{x}_3\overline{x}_4$ with the point 1000.

In summary, we can rephrase the definition of an EP-SOP-circuit given in [11] as follows (as before, $S(f)$ indicates a SOP circuit implementing a Boolean function f).

Definition 3 ([13]). *An* EP-SOP-decomposition*, with respect to the variables x_i and x_j, of a completely specified function f is the expression:*

$$\textit{EP-SOP}(f) = (\overline{x}_i \oplus x_j)\, S(f^=) + (x_i \oplus x_j)\, S(f^{\neq}) + S(f^R)$$

where

1. $(f^{on}|_{x_i=x_j} \setminus R) \subseteq f^= \subseteq f^{on}|_{x_i=x_j} \cup f^{dc}|_{x_i=x_j}$
2. $(f^{on}|_{x_i\neq x_j} \setminus R) \subseteq f^{\neq} \subseteq f^{on}|_{x_i\neq x_j} \cup f^{dc}|_{x_i\neq x_j}$
3. $\emptyset \subseteq f^R \subseteq R$
4. *EP-SOP*$(f) = f$.

For our running example in Figure 4, we get the EP-SOP form EP-SOP$(f) = (\overline{x}_1 \oplus x_2)(\overline{x}_3) + (x_1 \oplus x_2)(x_3\overline{x}_4) + (x_1\overline{x}_3\overline{x}_4 + x_1x_3x_4)$, that contains fewer and bigger cubes than a classical minimal SOP cover $f = \overline{x}_1\overline{x}_2\overline{x}_3 + \overline{x}_1x_2x_3\overline{x}_4 + x_1x_2\overline{x}_3 + x_1x_3x_4 + x_1\overline{x}_2\overline{x}_4$. Note that any point of the function is covered at least once, by $f^=$, $f^{\neq}$, or f^R. For example, 0000 is covered by $f^=$, 1100 is covered by both $f^=$ and f^R, and 1000 is covered by f^R (note that 1000 is also covered by $f^=$ but not by $(\overline{x}_1 \oplus x_2)f^=$ in the final form).

This definition can be generalized to incompletely specified Boolean functions $f = \{f^{on}, f^{dc}\}$. When f is an incompletely specified Boolean function, $f|_{x_i=x_j}$ and $f|_{x_i\neq x_j}$ can be defined as follows: 1) $f^{on}|_{x_i=x_j}$ ($f^{on}|_{x_i\neq x_j}$) contains the points of f^{on} such that $x_i = x_j$ (resp. $x_i \neq x_j$); 2) $f^{dc}|_{x_i=x_j}$ ($f^{dc}|_{x_i\neq x_j}$) contains the points of f^{dc} such that $x_i = x_j$ (resp. $x_i \neq x_j$) together with the points such that $x_i \neq x_j$ (resp. $x_i = x_j$).The definition of the remainder R can be immediately generalized to incompletely specified Boolean functions, noting that the points potentially included in a crossing cube can now be defined as the points x in the on-set or in the dc-set of f, such that the two points obtained complementing in x the i-th and the j-th variable are not both included in the off-set.

Definition 4 ([13]). *An* EP-SOP-decomposition*, with respect to the variables x_i and x_j, of an incompletely specified function $f = \{f^{on}, f^{dc}\}$ is the expression:*

$$\textit{EP-SOP}(f) = (\overline{x}_i \oplus x_j)\, S(f^{=}) + (x_i \oplus x_j)\, S(f^{\neq}) + S(f^R)$$

where

$$R = \{x \in f^{on} \cup f^{dc} \,|(x^{(i)} \in f^{on} \cup f^{dc}) \vee (x^{(j)} \in f^{on} \cup f^{dc})\}$$

1. $(f^{on}|_{x_i=x_j} \setminus R) \;\subseteq\; f^{=} \;\subseteq\; f^{on}|_{x_i=x_j} \cup f^{dc}|_{x_i=x_j}$
2. $(f^{on}|_{x_i \neq x_j} \setminus R) \;\subseteq\; f^{\neq} \;\subseteq\; f^{on}|_{x_i \neq x_j} \cup f^{dc}|_{x_i \neq x_j}$
3. $\emptyset \;\subseteq\; f^R \;\subseteq\; R$
4. $f^{on} \;\subseteq\; \textit{EP-SOP}(f) \;\subseteq\; f^{on} \cup f^{dc}$.

As for P-circuits, the problem of EP-SOP synthesis can be nicely formalized and efficiently solved using Boolean relations, as discussed and proved in [13].

We finally observe that both decomposition techniques could be recursively applied to the cofactors $f^{=}$, $f^{\neq}$, and f^I or f^R, but this leads to an increase in the number of logic levels. Thus, this recursive approach could be very interesting for the synthesis of unbounded multilevel forms.

4. Decomposition with lattices

In this section we will discuss how the decomposition of Boolean functions can be exploited in the synthesis of switching lattices. 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 separate lattices, or 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 easier and should produce lattice implementations of smaller size.

Decomposition of logic functions is a widely studied field in multi-level logic, see [18] for a review on this subject. Here we focus on the decomposition method

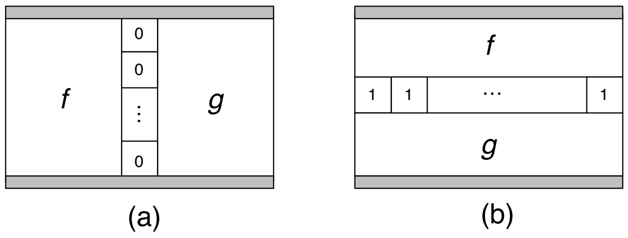


Figure 5: Lattice implementation of $f + g$ (a) and of $f \cdot g$ (b).

that gives rise to the bounded-level logic networks called P-circuits, described in Section 3.

First of all, we recall from [4] 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 $f \cdot g$ using a padding column of 0s and a padding row of 1s, respectively, as shown in Figure 5. Indeed, 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. Of course, 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, 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.

We can use these simple composition rules to derive a lattice describing a P-circuit expression $P(f) = (\overline{x}_i \oplus S(p))\, S(f^{=}) + (x_i \oplus S(p))\, S(f^{\neq}) + S(f^{I})$ for a given function f, using lattices for the three sets $f^{=}$, $f^{\neq}$, and f^{I} and for the projection functions $(\overline{x}_i \oplus p)$ and $(x_i \oplus p)$ as building blocks, as depicted in Figure 6.

We now formally prove that the lattice implementation of a P-circuit $P(f)$ described in Figure 6 correctly implements the function f.

Theorem 1. *Let f be a Boolean function depending on n binary variables, and let*

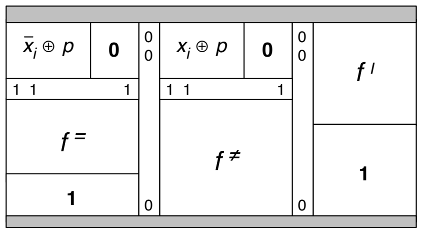


Figure 6: Lattice implementation of a P-circuit.

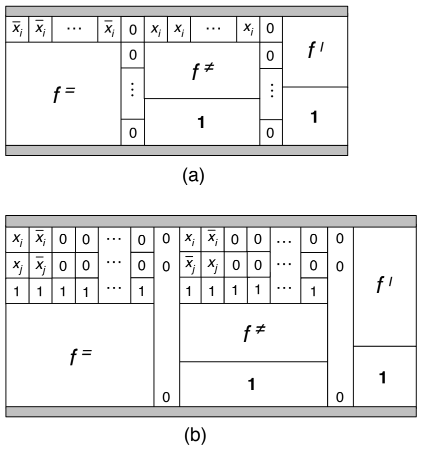


Figure 7: Lattice implementation of a P-circuit with projection function $p = 0$ (a) and with projection function $p = x_j$ (b).

$P(f) = (\overline{x}_i \oplus S(p))\, S(f^=) + (x_i \oplus S(p))\, S(f^{\neq}) + S(f^I)$ be a P-circuit representing f. The lattice obtained composing the lattices for the three sets $f^=$, $f^{\neq}$, and f^I and for the projection functions $(\overline{x}_i \oplus p)$ and $(x_i \oplus p)$, as shown in Figure 6, implements the function f.

Proof. In order to prove the theorem, we need to show that the function f evaluates to 1 on a given input assignment if and only if there exists a connected path between the top and the bottom edge of the lattice in Figure 6.

First suppose that f evaluates to 1 on a given input $(x_1, x_2, \ldots, x_n)$. Then, at least one of the three terms in the expression for $P(f)$ must evaluate to 1. Suppose that the first term $(\overline{x}_i \oplus S(p))\, S(f^=)$ takes the value 1 on $(x_1, x_2, \ldots, x_n)$. Then, both $(\overline{x}_i \oplus S(p))$ and $S(f^=)$ are equal to 1, giving rise to a top-to-bottom connected path in the left side of the lattice. An analogous situation arises if the second or the third term are equal to 1.

Now suppose that a given input assignment $(x_1, x_2, \ldots, x_n)$ induces some connected top-to-bottom paths on the lattice. Due to the presence of the two columns of 0s, separating the left, center, and right portions of the lattice corresponding to the implementations of the functions $(\overline{x}_i \oplus p)\, f^=$, $(x_i \oplus p)\, f^{\neq}$, and f^I, respectively, each connected path is entirely contained in one of the three portions. This implies that at least one of the three terms in input to the final OR gate of the P-circuit representing f is equal to 1, and the thesis immediately follows. ■

The lattice description of the P-circuit can be simplified depending on the chosen projection function. For instance, if we choose the P-circuit $P(f) = \overline{x}_i\, S(f^=) + x_i\, S(f^{\neq}) + S(f^I)$ based on the generalization of the classical Shannon decomposition with projection function $p = 0$, which experimentally represents a very efficient and effective solution, we get the lattice shown in Figure 7 (a), where the two padding rows of 1s have been substituted with one row of cells assigned to the literal $\overline{x}_i$ and one row of cells assigned to x_i. Figure 7 (b) shows the lattice implementation of the P-circuit $P(f) = (\overline{x}_i \oplus x_j)\, S(f^=) + (x_i \oplus x_j)\, S(f^{\neq}) + S(f^I)$ corresponding to the choice $p = x_j$. Both lattices correctly implement the function f, as proved in the following corollary.

Corollary 1. *The two lattices in Figure 7 implement the function f through its P-circuit representations with projection functions $p = 0$ and $p = x_j$, respectively.*

Proof. Let us consider the first lattice, corresponding to the P-circuit representation of f with projection function $p = 0$. As before, observe that each top-to-bottom connected path must be entirely contained in one of the three portions of the lattice, because of the two padding columns of 0s. Moreover, the row of cells assigned to $\overline{x}_i$ on top of the lattice for $f^=$ allows to derive a top-to-bottom connected path for f from a connected path for $f^=$ if and only if $x_i = 0$; analogously, the row of cell assigned to x_i on top of the lattice for $f^{\neq}$ allows to derive a top-to-bottom connected path for f from a connected path for $f^{\neq}$ if and only if $x_i = 1$. Finally, any connected path for f^I can be extended to a connected path for f. Thus the thesis immediately follows from the definition of P-circuit and of the terms $f^=$, $f^{\not=}$, and f^I (see Definitions 1 and 2).

Now consider the lattice in Figure 7 (b), corresponding to the P-circuit for f with projection function $p = x_j$. Any connected path for f^I can be extended to a connected path for f. Moreover, any connected path for $f^=$ can be extended to a connected path for f if and only if both variables x_i and x_j assume the same value, i.e., $x_i = x_j$, while any connected path for $f^{\neq}$ can be extended to a connected path for f if and only if the two variables assume different values, i.e., $x_i = \overline{x}_j$. Since the presence of the two columns of 0s prevents the existence of top-to-bottom connected paths intersecting different regions of the lattice, the thesis immediately follows from Definitions 1 and 2. ■

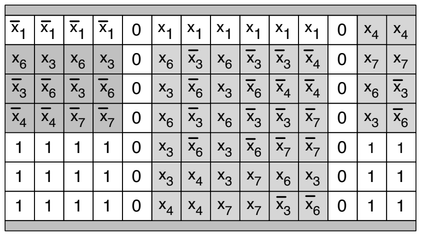
As an example of synthesis of lattices based on the P-circuit decomposition, let us consider the third output, here denoted by z, of the benchmark $z4$ taken from LGSynth93 [17], whose P-circuit decomposition is shown in Figure 3 and discussed in Section 3.1. We can derive a lattice implementation of z using lattices for the three subfunctions $z^=$, $z^{\neq}$, and z^I as building blocks, as depicted in Figure 7 (a). Recall from Section 3.1 that the SOP representations of the three subfunctions are

$$\begin{aligned}
S(z^=) &= \overline{x}_3\overline{x}_4x_6 + x_3\overline{x}_4\overline{x}_6 + \overline{x}_3x_6\overline{x}_7 + x_3\overline{x}_6\overline{x}_7\,,\\
S(z^{\neq}) &= x_3x_4x_6 + \overline{x}_3x_4\overline{x}_6 + x_3x_6x_7 + \overline{x}_3\overline{x}_6x_7 + \overline{x}_3\overline{x}_4x_6\overline{x}_7 + x_3\overline{x}_4\overline{x}_6\overline{x}_7\,,\\
S(z^I) &= x_3x_4x_6x_7 + \overline{x}_3x_4\overline{x}_6x_7\,,
\end{aligned}$$

and contain 4, 6, and 2 products, respectively. The SOP expressions for the corresponding dual functions are

$$\begin{aligned}
S(z^{=^D}) &= x_3x_6 + \overline{x}_3\overline{x}_6 + \overline{x}_4\overline{x}_7\,,\\
S(z^{\neq^D}) &= \overline{x}_3\overline{x}_4x_6 + x_3\overline{x}_4\overline{x}_6 + \overline{x}_3x_6\overline{x}_7 + x_3\overline{x}_6\overline{x}_7 + x_3x_4x_6x_7 + \overline{x}_3x_4\overline{x}_6x_7\,,\\
S(z^{I^D}) &= x_4 + x_7 + \overline{x}_3x_6 + x_3\overline{x}_6\,,
\end{aligned}$$

with 3, 6, and 4 products, respectively. Using the method described in [3], we can compute the three sublattices for $z^=$, $z^{\neq}$, and z^I, whose dimensions are 3×4, 6×6, and 4×2. Finally, composing these three sublattices as shown in Figure 8, we



$\overline{x}_1$	$\overline{x}_1$	$\overline{x}_1$	$\overline{x}_1$	0	x_1	x_1	x_1	x_1	x_1	x_1	0	x_4	x_4
x_6	x_3	x_6	x_3	0	x_6	$\overline{x}_3$	x_6	$\overline{x}_3$	$\overline{x}_3$	$\overline{x}_4$	0	x_7	x_7
$\overline{x}_3$	$\overline{x}_6$	$\overline{x}_3$	$\overline{x}_6$	0	x_3	$\overline{x}_6$	x_3	$\overline{x}_6$	$\overline{x}_4$	$\overline{x}_4$	0	x_6	$\overline{x}_3$
$\overline{x}_4$	$\overline{x}_4$	$\overline{x}_7$	$\overline{x}_7$	0	x_6	$\overline{x}_3$	x_6	$\overline{x}_3$	$\overline{x}_3$	$\overline{x}_7$	0	x_3	$\overline{x}_6$
1	1	1	1	0	x_3	$\overline{x}_6$	x_3	$\overline{x}_6$	$\overline{x}_7$	$\overline{x}_7$	0	1	1
1	1	1	1	0	x_3	x_4	x_3	x_7	x_6	x_3	0	1	1
1	1	1	1	0	x_4	x_4	x_7	x_7	$\overline{x}_3$	$\overline{x}_6$	0	1	1

Figure 8: Lattice for the benchmark $z = z4(2)$ based on the P-circuit decomposition. The sublattices for $z^{=}$, $z^{\neq}$, and z^{I} are highlighted in grey.

get an overall lattice of dimension 7×14 for the benchmark $z = z4(2)$. Note that the synthesis method in [3] applied directly to z, without exploiting its P-circuit decomposition, would produce a lattice of dimension 12×12.

As already observed, the main idea behind this approach is that lattice synthesis of the subfunctions $f^{=}$, $f^{\neq}$, and f^{I}, which depend on $n-1$ variables instead of n and have a smaller on-set than f, should be an easier task, and should produce lattices of reduced size, so that the overall lattice for f - derived using minimal lattices for $f^{=}$, $f^{\neq}$, and f^{I} as building blocks - could be smaller than the one derived for f without exploiting its decomposition in P-circuits. This expectation has been confirmed by our experimental results (Section 5).

In a very similar way, we can synthesize on lattice a function f exploiting its EP-SOP decomposition form. The decomposed lattice has the same structure of the lattice shown in Figure 7 (b), with the lattice for the intersection f^{I} replaced with a lattice for the remainder f^{R}. Also recall that now the three building blocks $f^{=}$, $f^{\neq}$, and f^{R} depend on n variables, but contain fewer points than the original function f. The lattice obtained in this way correctly implements the function f, as stated in the following theorem, whose proof can be immediately derived from the proofs of Theorem 1 and Corollary 1.

Theorem 2. *Let f be a Boolean function depending on n binary variables, and let EP-SOP$(f) = (\overline{x}_i \oplus x_j)\, S(f^{=}) + (x_i \oplus x_j)\, S(f^{\neq}) + S(f^{R})$ be an EP-SOP form for f. The lattice obtained composing the lattices for $f^{=}$, $f^{\neq}$, f^{R}, and for the projection function $(\overline{x}_i \oplus x_j)$ implements the function f.*

We finally observe that, applying recursively these decomposition methods, we could further decompose each single lattice block. However, this approach requires the insertion of additional padding rows and/or columns of 1s or 0s, that in turn might lead to an increase of the dimension of the final overall lattice.

5. Experimental results

In this section we report the experimental results obtained by applying the decomposition with lattices described in Section 4. 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 (in PLA form) are taken from LGSynth93 [17]. We considered each output as a separate Boolean function, for a total of 1886 functions. Due to the limited space available, we report in the following only a significant subset of the functions as representative indicators of our experiments.

The algorithms for P-circuit and EP-SOP decomposition have been implemented in C, using the CUDD library for OBDDs [19, 20, 21, 22] to represent Boolean functions, and BREL [23] to solve Boolean relations, as detailed in [9, 13]. For the P-circuit decomposition model, we evaluate and report the results for both projection function $p = 0$, using $x_i = x_0$, i.e. we decompose with respect to the first input variable of each benchmark, and for $p = x_j$, using $p = x_1$, that is we decompose with respect to the subspace $x_0 \oplus x_1$ described by the first two variables. This choice of variables is arbitrary, as the main objective of this set of experiments is to evaluate how decomposition techniques allow to mitigate the cost of implementing switching lattices. However, we recall here that P-circuit decomposition should be performed with respect to critical signal that should be pushed closer to the outputs, i.e., we should choose as x_i the signal with the highest switching activity (to decrease power consumption), or with higher delay; on the other hand, the choice of the function p should be driven by other considerations, for instance area reduction.

EP-SOP decomposition, originally introduced mainly for area minimization, is performed in this experiments with respect to the pair of variables appearing together most frequently among the products of a minimal SOP for the target function, as suggested in [13].

In order to evaluate the utility of our approach, we compare our results with the ones obtained by the methods presented in [3] and in [4]. We used ESPRESSO to implement the method described in [3], and a collection of Python scripts for computing minimum-area lattices by transformation to a series of SAT problems, to simulate the results reported in [4].

Table 1: Proposed lattice sizes for standard benchmark circuits: a comparison of the proposed method with the results presented in [3] and in [4]. All cases where the SAT-based synthesis of a non-decomposed lattice (columns 4 and 5) is stopped after 10 minutes are marked with −. Results for decomposed lattices are marked with * when SAT-based synthesis [4] is stopped after 10 minutes and replaced with [3].

	Standard Synthesis				Decomp. with $p = x_0$				Decomp. with $p = x_0 \oplus x_1$				Decomp. EP-SOP			
	[3]		[4]		[3]		[4]		[3]		[4]		[3]		[4]	
	X×Y	A	X×Y	A	X×Y	A	X×Y	A	X×Y	A	X×Y	A	X×Y	A	X×Y	A
adr4(1)	36×36	1296	–	–	37×19	703	37×19	703*	37×21	777	37×21	777*	30×21	630	**25×19**	**475***
alu2(2)	11×10	110	**7×3**	**21**	13×7	91	10×6	60	14×8	112	11×8	88	12×11	132	8×6	48*
alu2(5)	14×13	182	–	–	**16×10**	**160**	**16×10**	**160***	17×10	170	17×10	170*	15×14	210	15×14	210*
alu2(6)	4×4	16	**4×3**	**12**	4×4	16	**4×3**	**12***	4×4	16	4×3	12*	4×7	28	4×6	24*
alu3(0)	**5×4**	**20**	–	–	7×4	28	7×4	28	8×6	48	8×6	48	7×5	35	7×5	35
alu3(1)	8×7	56	**4×3**	**12**	10×5	50	8×5	40	11×7	77	9×7	63	9×8	72	6×6	36*
b12(0)	4×6	24	**3×4**	**12**	4×6	24	**3×4**	**12***	6×7	42	6×7	42*	5×7	35	4×6	24*
b12(1)	7×5	35	**4×4**	**16**	7×5	35	**4×4**	**16***	8×7	56	6×7	42*	8×6	48	5×6	30*
b12(2)	7×6	42	**4×4**	**16**	8×6	48	8×5	40*	11×7	77	11×6	66	10×8	80	10×7	70
bcc(5)	9×27	243	–	–	9×26	234	9×26	234*	10×23	230	10×23	230*	**10×20**	**200**	**10×20**	**200***
bcc(7)	11×31	341	–	–	12×29	348	12×29	348*	12×28	336	12×28	336*	**12×26**	**312**	**12×26**	**312***
bcc(8)	12×31	372	–	–	13×29	377	13×29	377*	13×27	351	13×27	351*	**13×24**	**312**	**13×24**	**312***
bcc(12)	11×31	341	–	–	11×30	330	11×30	330*	12×28	336	12×28	336*	**12×25**	**300**	**12×25**	**300***
bcc(27)	19×38	741	–	–	20×33	660	20×33	660*	20×31	620	20×31	620*	**21×29**	**609**	**21×29**	**609***
bcc(43)	10×20	200	–	–	**11×16**	**176**	**11×16**	**176***	10×22	220	10×22	220*	12×20	240	12×20	240*
bcd.div3(1)	3×4	12	**3×3**	**9**	5×4	20	5×4	20	5×5	25	5×5	25*	4×5	20	4×5	20*
bcd.div3(2)	3×4	12	**3×3**	**9**	5×4	20	5×4	20	7×5	35	7×5	35	4×5	20	4×5	20*
bcd.div3(3)	3×5	15	**3×4**	**12**	4×4	16	4×4	16*	5×5	25	5×5	25*	5×4	20	5×4	20*
bench1(2)	24×45	1080	–	–	33×29	957	33×29	957*	31×27	837	31×27	837*	**31×23**	**682**	**31×22**	**682***
bench1(3)	16×31	496	–	–	20×18	360	21×18	378*	23×22	506	23×22	506*	**21×17**	**357**	**21×17**	**357**
bench1(5)	2×50	1350	–	–	**32×28**	896	32×28	896*	**28×22**	**616**	**28×22**	**616***	31×26	806	31×26	806*
bench1(6)	21×35	735	–	–	**26×24**	**624**	**26×24**	**624***	28×30	840	28×30	840*	26×26	676	26×26	676*
bench1(7)	21×43	903	–	–	**27×20**	**540**	**27×20**	**540***	28×28	784	28×28	784*	29×30	870	29×30	870*
bench1(8)	24×44	1056	–	–	31×26	806	31×26	806*	29×28	812	29×28	812*	26×24	624	25×24	600*
bench(6)	4×8	32	**3×4**	**12**	6×3	18	6×3	18	5×5	25	5×5	25*	8×6	48	8×6	48
br2(4)	**8×18**	**144**	–	–	**8×18**	**144**	**8×18**	**144***	8×20	160	8×20	160*	10×15	150	10×15	150*
br2(5)	**4×14**	**56**	–	–	**4×14**	**56**	**4×14**	**56***	4×16	64	4×16	64*	6×13	78	6×13	78*
br2(6)	**5×16**	**80**	–	–	**5×16**	**80**	**5×16**	**80***	5×18	90	5×18	90*	7×13	91	7×13	91*
clpl(2)	**2×2**	**4**	**2×2**	**4**	3×2	6	3×2	6*	7×5	35	7×5	35	5×4	20	5×4	20*
clpl(3)	6×6	36	**6×3**	**18**	9×6	54	9×6	54*	10×8	80	10×8	80*	10×9	90	10×6	60
clpl(4)	5×5	25	**5×3**	**15**	8×5	40	8×5	40*	9×7	63	9×7	63*	9×8	72	9×6	54
co14(0)	14×92	1288	–	–	15×80	1200	15×80	1200*	15×71	1065	15×71	1065*	**15×67**	**1005**	**15×67**	**1005**
dc1(0)	4×4	16	**3×3**	**9**	5×4	20	4×4	16*	5×6	30	5×6	30*	4×6	24	4×6	24
dc1(1)	**2×3**	**6**	**2×3**	**6**	3×3	9	3×3	9*	5×6	30	5×6	30*	5×6	30	5×6	30
dc1(4)	4×5	20	**3×4**	**12**	5×4	20	5×4	20*	7×6	42	7×6	42	7×6	42	7×6	42
dc1(6)	3×3	9	**3×2**	**6**	4×2	8	4×2	8*	7×5	35	7×5	35	3×6	18	3×5	15
dc2(4)	9×10	90	**4×5**	**20**	10×9	90	8×5	40*	13×12	156	9×7	63*	10×11	110	8×7	56*
dc2(5)	6×6	36	**2×6**	**12**	7×7	49	5×6	30*	10×8	80	6×7	42*	6×9	54	2×9	18*
dk17(0)	2×8	16	**2×6**	**12**	4×4	16	4×4	16*	5×6	30	5×6	30*	4×4	16	4×4	16
dk17(1)	2×8	16	**2×6**	**12**	4×4	16	4×4	16*	5×6	30	5×6	30*	4×4	16	4×4	16
dk17(3)	3×11	33	**2×7**	**14**	4×7	28	6×3	18*	7×7	49	7×6	42*	9×6	54	9×6	54
dk17(4)	3×9	27	**2×7**	**14**	6×4	24	6×4	24	8×6	48	8×6	48	6×7	42	6×6	36
dk27(6)	**1×2**	**2**	**1×2**	**2**	**1×2**	**2**	**1×2**	**2***	2×5	10	2×5	10*	1×3	3	1×3	3*
ex4(4)	**6×17**	**102**	–	–	**6×17**	**102**	**6×17**	**102***	**6×17**	**102**	**6×17**	**102***	6×20	120	6×20	120*
ex4(5)	**45×35**	**1575**	–	–	**45×35**	**1575**	**45×35**	**1575***	**45×35**	**1575**	**45×35**	**1575***	45×38	1710	45×38	1710*
ex5(31)	8×4	32	**6×3**	**18**	10×4	40	10×3	30	11×5	55	11×5	55	9×7	63	7×6	42*
ex5(32)	10×4	40	**6×4**	**24**	13×3	39	13×3	39	13×5	65	13×5	65	13×5	65	13×5	65*
ex5(33)	**7×3**	21	–	–	7×3	21	7×3	21*	11×5	55	11×5	55	8×6	48	8×6	48*
ex5(36)	**8×2**	**16**	**8×2**	**16**	10×2	20	10×2	20	12×4	48	12×4	48	13×4	52	13×4	52
ex5(38)	9×4	36	**6×4**	**24**	13×3	39	13×3	39	13×5	65	13×5	65	13×5	65	13×5	65
ex5(39)	8×2	16	–	–	11×3	33	11×3	33	10×4	40	10×4	40*	11×5	55	11×5	55
ex5(40)	12×6	72	–	–	15×5	75	13×4	52	17×7	119	15×6	90*	13×9	117	13×9	117*
ex5(43)	14×8	112	–	–	17×6	102	13×4	52*	17×8	136	13×6	78*	16×14	224	16×14	224*
exam(5)	6×11	66	–	–	7×6	42	**6×5**	**30***	8×7	56	7×7	49*	7×9	63	8×6	48*
exam(9)	30×59	1770	–	–	38×30	1140	**33×30**	**990***	42×29	1218	42×29	1218*	39×47	1833	39×47	1833*
max128(5)	17×14	238	–	–	19×9	171	14×5	70	20×12	240	13×7	91*	18×9	162	**11×6**	**66***
max128(8)	10×5	50	–	–	11×4	44	**10×4**	**40***	9×6	54	9×6	54*	9×5	45	9×5	45*
max128(17)	25×26	650	–	–	26×15	390	26×15	390*	25×16	400	25×16	400	18×11	198	**11×7**	**77**
mp2d(6)	6×10	60	–	–	6×10	60	**3×7**	**21***	7×11	77	5×10	50*	7×11	77	5×10	50*
mp2d(9)	8×6	48	–	–	9×6	54	**9×4**	**36***	11×7	77	11×7	77*	10×7	70	10×6	60
mp2d(10)	3×6	18	**3×4**	**12**	4×5	20	4×5	20*	3×7	21	2×7	14*	5×7	35	5×7	35*
z4(0)	15×15	225	**4×5**	**20**	16×11	176	6×6	36*	18×6	108	10×6	60	16×16	256	7×7	49*
z4(1)	28×28	784	–	–	30×16	480	**10×7**	**70**	22×16	352	10×8	80	22×19	418	24×23	552*
Z5xp1(2)	11×12	132	–	–	13×7	91	**11×5**	**55***	13×9	117	13×7	91*	13×8	104	11×7	77*
Z5xp1(3)	18×18	324	–	–	19×11	209	**10×6**	**60***	19×12	228	13×7	91*	14×13	182	13×6	78*

Table 2: Proposed lattice synthesis times for standard benchmark circuits: a comparison of the proposed method with the results presented in [3] and in [4]. All cases where the SAT-based synthesis of a non-decomposed lattice (column 3) is stopped after 10 minutes are marked with $-$.

	Standard Synthesis		$p = x_0$		$p = x_0 \oplus x_1$		$EP - SOP$	
	[3]	[4]	[3]	[4]	[3]	[4]	[3]	[4]
	t(s)	t(s)	t(s)	t(s)	t(s)	t(s)	t(s)	t(s)
adr4(1)	**0**	–	1.14	1.14	**0**	**0**	2.87	2.59
alu2(2)	**0**	5420.91	0.95	2.02	**0**	53.012	1.36	1470.75
alu2(5)	**0**	–	2.87	2.87	**0**	**0**	4.63	4.61
alu2(6)	**0**	5420.91	2.98	4.702	**0**	1.734	4.89	6.642
alu3(0)	**0**	–	0.03	0.03	**0**	**0**	0.07	0.07
alu3(1)	**0**	1.749	0.18	0.923	**0**	0.759	0.58	5.7
b12(0)	**0**	2.485	**0**	0.872	**0**	**0**	0.01	0.197
b12(1)	**0**	94.477	0.01	3.292	**0**	1.841	0.02	1271
b12(2)	**0**	309.477	0.11	4.672	**0**	4.505	0.63	5881
bcc(5)	1.09	–	0.39	0.4	**0**	**0**	3.51	3.51
bcc(7)	1.09	–	0.58	0.54	**0.03**	**0.03**	3.63	3.63
bcc(8)	1.09	–	0.66	0.65	**0.03**	**0.03**	1.09	3.7
bcc(12)	1.09	–	0.98	0.97	**0**	**0**	4.13	4.12
bcc(27)	1.09	–	2.35	2.32	**0.03**	**0.03**	12.55	12.36
bcc(43)	1.09	–	11.51	11.62	**0**	**0**	58.65	58.43
bcd.div3(1)	**0**	0.331	0.02	0.03	**0**	**0**	0.05	0.05
bcd.div3(2)	**0**	0.315	0.05	0.06	**0**	**0**	0.09	0.09
bcd.div3(3)	**0**	0.599	0.08	0.09	**0**	**0**	0.11	0.1
bench1(2)	**0.04**	–	13.45	13	0.332	0.03	11.73	11.71
bench1(3)	**0.04**	–	15.73	19.8	**0**	3.72	14.16	83.2
bench1(5)	0.04	–	23.76	23.88	**0**	**0**	20.56	20.58
bench1(6)	0.04	–	27.1	27.25	**0**	**0**	23.74	23.75
bench1(7)	0.04	–	30.21	32.547	**0**	**0**	27.08	27.07
bench1(8)	0.04	–	33.92	34.08	**0**	**0**	30.2	32.097
bench(4)	**0**	1.546	0.41	0.41	**0**	0.115	0.76	1.094
bench(5)	**0**	**0**	0.44	0.44	**0**	**0**	0.83	0.83
bench(6)	**0**	2.668	0.51	0.51	**0**	**0**	1.1	1.11
br2(4)	**0**	–	0.07	0.08	**0**	**0**	0.19	0.19
br2(5)	**0**	–	0.09	0.1	**0**	**0**	0.26	0.25
br2(6)	**0**	–	0.11	0.13	**0**	**0**	0.31	0.3
clpl(2)	**0**	**0**	0.24	0.23	**0**	**0**	0.44	0.42
clpl(3)	**0**	2953	0.96	10.078	**0**	9.127	1.19	308.15
clpl(4)	**0**	52.9	1.46	2.324	**0**	0.86	1.71	10.258
co14(0)	**0**		0.98	0.97	**0**	**0**	0.6	0.6
dc1(0)	**0**	0.413	0.01	0.207	**0**	**0**	0.04	0.04
dc1(1)	**0**	**0**	0.02	0.02	**0**	**0**	0.06	0.06
dc1(4)	**0**	0.585	0.04	0.04	**0**	**0**	0.14	0.14
dc1(6)	**0**	0.198	0.06	0.06	**0**	**0**	0.35	0.17
dc2(4)	**0**	1452.36	0.44	15.284	**0**	117.233	1.42	16.69
dc2(5)	**0**	35.305	0.52	2.424	**0**	5.72	1.54	45.741
dk17(0)	**0**	58.086	0.07	0.07	**0**	**0**	0.05	0.05
dk17(1)	**0**	56.658	0.14	0.14	**0**	**0**	0.1	0.1
dk17(3)	**0**	890.945	0.48	74.169	**0**	0.92	0.58	0.58
dk17(4)	**0**	788.761	0.66	0.67	**0**	**0**	0.68	1166
dk27(6)	**0**	**0**	0.11	0.13	**0**	**0**	0.08	0.08
ex4(4)	0.07	–	2.98	3	**0**	**0**	5.24	5.24
ex4(5)	0.07	–	8.34	8.36	**0**	**0**	10.69	10.68
ex5(31)	0.02	14.408	0.41	1.967	**0**	1.544	0.41	43.511
ex5(32)	0.02	2192.91	0.94	0.91	**0**	**0**	0.92	0.91
ex5(33)	0.02	–	1.23	1.21	**0**	**0**	1.24	1.23
ex5(36)	0.02	0.02	2.38	2.39	**0**	**0**	2.73	2.72
ex5(38)	0.02	1027.5	3.38	3.39	**0**	**0**	3.84	3.83
ex5(39)	0.02	–	3.83	3.84	**0**	**0**	4.32	4.31
ex5(40)	0.02	–	4.6	7.348	**0**	252.83	5.11	5.1
ex5(43)	0.02	–	6.18	96.164	**0**	819.055	6.96	6.94
exam(5)	0.03	–	4.89	40.155	**0**	67.401	4.94	16.151
exam(9)	0.03	–	11.19	7113.73	**0**	**0**	10.84	11.642
max128(5)	**0**	–	2.24	5266.51	**0**	10602.8	2.27	400.61
max128(8)	**0**	–	2.84	7.372	**0**	**0**	3.19	3.22
max128(17)	**0**	–	8	9.503	**0**	**0**	8.63	876.63
mp2d(6)	**0**	–	2.18	3164.15	**0**	433.23	2.62	424.213
mp2d(9)	**0**	–	2.37	2.847	**0**	**0**	2.99	3.05
mp2d(10)	**0**	1.704	2.37	2.4	**0**	0.327	3.01	3.279
z4(0)	**0**	1888.26	0.89	7.31	**0**	5.358	0.97	274.141
z4(1)	**0**	–	2.51	7329.12	**0**	6816.19	2.22	10.065
Z5xp1(2)	**0**	–	0.55	46.701	**0**	200.352	0.72	8.902
Z5xp1(3)	**0**	–	0.58	2279.54	**0**	9393.85	1.79	195.769

In Table 1 we report dimensions and areas of lattices, in Table 2 we report simulation times. Each row of the tables lists the results for any separate output function of the benchmark circuit.

More precisely, in Table 1, the first column reports the name and the number of the considered output of each instance. The following columns report dimension ($X \times Y$) and area ($Area = X \cdot Y$) of lattices for each method. In particular, the first group refers to the synthesis of the lattices, as described in [3] (columns 2 and 3) and in [4] (columns 4 and 5) without any decomposition; the second group refers to the synthesis of the lattices, as described in [3] (columns 6 and 7) and in [4] (columns 8 and 9), based on the P-circuit scheme decomposition with $p = 0$; the third group refers to the synthesis of the lattices, as described in [3] (columns 10 and 11) and in [4] (columns 12 and 13), based on the P-circuit scheme decomposition with $p = x_j$; finally, the last group refers to the synthesis of the lattices, as described in [3] (columns 14 and 15) and in [4] (columns 16 and 17) based on the EP-SOP decomposition. For each function, we bolded the best area. We marked with $-$ all cases where the synthesis of a non-decomposed lattice is stopped.

Moreover, in some cases the method proposed in [4] 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. In the synthesis of sublattices, whenever [4] is stopped, we use the respective sublattice synthesized with [3], 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 [4] did not find a solution within the fixed time limit for at least one sublattice, and had to be replaced with [3]. We also note that some benchmarks (e.g., *exam(5)* or *mp2d(9)*) show an overall area reduction with respect to the decomposed lattice synthesized only with [3], even if only one or two sublattices are synthesized with [4].

The first column of Table 2 reports the name and the number of the considered output of each instance. The following columns report the execution times for each considered method. The time values include the time for the decomposition plus the time for the synthesis of lattices used to compose the final lattice. For each function, we bolded the best execution time. Note that the execution times for the method in [3] are often equal to zero. To synthesize the lattices we used ESPRESSO that has a time granularity of 0.01s; smaller synthesis-times are indicated as zero.

In Table 3 we summarize the improvements of synthesis with decomposition

Table 3: Results of the decomposition. The values indicate the improvement with respect to the standard synthesis methods. When [4] does not complete, it is compared with [3].

	Smaller area		Average area gain		Less time		Average time gain	
	[3]	[4]	[3]	[4]	[3]	[4]	[3]	[4]
$p = x_0$	34%	15%	25%	30%	6%	25%	-2900%	53%
$p = x_0 \oplus x_1$	27%	13%	22%	27%	53%	50%	98%	57%
$EP - SOP$	27%	35%	28%	33%	2%	3%	-4900%	90%

vs. synthesis without decomposition. Every line of the table shows a different type of decomposition. The first column shows the considered decomposition scheme. The second and third columns report the percentages of lattices with smaller area with respect to the results reported in [3] and in [4], respectively. Columns four and five report the average area gain of the lattices that have a smaller area after the decomposition. Columns six and seven show the percentages of lattices taking less time for synthesis (not necessary with a more compact area); finally the last two columns report the average gain of execution time. The negative values of time gain in column eight are due to the poor granularity of ESPRESSO time. The synthesis in [3] is performed using ESPRESSO, and in many cases it takes less than 0.01 s. There are negative values because the time for decomposition is usually less than few seconds, and therefore it has a negligible impact on total synthesis time with respect to area gain.

By comparing the two projection function $p = 0$ and $p = x_j$ for the P-circuit decomposition model, we observe that we obtain better area results considering the $p = 0$ choice, at the expense of a limited increase in the run time needed to decompose the input functions.

By comparing our results with the ones obtained in [4], we obtain a higher percentage of benchmarks with a smaller area by applying the EP-SOP circuit decomposition scheme (35% of smaller area, in contrast to 15% and 13% obtained with the P-circuit scheme), with an average gain of about 33%. The difference between P-circuit-based and EP-SOP-based synthesis results is only partially explained by the fact that two decomposition methods differ in the way they handle crossing cubes. Most likely, this difference is a consequence of the fact that the two decompositions are performed with respect to different input variables. In particular, EP-SOP decomposition is performed with respect to the pair of variables appearing together most frequently among the products of a minimal SOP for the target benchmark, while for P-circuits we used the first two variables x_0 and x_1. Thus, these results clearly indicate that the simple frequency-based heuristic should be applied in the P-circuit scheme as well.

6. Conclusions

In this paper we proposed a new method for the synthesis of minimal-sized lattices, preprocessing the logic functions with a decomposition based on P-circuits and EP-SOP forms. The results demonstrate that lattice synthesis benefits from these Boolean decompositions, yielding smaller circuits with an affordable computation time (even less in some cases). Future work includes assessing the impact of more complex types of decompositions, with more expressive projection functions.

7. Acknowledgments

This project was partially funded by 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. 21 (8) (1972) 848–857.

[2] M. Altun, 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.

[3] M. Altun, M. D. Riedel, Logic Synthesis for Switching Lattices, IEEE Trans. Computers 61 (11) (2012) 1588–1600.

[4] G. Gange, H. Søndergaard, P. J. Stuckey, Synthesizing Optimal Switching Lattices, ACM Trans. Design Autom. Electr. Syst. 20 (1) (2014) 6:1–6:14.

[5] Y. Chen, G.-Y. Jung, D. A. A. Ohlberg, X. Li, D. R. Stewart, J. O. Jeppesen, K. A. Nielsen, J. F. Stoddart, R. S. Williams, Nanoscale molecular-switch crossbar circuits, Nanotechnology 14 (4) (2003) 462–468.

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

[7] A. Bernasconi, V. Ciriani, G. Trucco, T. Villa, Logic Synthesis by Signal-Driven Decomposition, in: K. Gulati (Ed.), Advanced Techniques in Logic Synthesis, Optimizations and Applications, Springer New York, 2011, pp. 9–29.

[8] A. Bernasconi, V. Ciriani, V. Liberali, G. Trucco, T. Villa, Synthesis of P-Circuits for Logic Restructuring, Integration 45 (3) (2012) 282–293.

[9] A. Bernasconi, V. Ciriani, G. Trucco, T. Villa, Using Flexibility in P-Circuits by Boolean Relations, IEEE Trans. Computers 64 (12) (2015) 3605–3618.

[10] A. Bernasconi, V. Ciriani, R. Cordone, EXOR Projected Sum of Products, in: IFIP/IEEE VLSI-SoC 2006 - International Conference on Very Large Scale Integration of System-on-Chip,, 2006.

[11] A. Bernasconi, V. Ciriani, R. Cordone, On Projecting Sums of Products, in: 11th Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools, 2008, pp. 787–794.

[12] A. Bernasconi, V. Ciriani, G. Trucco, T. Villa, Projected Don't Cares, in: Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools (DSD), 2012, pp. 57–64.

[13] A. Bernasconi, V. Ciriani, G. Trucco, 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.

[14] A. Bernasconi, V. Ciriani, L. Frontini, V. Liberali, G. Trucco, 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.

[15] J. C. Bioch, The Complexity of Modular Decomposition of Boolean Functions, Discrete Applied Mathematics 149 (1-3) (2005) 1–13.

[16] V. Kravets, Constructive Multi-Level Synthesis by Way of Functional Properties, Ph.D. thesis, Computer Science Engineering, University of Michigan (2001).

[17] S. Yang, Logic Synthesis and Optimization Benchmarks User Guide Version 3.0, User guide, Microelectronic Center (1991).

[18] T. Sasao, Switching Theory for Logic Synthesis, Kluwer Academic Publishers, 1999.

[19] R. Bryant, Graph Based Algorithm for Boolean Function Manipulation, IEEE Transactions on Computers 35 (9) (1986) 667–691.

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

[21] A. Bernasconi, V. Ciriani, L. Lago, On the error resilience of ordered binary decision diagrams, Theor. Comput. Sci. 595 (2015) 11–33.

[22] A. Bernasconi, V. Ciriani, Index-resilient zero-suppressed bdds: Definition and operations, ACM Trans. Design Autom. Electr. Syst. 21 (4) (2016) 72:1–72:27.

[23] D. Bañeres, J. Cortadella, M. Kishinevsky, A Recursive Paradigm to Solve Boolean Relations, IEEE Transactions on Computers 58 (4) (2009) 512–527.