Synthesis on Switching Lattices of Dimension-Reducible Boolean Functions

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

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

***Abstract*—In this paper we study the switching lattice synthesis of a special class of regular Boolean functions called *D-reducible* functions. D-reducible functions are functions whose points are completely contained in an affine space A strictly smaller than the whole Boolean cube $\{0,1\}^n$. The D-reducibility of a function f can be exploited in the lattice synthesis process: the idea is to independently find lattice implementations for the characteristic function of the subspace A and for the projection of f onto A, and to compose them in order to construct the lattice for f. The overall lattice area can be further reduced exploiting the peculiar structure of the affine subspaces of $\{0,1\}^n$. To this aim, we propose a method for implementing compact lattice representations of affine subspaces whose characteristic function is represented by the product of single literals and EXOR factors of two literals. The experimental results validate the proposed approach.**

I. Introduction

In this paper we study the lattice synthesis of *D-reducible* (or Dimension-reducible) Boolean functions [3], [4], [5], [6]. A D-reducible function $f : \{0,1\}^n \to \{0,1\}$ is a function that depends on all its n input variables, but can be studied and synthesized in a space of dimension strictly smaller than n. More precisely, the minterms of a D-reducible function f are completely contained in an affine space A strictly smaller than the whole Boolean cube $\{0,1\}^n$, so that f can be written as $f = \chi_A \cdot f_A$, where A is its unique associated affine space, χ_A is the characteristic function of A, and f_A is the projection of f onto A. Notice that f and f_A have the same number of points, but these are now compacted in a smaller space.

The D-reducibility of a function f can be exploited in the minimization process: the idea is to minimize the projection f_A of f onto the space A, instead of f. This approach thus requires two steps: *(i)* deriving the affine space A and the projection f_A; *(ii)* minimizing f_A in a given logic framework. As shown in [3], the first step can be performed in time polynomial in the initial representation of f. Previous studies on this subject [3], [4], [5] focused on the standard SOP (Sum of Products) minimization of f_A and proved how this approach to the synthesis of D-reducible functions often turns out to be convenient. They also showed that about 70% of the functions in the classical ESPRESSO benchmark suite have at least one output that is D-reducible: although D-reducible functions form a subset of all possible Boolean functions, a great amount of standard functions of practical interest falls in this class.

The aim of this paper is to analyze whether the D-reducibility structural property can be exploited in the switching lattice synthesis process as well. 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], but 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], [8].

Previous studies on this subject 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 synthesis on switching lattices, 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.

In this paper, we apply an approach based on decomposition to the class of D-reducible functions: the idea is to independently find lattice implementations for the projection f_A of a D-reducible function f, and for the characteristic function χ_A of the space A, and then to compose them in order to construct the lattice for f. To further reduce the overall lattice area, we exploit the peculiar structure of the function χ_A, that represents the minterms of an affine subspace of $\{0,1\}^n$. To this aim, we describe a method for implementing compact lattice representations of affine subspaces whose characteristic

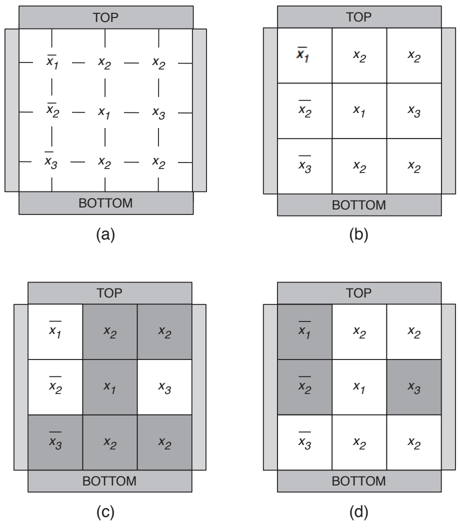

Fig. 1. A four terminal switching network implementing the function $f = \overline{x}_1\overline{x}_2\overline{x}_3+x_1x_2+x_2x_3$ (a); its corresponding lattice form (b); the lattice evaluated on the assignments 1,1,0 (c) and 0, 0, 1 (d), with grey and white squares representing ON and OFF switches,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

function is represented by the product of single literals and EXOR factors of two literals. Experimental results demonstrate that the synthesis of lattices based on D-reducible Boolean functions allows to obtain a more compact area. Moreover, the proposed approach allows to reduce the synthesis time, demonstrating that a polynomial preprocessing, which reduces the size of the problem, can be very useful for the overall computation time.

The paper is organized as follows. Preliminaries on switching lattices and D-reducible Boolean functions are described in Section II. Section III shows how a D-reducible function can be efficiently decomposed and implemented by a switching lattice. Section IV provides the experimental results and concludes the paper.

II. Preliminaries

In this section we briefly review some basic notions and results on switching lattices [1], [2], [8] and D-reducible functions [3], [4], [5], [6].

A. Switching Lattices

A switching lattice is a two-dimensional lattice 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 [8]:

- each four-terminal switch is controlled by a literal;
- each switch may be 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;

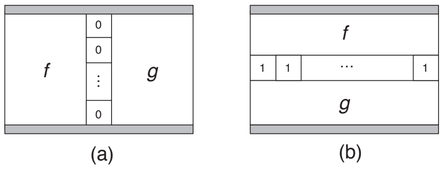

Fig. 2. Lattice implementation of $f \vee g$ (a) and of $f \wedge g$ (b).

- 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 produce any path from the top to the bottom edge (Figure 1 (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 [2] 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 of size $r \times s$, where r and s are the number of products in an irredundant SOP representations of f^D and f, respectively. 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 methods 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 [8], 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

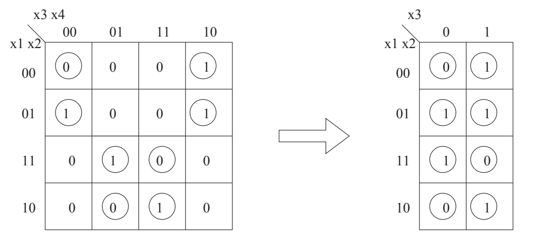

Fig. 3. Karnaugh maps of a D-reducible function f and its corresponding projection f_A.

lattice. It then searches for successively better implementations until either an optimal solution is found, or else 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 is allowed, moreover the lattice considers only the top-to-bottom paths and implements the function f, but not its dual.

Finally, we recall from [8] that given the switching lattices implementing two functions f and g, we can construct the lattices representing their disjunction and their conjunction using a padding column of 0s and a padding row of 1s, respectively, as shown in Figure 2. Indeed, the column of 0s in Figure 2 (a) 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, the padding row of 1s in the lattice in Figure 2 (b) allows to join any top-to-bottom accepting path for 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 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 functions can never reach the opposite edge of the lattice if the other function evaluates to 0.

B. D-reducible Boolean functions

D-reducible functions are functions whose points are completely contained in an affine space A strictly smaller than the whole Boolean cube $\{0,1\}^n$:

Definition 1: The Boolean function $f : \{0,1\}^n \to \{0,1\}$ is *D-reducible* if $f \subseteq A$, where $A \subset \{0,1\}^n$ is an affine space of dimension strictly smaller than n.

Recall that an affine space is a vector space, or the translation of a vector space. Formally, given a vector subspace V of $(\{0,1\}^n, \oplus)$, and a point α in $\{0,1\}^n$, then the set $A = \alpha \oplus V = \{\alpha \oplus v \mid v \in V\}$ is an *affine space* over V with *translation point* α [3].

Let f be a D-reducible function. The minimal affine space A containing f is unique and it is called the *associated affine space* of f. The function f can be represented in the following way: $f = \chi_A \cdot f_A$, where $f_A \subseteq \{0,1\}^{\dim A}$ is the projection of f onto A and χ_A is the characteristic function of A. Moreover, as shown in [7], an affine space can be represented by a simple expression, called *pseudoproduct*, consisting in an AND of EXORs or literals. In particular, an affine space of dimension $\dim A$ can be represented by a pseudoproduct containing $(n - \dim A)$ EXOR factors.

For instance, consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$ in the Karnaugh map on the left side of Figure 3. The function f is D-reducible, i.e., we can project it onto a space of dimension three (the space marked with circles in the Karnaugh map). We can therefore study the new function f_A that depends only on three variables, represented in the Karnaugh map on the right side of Figure 3. Notice that f and f_A have the same number of points, but these are now compacted in a smaller space. If we synthesize f and f_A in the classical SOP framework we obtain $f = \overline{x}_1 x_3 \overline{x}_4 + \overline{x}_1 x_2 \overline{x}_4 + x_1 \overline{x}_2 x_3 x_4 + x_1 x_2 \overline{x}_3 x_4$, and $f_A = \overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3$. (Note that f depends on all the variables $x_1, \dots x_4$.) The new and more compact form for f is then $f = (x_1 \oplus \overline{x}_4)(\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3)$. The EXOR $(x_1 \oplus \overline{x}_4)$ represents the new Boolean space where we study f_A.

The test that establishes whether a function f is D-reducible and the computation of the smallest affine space containing f can be performed in polynomial time, by finding the reduced row echelon form of a matrix derived from any SOP representation of f (see [5] for more details). Moreover, the projection f_A of f onto A can be simply derived from f by deleting $\dim A$ variables. It is important to note that f_A can be computed starting from any SOP representation of f without generating all its minterms.

III. Switching lattices for D-reducible functions

In this section we will discuss how to obtain a lattice for a D-reducible function implementing the characteristic function of the affine space A and the projection f_A with physically separated regions in a single lattice. Recall from Section II-B that a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where A is its unique associated affine space, χ_A is the characteristic function of A, and f_A is the projection of f onto A.

Given the two switching lattices implementing χ_A and f_A, we can easily construct the lattices representing their conjunction using a padding row of 1s, as shown in Figure 4. Indeed, the row of 1s allows to join any top-to-bottom accepting path for the characteristic function of A with any top-to-bottom accepting path for the projection f_A, so that the overall lattice evaluates to 1 if and only if both χ_A and f_A evaluate to 1. Of course, if the lattices for χ_A and f_A 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.

Since the two functions f_A and χ_A depend on fewer variables than the original function f, their synthesis should be more feasible and should produce lattice implementations of smaller area. 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, reducing the synthesis of a target Boolean function to the synthesis of smaller functions could represent a very beneficial approach.

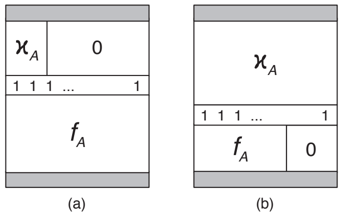

Fig. 4. Lattice implementations of a D-reducible function $f = \chi_A \cdot f_A$. (a) Composition scheme when the lattice for χ_A has fewer columns. (b) Composition scheme when the lattice for f_A has fewer columns.

To further reduce the overall lattice area, we could exploit the peculiar structure of the function χ_A, that represents the minterms of an affine subspace of $\{0,1\}^n$, building its lattice representation block by block. More precisely, we have two possible approaches for the synthesis of a minimal-sized lattice for χ_A: *(i)* directly apply one of the synthesis methods presented in [2] and in [8], or *(ii)* build and compose the lattices representing each EXOR factor, or group of EXOR factors, occurring in χ_A. For this second approach, we now describe a method for implementing a compact lattice representation of affine spaces that can be represented by a 2-*pseudoproduct*, i.e., a product of EXOR of at most two literals. Observe that the class of D-reducible functions whose affine subspace can be described with a 2-pseudoproduct is particularly interesting as EXOR factors are considered technologically feasible if they contain a bounded number of literals, typically 2 [9].

The problem of the minimization of the number of literals in the characteristic function χ_A of the affine space A has been addressed in [4]. The representation of an affine space is not unique, and different pseudoproducts can be characteristic functions of the same affine space. Unfortunately, finding a minimal pseudoproduct, in terms of number of literals, representing an affine space is an NP-hard problem [4], therefore, in the same paper, a greedy heuristic algorithm has been designed. Thus, to avoid the presence of EXOR factors with an unbounded number of literals in the function χ_A, we can first heuristically find an optimal representation of the affine space, and then remove from it all EXOR factors with more than two literals. In this way we obtain the algebraic representation of a new affine subspace A' that contains the original affine space A, and we can still decompose f as $f = \chi_{A'} \cdot f_{A'}$.

In the following analysis we will then restrict our attention to D-reducible functions decomposed w.r.t. an affine subspace, not necessarily the smallest, represented by the product of single literals and EXOR factors of two literals. The presence of at most two literals in each EXOR factors gives us a simple method for partitioning the variables in order to determine a compact lattice for χ_A. First of all, observe that the pseudoproduct describing A corresponds to a linear system, whose solutions are exactly the minterms in A. For instance, the pseudoproduct $(x_1 \oplus \overline{x}_3) \cdot (x_2 \oplus x_4) \cdot \overline{x}_5 \cdot (x_1 \oplus x_8)$, which describes an affine subspace of $\{0,1\}^8$, is represented by the system

$$\begin{cases} x_1 \oplus \overline{x}_3 &= 1 \\ x_2 \oplus x_4 &= 1 \\ \overline{x}_5 &= 1 \\ x_1 \oplus x_8 &= 1 \end{cases}$$

that can be rewritten as

$$\begin{cases} x_1 &= x_3 \\ x_2 &= \overline{x}_4 \\ x_5 &= 0 \\ x_1 &= \overline{x}_8 \end{cases}$$

From this system we immediately derive the following equalities

$$\begin{aligned} x_1 &= x_3 = \overline{x}_8 \\ x_2 &= \overline{x}_4 \\ x_5 &= 0\,, \end{aligned}$$

that suggest a natural partition of a subset of the input variables:

$$\{\{0, x_5\}, \{x_1, x_3, \overline{x}_8\}, \{x_2, \overline{x}_4\}\}\,,$$

where each subset of the partition contains a set of literals (or the constant 0) that get the same value on A. The input variables missing from the partition (x_6 and x_7 in the example) are the variables that can assume all the possible values on A. In particular, in our example x_5 must be always equal to 0, while x_1, x_3, and $\overline{x}_8$ must have the same value (0 or 1), as well as x_2 and $\overline{x}_4$.

As this example clearly suggests, it is always possible to describe an affine space A, described by an EXOR of at most two literals, through a partition P_A of the input variables, where two variables, possibly complemented, are in the same subset of the partition if and only if they are equal on A. This partition can now be exploited to build the lattice for χ_A.

Theorem 1: Let A be an affine subspace of $\{0,1\}^n$ described by the product of single literals and EXOR of two literals, let P_A be the partition of the subset of input variables that defines A, and let $n' \leq n$ be the number of distinct variables occurring in P_A. Suppose that P_A contains ℓ subsets of literals, in addition to the subset C with the constant 0. Finally, let c be the number of literals in C. Then A can be implemented with a lattice of area $r \times 2$, where the number r of rows is given by

$$r = \begin{cases} n' & \text{if } c \geq \ell - 1 \\ n' + \ell - 1 - c & \text{if } c < \ell - 1 \end{cases}$$

Proof: Let $S \in P_A$, be one of the ℓ subsets of P_A without the constant 0. The literals in S must be equal to each other on A, thus this subset can be described by the disjunction of two products: the product of all literals in S and the product of the complement of each literal. Thus we can easily build the lattice for S, using two columns representing the two products, that have the same length. Since the two switches on each row are controlled by a variable and its complement, the top-to-bottom accepting paths cannot intersect the two columns, therefore we do not need the padding column of 0 between the two terms of the disjunction. Now, let us consider the set C that contains the c literals that are constant and equal to 0 on A. This set can be implemented with a single column lattice, with a switch assigned to each literal of C. Since this one column lattice must be composed with the previous two column ones, we can extend it with a second column, identical to the first one. Observe that each of the n' literals in P_A occurs in exactly one subset of the partition, and therefore in exactly one row of the lattice.

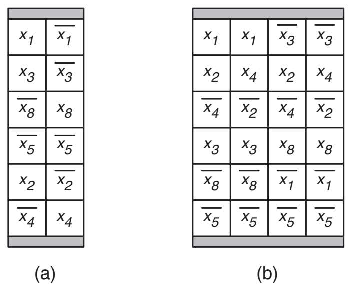
x_1 $\overline{x}_1$
x_3 $\overline{x}_3$
$\overline{x}_8$ x_8
$\overline{x}_5$ $\overline{x}_5$
x_2 $\overline{x}_2$
$\overline{x}_4$ x_4
(a)
x_1 x_1 $\overline{x}_3$ $\overline{x}_3$
x_2 x_4 x_2 x_4
$\overline{x}_4$ $\overline{x}_2$ $\overline{x}_4$ $\overline{x}_2$
x_3 x_3 x_8 x_8
$\overline{x}_8$ $\overline{x}_8$ $\overline{x}_1$ $\overline{x}_1$
$\overline{x}_5$ $\overline{x}_5$ $\overline{x}_5$ $\overline{x}_5$
(b)

Fig. 5. Lattice implementations of the function $\chi_A = (x_1 \oplus \overline{x}_3) \cdot (x_2 \oplus x_4) \cdot \overline{x}_5 \cdot (x_1 \oplus x_8)$: (a) lattice derived applying Theorem 1; (b) lattice synthesized with the algorithm described in [2].

To compose the sublattices and build the overall lattice representing A, we can exploit the particular structure of the sublattice for C to save padding rows of 1s. Indeed, thanks to the presence of the two identical columns, the two switches on each row of the sublattice for C are controlled by the same Boolean literal. Thus, each single row can be directly inserted between two sublattices representing EXOR factors, as the repeated literal allows to extend any accepting path that reaches the bottom of the first sublattice to the top of the other sublattice, whenever the literal gets the value 0 (i.e., its complement is true). In other words, the sublattice for C is split and each row is inserted between two sublattices for the other subsets of P_A, in order to save padding rows of 1s. Now observe that to join the ℓ sublattices representing the subsets of P_A other than C, we would need $\ell - 1$ padding rows of 1s, that can be all saved if C contains enough literals, i.e., if $c \geq \ell - 1$. In this case, the overall number of rows is given by the number n' of literals occurring in the partition P_A. Otherwise, if $c < \ell - 1$, we must insert $\ell - 1 - c$ padding rows of 1s. ■

Applying the construction described in this theorem to our running example $\chi_A = (x_1 \oplus \overline{x}_3) \cdot (x_2 \oplus x_4) \cdot \overline{x}_5 \cdot (x_1 \oplus x_8)$, we get the lattice of size 12 depicted in Figure 5 (a). Observe that we do not need the two padding rows of 1s after and before the row whose switches are controlled by the same Boolean literal $\overline{x}_5$, and that the number of rows is equal to the number of distinct variables occurring in the characteristic function χ_A. Figure 5 (b) shows the lattice of size 24 for χ_A, obtained using the synthesis algorithm described in [2]. The method based on SAT, described in [8], synthesizes a lattice of size 12, equivalent to the one obtained applying Theorem 1.

Now, consider the affine space $\chi_A = x_1\overline{x}_2(x_3 \oplus x_4) \cdot (x_5 \oplus x_6) \cdot (x_7 \oplus x_8) \cdot (x_9 \oplus x_{10})$, corresponding to the partition $P_A = \{\{0, \overline{x}_1, x_2\}, \{x_3, \overline{x}_4\}, \{x_5, \overline{x}_6\}, \{x_7, \overline{x}_8\}, \{x_9, \overline{x}_{10}\}\}$. In this example, $c < \ell - 1$, as $c = 2$ and $\ell = 4$. Thus, the lattice for χ_A, built applying Thereom 1, contains a padding row of 1s, in addition to the $n' = 10$ rows associated to the literals occurring in P_A, as shown in Figure 6 for a lattice with left-to-right connectivity.

With the construction described in Theorem 1, it is possible to derive lattices more compact than those synthesized with the method presented in [2], with a gain in area that increases with the number ℓ of subsets in the partition associated to the affine space A. Consider for instance an affine space described

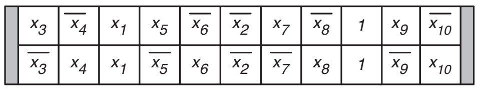
x_3 $\overline{x}_4$ x_1 x_5 $\overline{x}_6$ $\overline{x}_2$ x_7 $\overline{x}_8$ 1 x_9 $\overline{x}_{10}$
$\overline{x}_3$ x_4 x_1 $\overline{x}_5$ x_6 $\overline{x}_2$ $\overline{x}_7$ x_8 1 $\overline{x}_9$ x_{10}

Fig. 6. Left-to-right lattice implementation of the function $\chi_A = x_1\overline{x}_2(x_3 \oplus x_4) \cdot (x_5 \oplus x_6) \cdot (x_7 \oplus x_8) \cdot (x_9 \oplus x_{10})$.

by exactly ℓ EXOR of two literals, with no literal in common. Applying our method, we can synthesize a lattice of area $(2\,\ell + \ell - 1) \times 2 = 6\,\ell - 2$, while the algorithm proposed in [2] would synthesize a lattice of area $2^\ell \times 2\,\ell = 2^{\ell+1}\ell$, since the minimal SOP forms for this affine space and for its dual contain 2^ℓ and $2\,\ell$ products, respectively.

In general, the affine subspaces containing EXOR of more than two literals in their characteristic function, have a more complex structure, which cannot be simply described with a partition of the input variables. In this case, we can build a lattice implementation composing the lattices derived for each EXOR factor, or group of EXOR factors, in the characteristic function χ_A of the given affine space A. More precisely, we can implement a lattice representing the product of the single literals and of the EXOR factors of two literals occurring in χ_A applying Theorem 1, and compose it with the lattice implementations of the other EXOR factors. Moreover, we can use the recursive method developed in [2] for the specific case of the parity function, to implement an EXOR of m literals with a lattice of area $m \times 2^{m-1}$, instead of the general method that would synthesize a lattice of dimension $2^{m-1} \times 2^{m-1}$. Finally, we can use the rows controlled by single literals to join the sublattices of the different EXOR factors, in place of the padding rows of 1s.

IV. Experimental results and conclusion

In this section we report the experimental results for the synthesis of lattices of D-reducible functions, as described in Section III. The aim of our experimentation is to determine if and how much the lattice implementation based on the projection onto affine subspaces is more compact than the implementation of plain lattices [8]. As already mentioned, EXOR factors are considered technologically feasible if they contain a bounded number of literals, typically 2 [9]. For this reason, in our experiments we have considered only D-reducible functions decomposed with respect to affine subspaces, not necessarily the smallest, represented by the product of single literals and EXOR factors of two literals. In the following, we will refer to these functions as *2D-reducible functions*.

The algorithms have been implemented in C. The experiments have been run on a machine with two AMD Opteron 4274HE for a total of 16 CPU at 2.5 GHz and 128 GByte of main memory, running Linux CentOS 6.6. The benchmarks are taken from LGSynth93 [10]. We considered each output as a separate Boolean function, for a total of 385 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. To synthesize the lattices of the benchmarks and of their projection onto the affine subspaces, we used a collection of Python scripts for computing minimum-

TABLE I. LATTICE SIZES FOR 2D-REDUCIBLE BENCHMARK CIRCUITS: A COMPARISON OF PLAIN AND DECOMPOSED LATTICES.

	[8]				2D-Red									
Benchmark	X	Y	$Area$	$Time$	X_{tot}	Y_{tot}	$Area_{tot}$	$Time$	X_χ	Y_χ	$Area_\chi$	X_Z	Y_Z	$Area_Z$
addm4(0)	9	12	108	0.32	3	8	**24**	27.4	1	3	3	3	5	15
addm4(1)	22	23	506	87.14	22	23	506	24.42	1	1	1	22	22	484
addm4(2)	33	36	1188	92.85	33	36	1188	1089	1	1	1	33	35	1155
amd(3)	4	5	**20**	446.04	3	7	21	36.79	1	1	1	3	6	18
amd(4)	10	14	140	723.56	10	14	140	55.98	1	1	1	10	13	130
amd(5)	6	2	**12**	37.72	2	8	16	1	2	6	12	2	2	4
amd(6)	6	3	**18**	1686.65	3	8	24	0.34	1	5	5	3	3	9
amd(7)	5	5	20	88.57	3	6	**18**	4.13	1	1	1	3	5	15
exam(4)	9	25	225	78.26	9	20	**180**	1286.98	1	2	2	9	18	162
exp(6)	5	4	**20**	1118.05	3	7	21	16.1	1	2	2	3	5	15
exp(10)	6	12	72	3361.18	6	5	**30**	294	1	2	2	5	4	20
exp(11)	6	12	72	2058.63	5	8	**40**	248	2	4	8	5	3	15
gary(2)	12	14	**168**	0.03	12	15	180	152.15	2	2	4	12	12	144
gary(3)	5	12	60	253.35	5	12	60	76.3	1	2	2	5	10	50
in2(6)	39	36	1404	0.1	39	35	**1365**	0.1	1	2	2	39	33	1287
in2(7)	17	26	442	0.03	17	26	442	0.06	1	1	1	17	25	425
in2(8)	27	31	837	0.06	27	31	837	0.08	1	1	1	27	30	810
in2(9)	40	36	1440	0.11	40	36	1440	0.11	1	1	1	40	35	1400
in5(6)	5	4	**20**	1129	3	7	21	1.03	1	3	3	4	4	16
m2(5)	8	9	72	1222.7	4	6	**24**	975	1	1	1	4	5	20
t1(0)	6	9	54	1409.76	3	8	**24**	1416.78	1	1	1	3	7	21
t1(1)	7	9	63	236.07	7	9	63	831.62	1	1	1	7	8	56
t1(3)	3	4	**12**	0.97	3	5	15	0.33	1	1	1	3	4	12
t1(4)	3	3	**9**	0.31	3	4	12	0.04	1	1	1	3	3	9
t1(5)	5	3	15	12.77	3	5	15	10.24	1	1	1	3	4	12
			6997	14044.23			6706	6458						

area switching lattices, using transformation to a series of SAT problems [8]. The lattice implementation of the characteristic functions of the affine subspaces have been derived applying Theorem 1, as they are described by products of EXOR factors of at most two literals.

In Table I for each benchmark we compare the area of the lattice for the plain benchmark with the area of the lattice built applying the decomposition based on the D-reducibility property (see Figure 4). In more detail, the first column reports the name of the benchmarks. The following four columns report the dimensions (X,Y), the area ($Area$) and the synthesis time ($Time$, in seconds) of plain lattices. The other columns report the dimensions of the lattices obtained applying the decomposition scheme. In particular we report the dimensions of the overall lattice and the synthesis time (X_{tot}, Y_{tot}, $Area_{tot}$, $Time$), the dimension of the lattice implementation of the affine spaces (X_χ, Y_χ, $Area_\chi$), and the dimension of the projection of the benchmark on the affine space (X_Z, Y_Z, $Area_Z$). In column eight we have bolded the values where we obtain a more compact area. In the last row we report the sum of the corresponding columns.

Results demonstrate that the lattice synthesis of 2D-reducible Boolean functions allows to obtain a more compact area in 15% of the considered cases, with an average gain of about 24%. In particular, considering only the subset of functions whose affine subspace description contains at least one EXOR of two literals (i.e., not only single literals), we obtain a more compact area in about 11% of the cases, with an average gain of about 40%. Moreover, our results show that for 2D-reducible functions we can reduce the synthesis time of the lattices of about 50%, with respect to the time needed for the synthesis of plain lattices. This is due to the fact that the proposed approach is based on a polynomial-time computation of the lattice implementation of the affine spaces, and this is useful to reduce the dimension of the considered problem, allowing to reduce the overall time needed to compute the solution.

In conclusion, in this paper we propose a novel approach for the synthesis of switching lattices of D-reducible Boolean functions. Results demonstrate that we can obtain a more compact area, computed with a reduced synthesis time. In future work we will extend the experimental results, considering the whole class of D-reducible Boolean functions, defined with respect to affine spaces with EXOR factors of fan-in greater than two. Another interesting research direction would be assessing the impact of other types of decompositions.

V. ACKNOWLEDGMENTS

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

REFERENCES

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

[2] M. Altun and M. D. Riedel, "Logic synthesis for switching lattices," *IEEE Trans. Computers*, vol. 61, no. 11, pp. 1588–1600, 2012.

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

[4] ——, "Logic synthesis and testability of d-reducible functions," in *VLSI-SoC*, 2010, pp. 280–285.

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

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

[7] V. Ciriani, "Synthesis of SPP Three-Level Logic Networks using Affine Spaces," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 22, no. 10, pp. 1310–1323, 2003.

[8] 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.

[9] N. Weste and K. Eshraghian, *Principles of CMOS VLSI Design*. Addison-Wesley Publishing Company, 1993.

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