Dimension-reducible Boolean Functions based on Affine Spaces [1]

Anna Bernasconi[2]
and
Valentina Ciriani[3]

We define and study a new class of regular Boolean functions called D-reducible. A D-reducible function, depending on all its n input variables, can be studied and synthesized in a space of dimension strictly smaller than n. We show that the D-reducibility property can be efficiently tested, in time polynomial in the representation of f, i.e., an initial SOP form of f. A D-reducible function can be efficiently decomposed, giving rise to a new logic form, that we have called DredSOP. This form is shown here to be generally smaller than the corresponding minimum SOP form. Our experiments have also shown that a great number of functions of practical importance are indeed D-reducible, thus validating the overall interest of our approach.

Categories and Subject Descriptors: B.6.3 [**Logic design**]: Design Aids

General Terms: Automatic synthesis, Optimization

Additional Key Words and Phrases: logic synthesis, regular Boolean functions

1. INTRODUCTION

A function encoding a real life problem often exhibits a regular structure that should be exploited for logic synthesis [Sasao 1993; Bernasconi et al. 2003]. In general, it is not always clear whether a Boolean function is "regular", and which type of regularity could be exploited for its synthesis [Aloul et al. 2002; Kravets and Sakallah 2000; 2001; Bernasconi et al. 2002a; 2002b]. Some heuristic attempts have been developed in this direction [Bernasconi et al. 2008], but the general question still remains open. Here we study this problem for functions exhibiting a new type of regularity (*Dimension-reducibility based on affine spaces*) that, as we will see, is sufficiently common to make the case interesting, and easy to be tested.

Informally, *Dimension-reducible functions based on affine spaces* (shortly, *D-reducible functions*) are functions whose minterms are contained in a space A strictly smaller than the whole Boolean space $\{0,1\}^n$. 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 A, instead of f. This approach thus requires two steps: *(i)* deriving the space A and the projection f_A; *(ii)* minimizing f_A in a given logic framework.

In this paper we focus on the standard SOP (Sum of Products) minimization, and we prove how our approach to the synthesis of D-reducible functions often turns out to be convenient. Moreover, the algorithm deriving the smallest space containing

[1]A preliminary version of this paper has been presented in *DSD Euromicro* [Bernasconi and Ciriani 2006].
[2]Department of Computer Science, Università di Pisa, 56100 Pisa Italy, annab@di.unipi.it
[3]Department of Information Technologies, Università degli Studi di Milano, 26013 Crema (CR) Italy, ciriani@dti.unimi.it

f has time complexity polynomial in the representation of f, i.e., the initial SOP form of f. Indeed, even if the property of dimension-reducibility depends on the minterms of a Boolean function, seen as vectors in the Boolean space $\{0,1\}^n$, we are able to determine whether a function is D-reducible working on its products *without generating all the minterms.* This result is very important from a computational point of view, as the number of products in a SOP of a function can be exponentially smaller than the number of its minterms.

As this study will need non trivial formal tools, we start here by giving an intuitive presentation of D-reducibility. Consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$ in the Karnaugh map on the left side of Figure 1. 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 the figure. Notice that f and f_A have the same number of minterms, 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 $f_A = \overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3$, respectively. (Note that f depends on all the variables $x_1, \ldots, x_4$.) 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 Boolean space A where we study f_A. Figure 2 shows the resulting network for the function f.

It is important to notice that, in general, D-reducible functions depend on all their n input variables, even if we are able to study them in a space of dimension strictly smaller than n. In other words, D-reducible functions are, in general, not degenerate.

The key idea of this paper is that if we project a function onto a smaller Boolean space we have the chance of reducing the Hamming distance between its minterms in order to merge them in bigger cubes in the final SOP form. For example, consider the minterm 1101 in the Karnaugh map on the left side of Figure 1, its corresponding product $x_1 x_2 \overline{x}_3 x_4$ is prime since no other minterm can be merged with 1101. If we project the function onto the new space $(x_1 \oplus \overline{x}_4)$, its corresponding minterm 110 can be merged with 010 giving rise to the prime product $x_2 \overline{x}_3$. Observe that simple projections with single literals as $x_i \cdot f$ do not change the Hamming distance between minterms, while projections with EXORs do.

In this paper we describe a simple test that establishes whether a function is D-reducible and computes the smallest space that contains it. We then propose a new three level logic form (*DRedSOP*) for f, which is an AND of some EXOR factors (or literals) representing the projection space A, and the SOP expression for f_A. Figure 2 shows a DRedSOP network. The concept of D-reducibility is then developed for functions with don't care conditions, and the minimization technique is duly extended in this context. The advantage of DRedSOP circuits is that their depth is bounded, since the number of logic levels is equal to three. Moreover, circuits with a bounded number of levels, three or four ([Perkowski 1995; Sasao 1995; Dubrova and Ellervee 1999; Debnath and Sasao 1999; Luccio and Pagli 1999; Debnath and Vranesic 2003; Ishikawa et al. 2004]), often result to be much more compact in area than classical two level logic forms ([McGeer et al. 1993; Coudert

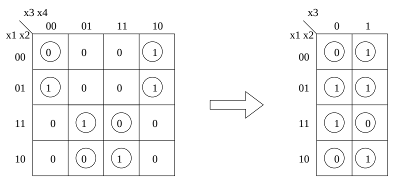

Fig. 1. *Karnaugh maps of a D-reducible function f and its corresponding projection f_A.*

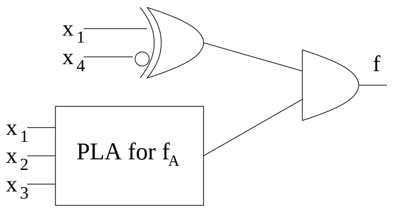

Fig. 2. *Network for the D-reducible function f of Figure 1, represented by* $(x_1 \oplus \overline{x}_4)(\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3)$.

1994; Fišer and Toman 2009]).

We can further observe that any Boolean function can be represented as $A \cdot f$ where A is a Boolean subspace of $\{0,1\}^n$. If f is D-reducible the space A is strictly contained in $\{0,1\}^n$, otherwise f is banally equivalent to the function $1 \cdot f$ where 1 represents the entire Boolean space $\{0,1\}^n$, i.e., $A = \{0,1\}^n$. We can view this synthesis method as a special Boolean factorization where instead of literal terms we have EXORs. Factorization of literal terms is a widely studied field in multi-level logic [Caruso 1991; Sasao 1999].

We finally study the relationship between D-reducible functions and another class of regular Boolean functions, the *autosymmetric functions*, introduced in [Luccio and Pagli 1999] and studied in [Bernasconi et al. 2003]. Similarly to D-reducible functions, autosymmetric functions exhibit a regular structure that can be described using the EXOR operation. We first observe that D-reducibility and autosymmetry are different regularities, since autosymmetric functions can be studied in a new space whose variables are EXOR combinations of the original ones, while D-reducible functions are studied in a projection space producing an expression where the EXOR gates are in AND with a SOP form. We then provide examples of autosymmetric functions that are not D-reducible, and D-reducible functions that are not autosymmetric. However, we also show that D-reducible functions have an interesting connection with autosymmetric functions through their Walsh transform.

Our experimental results show 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 paper is organized as follows. In the next section we review some basic definition and properties of affine spaces. In Section 3 we formally define D-reducible functions. In Section 4 we propose a synthesis algorithm for DRedSOP forms. In

Section 5 we analyze the relationship among D-reducible, degenerate and autosymmetric functions [Bernasconi et al. 2003]. Finally, in Section 6 we describe our experimental results.

2. ALGEBRAIC PRELIMINARIES

In this section we briefly review some basic notions on affine spaces that are useful in the sequel (for a more detailed introduction on affine spaces see [Cohn 1981; Ciriani 2003]).

We work in a Boolean space $\{0,1\}^n$ described by n variables $x_1, x_2, \dots, x_n$, where each point is represented by a binary vector of n *components*. Hereafter, we shall use the terms vector and point with the same meaning.

In the space $\{0,1\}^n$, an *EXOR factor* is an EXOR (or modulo 2 sum), denoted by $\oplus$, of variables, one of which possibly complemented (an EXOR with just one literal corresponds to the literal itself). Let us now extend the symbol $\oplus$ to denote the elementwise EXOR between two vectors. Then, $\alpha \oplus \beta$ is the vector obtained from β complementing in it the elements corresponding to the 1's of α. For example $1011 \oplus 0111 = 1100$.

We recall that, a vector subspace V of the vector space $(\{0,1\}^n, \oplus)$ is a subset of $\{0,1\}^n$ containing the zero vector $\mathbf{0} = 00\dots0$, such that for each v_1 and v_2 in V we have that $v_1 \oplus v_2$ is in V. Note that a vector subspace of a vector space is a vector space itself.

Example 1. *The set* $V = \{000, 001, 010, 011\}$ *is a vector subspace of* $(\{0,1\}^3, \oplus)$. *In fact,* $\mathbf{0} = 000$ *is in* V, *and* $001 \oplus 010 = 011 \in V$, $001 \oplus 011 = 010 \in V$, $010 \oplus 011 = 001 \in V$, $001 \oplus 000 = 001 \in V$, *etc.*

Each vector subspace V of $(\{0,1\}^n, \oplus)$ contains 2^k vectors, where k is a positive integer. We say that V has dimension k or is k-dimensional (shortly $\dim V = k$). The subspace of Example 1 has 2^2 points, and its dimension is 2.

A k-dimensional vector space V is generated by a basis B containing k vectors. Each vector v in a basis B is linearly independent of all the other vectors in B, i.e., v is not generated by any EXOR combination of the other vectors in B. A vector space, in general, has not a unique basis. In fact, a set of k linearly independent vectors in a vector space V of dimension k always forms a basis of V. For example the vector space $V = \{000,\ 001,\ 010,\ 011\}$ has three different bases, namely $\{010, 011\}$, $\{001, 010\}$, and $\{001, 011\}$.

Given a vector subspace V of $(\{0,1\}^n, \oplus)$, and a point α in $\{0,1\}^n$, we build an *affine space* performing the EXOR between α and each point of V. Formally we have:

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

Example 2. *Consider the vector space* $V = \{000, 010, 011, 001\}$ *and the vector* $\alpha = 100 \in \{0,1\}^3$. *The set* $A = \alpha \oplus V = 100 \oplus V = \{100, 110, 111, 101\}$ *is an affine space over* V. *Note that if we choose* α *as any vector of* A, *we obtain the same result. In this example* $A = 100 \oplus V = 110 \oplus V = 111 \oplus V = 101 \oplus V$.

$$
\begin{array}{cl}
\text{x1 x2 x3 x4 x5} & \\
\boxed{0\ 0\ 0\ 0\ 1} & = \mathrm{a} \\
0\ 0\ 1\ 0\ 0 & \\
0\ 1\ 0\ 1\ 1 & \\
0\ 1\ 1\ 1\ 0 & \\
1\ 0\ 0\ 1\ 1 & \\
1\ 0\ 1\ 1\ 0 & \\
1\ 1\ 0\ 0\ 1 & \\
1\ 1\ 1\ 0\ 0 & \\
\mathrm{A} &
\end{array}
\qquad
\begin{array}{cl}
\text{x1 x2 x3 x4 x5} & \\
0\ 0\ 0\ 0\ 0 & \\
\boxed{0\ 0\ 1\ 0\ 1} & = \mathrm{v}_1 \\
\boxed{0\ 1\ 0\ 1\ 0} & = \mathrm{v}_2 \\
0\ 1\ 1\ 1\ 1 & \\
\boxed{1\ 0\ 0\ 1\ 0} & = \mathrm{v}_3 \\
1\ 0\ 1\ 1\ 1 & \\
1\ 1\ 0\ 0\ 0 & \\
1\ 1\ 1\ 0\ 1 & \\
\mathrm{V} &
\end{array}
$$

Fig. 3. *An affine space $A = a \oplus V$ and the corresponding vector space V. The points v_1, v_2, v_3 form the canonical basis for V, and $a = \alpha_A$ is the canonical translation point of A (note that $V = a \oplus A$).*

An interesting property of affine spaces is that $(\alpha \oplus V \equiv V) \Leftrightarrow \alpha \in V$, i.e., if we choose as α a point of V then the affine space A is the vector space V itself. For example, let V be the vector space {000, 010, 011, 001}, then $A = 010 \oplus V =$ {000, 010, 011, 001} $= V$, because $010 \in V$. Thus, a vector space is an affine space.

If A is an affine space, there exists a *unique* vector space V such that for all α in A, $A = \alpha \oplus V$. Such space can be computed as $V = \alpha \oplus A$, where α is *any* point of A. Moreover, if A and A' are affine spaces over the same vector space V, then A and A' either coincide or are disjoint.

Example 3. *Consider the affine space $A =$ {0010, 0011, 0100, 0101}. Our aim is to find the unique vector space V such that $A = \alpha \oplus V$. Choosing $\alpha = 0010 \in A$, we have: $V = \alpha \oplus A = 0010 \oplus A =$ {0000, 0001, 0110, 0111}. It is easy to verify that choosing a different vector in A as translation point we achieve the same result, i.e., $V = 0010 \oplus A = 0011 \oplus A = 0100 \oplus A = 0101 \oplus A$.*

Let $A = \alpha \oplus V$ be an affine space. The *dimension* of A is the dimension of the corresponding vector space V. Since the translation point α can be chosen as any vector of A, and the vector space V can be represented by any of its bases, we need to define a unique representation of A. To this end we introduce some notation. A set of k points in $\{0,1\}^n$ (e.g., an affine or vector space) can be arranged in a $k \times n$ matrix whose rows correspond to the points, and whose columns correspond to the variables $x_1, x_2, \ldots, x_n$ (see for example Figure 3).

A matrix of points in $\{0,1\}^n$ is in *binary order* if its rows (points) are sorted as increasing binary numbers. For example the two matrices in Figure 3 are in binary order.

Definition 2. *Let A be an affine space over a vector space V. The* canonical translation point *α_A is the minimum point of A in binary order.*

For example, 00001 is the translation point of the affine space A in Figure 3.

Definition 3. *Let V be a vector space whose matrix is sorted in binary order, with the rows indexed from 0 to $2^k - 1$. And let $A = \alpha \oplus V$ be an affine space over V. The set of points of V with indices $2^0, 2^1, \ldots, 2^{k-1}$ will be called the* canonical basis *B_A of V (or, equivalently, of A).*

For example, the vectors $\{00101, 01010, 10010\}$ in rows $1, 2, 4$ form the canonical basis of the affine space A in Figure 3.

As proved in [Ciriani 2003], the canonical basis B_A is indeed a basis of V in the algebraic sense, i.e., the points of B_A are linearly independent.

DEFINITION 4. *The* canonical representation (α_A, B_A) *of an affine space A is given by its canonical translation point together with its canonical basis.*

For example, the canonical representation of the affine space A in Figure 3 is $\alpha_A = 00001$ and $B_A = \{00101, 01010, 10010\}$.

We can note that the canonical basis corresponds to the basis derived by a matrix in *reduced row echelon form* [Cohn 1981; Liebler 2003]:

DEFINITION 5. *A matrix is in* reduced row echelon form *if all of the following conditions are met:*

(*1*) *each* leading coefficient *(i.e., the first nonzero entry of each row) is equal to 1;*
(*2*) *rows of all zeros appear last;*
(*3*) *for each pair of successive rows that are not all zeros, the leading coefficient of the first row comes in an earlier column than the leading coefficient of the following row;*
(*4*) *each* pivotal column *(i.e., a column that contains the leading coefficient of one row) has only one nonzero entry.*

A matrix is in row echelon form *whenever the first three conditions are satisfied.*

Note that, because of conditions 2 and 3, all entries below and to the right of a leading coefficient are zero.

As the reduced row echelon form of a matrix is unique, the canonical representation uniquely specifies an affine space (see [Ciriani 2003] for more details).

We partition now the Boolean variables of an affine space in two sets as follows.

DEFINITION 6. *Let $A = \alpha_A \oplus V$ be an affine space with canonical basis $v_1, \ldots, v_k$. For each v_i, let x be the variable corresponding to the first 1-component from left of v_i, i.e., the variable corresponding to the leading coefficient of row v_i in the matrix representing A. The variable x is called* canonical variable. *The variables that are not canonical for any vector in the canonical basis are called* non-canonical.

For example in Figure 3, the canonical variables are: 1) x_3 for vector $v_1 = 00\mathbf{1}01$, 2) x_2 for vector $v_2 = 0\mathbf{1}010$, and 3) x_1 for vector $v_3 = \mathbf{1}0010$. The non-canonical variables are the remaining variables x_4 and x_5.

Observe that the canonical variables are the truly independent variables in the space A, in the sense that they can assume all possible combinations of 0-1 values. On the contrary, on A the non-canonical variables are not independent because they can be defined as linear combinations (i.e., EXORs) of the canonical ones. This fact is clearly expressed by the characteristic function of an affine space, represented by an algebraic expression involving AND and EXOR operators. In fact, as shown in [Ciriani 2003], an affine space can be represented by a simple expression (called *pseudoproduct*) consisting in an AND of EXORs or literals. (For example $x_2\overline{x}_1(x_3 \oplus x_4 \oplus \overline{x}_5)(x_3 \oplus x_7)$ is a pseudoproduct.)

The characteristic function of an affine space can be expressed in various ways as a pseudoproduct. Among these forms, a canonical (CEX) expression given in [Luccio and Pagli 1999] is of particular relevance. In the following definition we explain how to derive the CEX expression of a given affine space A (the direct connection between the canonical basis of an affine space and its CEX expression is explained in more details and proved in [Ciriani 2003]).

DEFINITION 7. *Let $A = \alpha_A \oplus V$ be an affine space. The* CEX(A) expression *of A is given by a product of EXOR factors such that:*

(*1*) *each non-canonical variable appears in exactly one EXOR factor, and each EXOR factor is composed by one non-canonical variable and possibly some canonical variables;*

(*2*) *the canonical variables that appear in the EXOR factor corresponding to the non-canonical variable x are the canonical variables (if any) appearing with value 1 in the vectors of B_A where x is equal to 1;*

(*3*) *all the canonical variables are not complemented; a non-canonical variable is complemented in its EXOR factor if and only if its corresponding component in α_A is* 0.

EXAMPLE 4. *Let $A = \alpha_A \oplus V$ be an affine space with canonical basis $B_A =$* {001010, 010110, 100000} *and translation point $\alpha_A =$* 000100*. The canonical variables are x_1, x_2 and x_3, and x_4, x_5 and x_6 are the non-canonical ones. The first vector in B_A shows that the canonical variable x_3 is in the EXOR factor corresponding to the non-canonical variable x_5. Vector* 010110 *shows that the canonical variable x_2 is in the two EXOR factors corresponding to the non-canonical variables x_4 and x_5. Vector* 100000 *shows that the canonical variable x_1 does not appear in any EXOR factor. By point* (1) *of Definition 7, we have three EXOR factors (one for each non-canonical variable):*

- *$x_2 \oplus x_4$, corresponding to the non-canonical variable x_4 and containing the canonical variable x_2, which is the canonical variable with value 1 in the vector of B_A where x_4 is 1;*
- *$x_2 \oplus x_3 \oplus \overline{x}_5$, corresponding to the non-canonical variable x_5 and containing the canonical variables x_2 and x_3, that are equal to 1 in the two vectors of B_A where x_5 is 1;*
- *and $\overline{x}_6$, corresponding to the non-canonical variable x_6 and not containing any canonical variable, as x_6 is always 0 in the vectors of B_A.*

Note that, by point (3) *of Definition 7, the vector $\alpha_A =$* 000100 *shows that the non-canonical variable x_4 is not complemented while x_5 and x_6 are complemented. Therefore the CEX expression is $(x_2 \oplus x_4)(x_2 \oplus x_3 \oplus \overline{x}_5)\overline{x}_6$.*

By the definitions of CEX expressions and canonical variables, the non-canonical ones are the variables with the higher index in each EXOR factor of the CEX expression. For example $CEX(A) = (x_1 \oplus \overline{x}_2) \cdot x_5 \cdot (x_1 \oplus x_3 \oplus \overline{x}_6) \cdot (x_4 \oplus x_7) \cdot (x_3 \oplus x_4 \oplus x_9)$ is the CEX expressions of an affine space in $\{0,1\}^9$ with canonical variables x_1, x_3, x_4, x_8, and non-canonical variables x_2, x_5, x_6, x_7, x_9. Note that, x_8 is a canonical variable of A but does not appear in its CEX expression.

3. D-REDUCIBLE FUNCTIONS BASED ON AFFINE SPACES

In this section we define the class of *D-reducible Boolean functions based on affine spaces* (shortly, *D-reducible functions*), and analyze their properties. Informally, D-reducible functions are functions whose minterms are contained in an affine space strictly smaller than the whole Boolean space $\{0,1\}^n$.

Definition 8. *The Boolean function $f : \{0,1\}^n \to \{0,1\}$ is* D-reducible on an affine space *if $f \subseteq A$, where $A \subset \{0,1\}^n$ is an affine space of dimension strictly smaller than n.*

Definition 9. *Let f be a D-reducible function. The smallest affine space containing f is called its* associated affine space*.*

For example, consider the function $f = \{00010, 01000, 10010, 10110, 11000\}$ that is D-reducible since is entirely contained in the affine space $A = \{00010, 00110, 01000, 01100, 10010, 10110, 11000, 11100\}$. The vector space corresponding to A (i.e., $V = 00010 \oplus A$) is $V = \{00000, 00100, 01010, 01110, 10000, 10100, 11010, 11110\}$ and its canonical basis is $B = \{00100, 01010, 10000\}$. We note that A is the smaller affine space containing f and its CEX expression is $(x_2 \oplus x_4)\overline{x}_5$.

Observe that A can be a vector space. The reason why we consider affine spaces, instead of vector spaces, is that the smallest affine space containing a function f can have dimension a unit smaller than the dimension of the smallest vector space containing f. For instance, the smallest vector space containing the parity function is $\{0,1\}^n$, while the smallest affine space is the set of binary vectors corresponding to the parity itself, i.e., the set of vectors with odd Hamming weight, which has dimension $n-1$.

We now prove some properties of D-reducible functions. We first show that there exists a unique minimal affine space A containing a given function f.

Proposition 1. *The smallest affine space containing a Boolean function f is unique.*

Proof. Let us suppose that $f \subseteq A_1$ and $f \subseteq A_2$. We first observe that A_1 and A_2 must be affine spaces over the same vector space (this can be verified by some algebraic manipulation). Thus the thesis easily follows since two affine spaces over the same vector space either coincide or are disjoint, and in our case $A_1 \cap A_2 \neq \emptyset$ since both contain f. $\square$

The following proposition makes explicit the relation between f and its projection f_A onto the affine space A.

Proposition 2. *Let f be a D-reducible function and A its associated affine space. Then*

$$f = \chi_A \cdot f_A$$

where χ_A is the characteristic function of A and f_A is the projection of f onto A, i.e., $f_A \subseteq \{0,1\}^{\dim A}$ is the characteristic function of the set $f \cap A$.

Proof. For any Boolean function f and any subset $A \subseteq \{0,1\}^n$, we can decompose f as $f = \chi_A \cdot f_A + \chi_{\overline{A}} \cdot f_{\overline{A}}$, where f_A and $f_{\overline{A}}$ are the projections of f onto A and $\overline{A}$, respectively: $f_A = \chi_{f \cap A}$, $f_{\overline{A}} = \chi_{f \cap \overline{A}}$. Let f be D-reducible, and let A be its associated affine space. Since $f \subseteq A$, $f_{\overline{A}}$ is the constant zero function, and the thesis immediately follows. □

The reduced function f_A contains less variables than the original function f, as proved in the following corollary.

Corollary 1. *Let f be a D-reducible function, and A its associated affine space. The function f_A depends on the $d = \dim A$ $(< n)$ canonical variables of the affine space A.*

Proof. As observed in Section 2, the d canonical variables are the truly independent variables on the affine space A, while the non-canonical variables can be defined on A as EXOR combinations of the canonical ones. Thus, we can replace each occurrence of a non canonical variable in f_A, with the corresponding EXOR combination of canonical variables. In this way, we derive an algebraic expression defining f_A that contains only the d canonical variables. □

Since A is an affine space, we finally have

Proposition 3. *The function χ_A can be expressed as a pseudoproduct.*

Proof. The thesis immediately follows from the fact that, as shown in [Ciriani 2003] and recalled in Definition 7, an affine space can be represented by a pseudoproduct. □

Example 5. *Consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$ whose Karnaugh map is shown on the left side of Figure 1. f is D-reducible, and its associated affine space is described by the CEX expression $(x_1 \oplus \overline{x}_4)$. If we project f onto the Boolean space of dimension 3, represented in the Karnaugh map on the left side of Figure 1 with circles, we obtain the function $f_A = \{001, 010, 011, 101, 110\}$, represented in the Karnaugh map on the right side of the figure, which depends only on the canonical variables of A, i.e., x_1, x_2, and x_3.*

In summary, a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where f_A depends only on $\dim A$ variables (see Corollary 1). Moreover χ_A, the characteristic function of the affine space A covering f, is a pseudoproduct with $(n - \dim A)$ EXOR factors, each containing a different non-canonical variable. The $\dim A$ variables on which f_A depends are exactly the canonical variables of A. Indeed the non-canonical variables depend, through the EXOR factors of χ_A, on the canonical ones.

Notice that a function depending on n variables can be D-reducible only if the number of its minterms is less or equal to 2^{n-1}, i.e., if the number of minterms is less than 50% of all input combinations. Indeed, an affine space of dimension at most $n - 1$ can contain no more than 2^{n-1} points. In particular, in order to reduce the number of variables from n to $\dim A$, the number of minterms of f must be less or equal to $2^{\dim A}$. In other words, the ratio of minterms of f gives a bound on its D-reducibility. This fact suggests a possible refinement of our technique: whenever the number of minterms is too high, one could study and try to reduce the complement of the function f, i.e., the function $\overline{f}$.

Finally, we observe that a similar approach has been proposed in [Czajkowski and Brown 2008], where Gaussian elimination is used to decompose the truth table of a logic function, represented in matrix form, into linearly independent set of

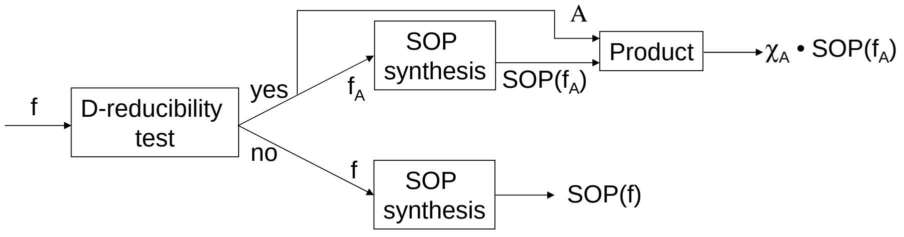

Fig. 4. *Synthesis strategy.*

subfunctions. The basic idea is to find subfunctions that can be reused in a logic expression in order to reduce the size of the logic implementation. This method depends heavily on the variable partitioning, as varying the assignment of variables to rows and columns, the decomposition of the resulting truth table changes. Our technique is different as we work on the products covering the function, and the affine space associated to a function is unique and independent on the variable ordering.

4. SYNTHESIS OF D-REDUCIBLE FUNCTIONS

We now show how the property of D-reducibility can be exploited to perform the synthesis of a Boolean function. Remember that a D-reducible function f can be written as $f = \chi_A \cdot f_A$, where χ_A is the characteristic function of A and f_A is the projection of f onto A (see Proposition 2). Intuitively, the idea is that of reducing the minimization of f to the minimization of f_A, which depends on a reduced number of variables.

The synthesis strategy is depicted in Figure 4. The output function f is first tested for D-reducibility. If f is not D-reducible a classical SOP synthesis is performed. Meanwhile, if the function exhibits the D-reducible property, the reduced function f_A is synthesized in SOP form ($SOP(f_A)$), and the corresponding DRed-SOP circuit is computed by performing a product between the characteristic function of the affine space A (i.e., χ_A) and $SOP(f_A)$. Observe that the D-reducibility test, besides deciding whether a function f is D-reducible or not, derives the reduced function f_A and the characteristic function of A. As already observed, the proposed synthesis strategy could be improved in the following way: whenever f is not D-reducible, the test for D-reducibility should be performed on its complement $\overline{f}$. If $\overline{f}$ is D-reducible, then one could compute a DRedSOP for $\overline{f}$ and obtain a network for f adding a final NOT gate.

Let us start by showing how to efficiently test whether a function is D-reducible, and derive its associated affine space. We then describe the overall synthesis algorithm for completely and incompletely specified Boolean functions.

4.1 D-reducibility Test from Minterms

Recall that a completely defined Boolean function $f : \{0,1\}^n \to \{0,1\}$ can be simply represented by the subset of $\{0,1\}^n$ containing the points p such that $f(p) = 1$, i.e., the on set of f.

Given a completely defined Boolean function, we can perform the D-reducibility

```
D-reducibility Test from minterms

INPUT: Function $f$ represented by a set of minterms in $\{0,1\}^n$
OUTPUT: The D-reducibility of $f$ and, if $f$ is D-reducible, the functions $f_A$ and the
affine space $A$
NOTATION: $\mathbf{0}$ is the zero vector

if ($\mathbf{0} \in f$) then $v = \mathbf{0}$ else $v =$ any minterm of $f$
$g = v \oplus f$
$V_A =$ GaussJordanElimination($g$) // the smallest vector space containing $g$
if ($\dim V_A == n$) then return (“no”, $\emptyset$, $\emptyset$) // $f$ is not D-reducible
else // $f$ is D-reducible
    $A = v \oplus V_A$
    $Var_c =$ canonical variables of $A$
    $f_A =$ Project($f$, $Var_c$) // deletes the non canonical variables
    return (“yes”, $f_A$, $A$)
```

Fig. 5. D-reducibility test from a set of minterms.

test by applying a classical linear algebra tool on its minterms: the *Gaussian elimination.*

Recall that the Gaussian elimination algorithm performs elementary row operations on a given matrix in order to obtain an equivalent matrix in row echelon form. Therefore, the whole procedure for reducing a matrix in reduced row echelon form is to first reduce it to row echelon form applying Gaussian elimination, and then work from right to left forcing zeros above each leading coefficient. The complete process is called *Gauss-Jordan elimination* [Liebler 2003].

Let $m = |f|$. If we execute the Gauss-Jordan elimination on the $m \times n$ matrix whose rows are the minterms of the function, we then obtain an equivalent matrix in row echelon form, from which we can easily get a basis for the smallest vector space containing f. As already noted we are interested in getting the smallest *affine* space containing f, since its dimension can be smaller. To this aim, we first note that if the zero vector is a minterm of f, then A is a vector space (indeed, whenever an affine space contains the zero vector, then it is actually a vector space). Otherwise, f can be contained in an affine space that is not a vector space.

We derive A performing the algorithm shown in Figure 5. We first pick any minterm of f, say v, and compute the set $v \oplus f$. If the zero vector belongs to f, we choose $v = \mathbf{0}$. We then compute the smallest vector space V_A containing $v \oplus f$ by Gauss-Jordan elimination. We finally derive $A \supset f$ from V_A as $A = v \oplus V_A$, where v is the same vector chosen in the first step. Note that, whenever f does not contain the zero vector, we can choose any minterm $v \in f$ in the first step without changing the result, i.e., the affine space A. This is a consequence of Proposition 1.

The time complexity of this D-reducibility test is polynomial in n and $|f|$. More precisely the computational cost of the Gauss-Jordan elimination is $O(n|f|^2)$. Since $|f|$ is often exponential in the number of variables n, the complexity of the test can be exponential in n. In Section 4.2 we discuss a more efficient test that computes the associated affine space of a function f in time polynomial in the original SOP representation of f, and not in the number of its minterms.

```
D-reducibility Test from SOP

INPUT: The set of products $P$ (in PLA form) representing the function $f$ in $\{0,1\}^n$
OUTPUT: The D-reducibility of $f$ and, if $f$ is D-reducible, the functions $f_A$ and the
affine space $A$
NOTATION: $\mathbf{0}$ is the zero vector.
When one of the operands is "-", the bitwise $\oplus$ operator returns "-".
The vector $e_i$ is the zero vector with a unique 1 at position $i$.

if ($\mathbf{0} \in f$) then $v = \mathbf{0}$ else $v =$ any minterm of $f$
$g = v \oplus P$
$GE = \emptyset$ // set of points that are in input to the GaussJordan elimination
for each $p \in g$
    for $i = 1, \ldots, n$
        if ($p[i]$ == "-") then
            $GE = \{e_i\} \cup GE$
            $p[i]$ = "0"
    $GE = \{p\} \cup GE$
// note that $GE$ contains points without don't cares
$V_A$ = GaussJordanElimination($GE$) // the smallest vector space containing $GE$
if ($\dim V_A == n$) then return ("no", $\emptyset$, $\emptyset$) // $f$ is not D-reducible
else // $f$ is D-reducible
    $A = v \oplus V_A$
    $Var_c$ = canonical variables of $A$
    $f_A$ = Project($f$, $Var_c$) // deletes the non canonical variables
    return ("yes", $f_A$, $A$)
```

Fig. 6. D-reducibility test from a SOP expression.

THEOREM 1 **Correctness**. *If a Boolean function f, represented by its on set, is D-reducible, the algorithm shown in Figure 5 computes the smallest affine space A containing f (i.e., A is the associated affine space of f).*

PROOF. Suppose, by contradiction, that there exists an affine space A' smaller than A that contains f. Let v be the minterm of f computed by the algorithm. Note that $v \in A'$, since $v \in f$ and A' contains f. Thus, we have that $V' = v \oplus A'$ is the vector space of A' and it contains $v \oplus f$. Moreover, V' has dimension smaller than $V_A = v \oplus A$, by hypothesis. This contradicts the fact that V_A is computed by the algorithm, using the Gauss-Jordan elimination method, as the smallest vector space containing $v \oplus f$. □

4.2 D-reducibility Test from SOP

The test algorithm described in Section 4.1 considers functions represented by their minterms. Generally Boolean functions are represented by SOP expressions containing cubes and not only minterms (e.g., PLAs). In this section we explain how to perform the D-reducibility test starting from a SOP of a function, *without generating all its minterms.* The complexity of this new version of the test becomes $O(nP^2)$, where P is the number of products in the given SOP for f. Observe that in practical cases, we often have $P << |f|$.

We describe the idea starting with an example. Let $f = \{000000, 001000, 010001,$

$010011, 011001, 011011\}$ be represented with the SOP: $\overline{x}_1\overline{x}_2\overline{x}_4\overline{x}_5\overline{x}_6 + \overline{x}_1 x_2\overline{x}_4 x_6$. The product $\overline{x}_1 x_2\overline{x}_4 x_6$ is represented in PLA form by the row 01-0-1. For each don't care in the product, we can generate a vector composed by all zeros but a 1 in the position corresponding to the don't care. For instance, in our example we generate the vectors 001000 and 000010. These vectors would be surely generated during the Gauss-Jordan elimination step. In fact we have: $001000 = 010001 \oplus 011001$ and $000010 = 010001 \oplus 010011$. The matrix to be processed by the Gauss-Jordan elimination algorithm will then contain: the original vector with 0 instead of the don't cares (010001) and the new generated vectors (001000 and 000010).

The D-reducibility test from a SOP expression is shown in Figure 6. Notice that a product is a particular pseudoproduct and represents an affine space $A = \alpha \oplus V_A$ of 2^d points, where d is the number of don't cares. Moreover the basis of V_A is a subset of the *standard basis* of $\{0, 1\}^n$, i.e., $e_1 = 100\ldots00$, $e_2 = 010\ldots00$, $\ldots$, $e_n = 000\ldots01$. Therefore our idea is to represent a product only with $d+1$ vectors instead of 2^d minterms. These $d+1$ vectors are the basis of V_A, together with α. Moreover, we add a vector of the basis of V_A if and only if it has not been already used for representing another product. In conclusion the P products in the given SOP of f are transformed into at most $P+n$ vectors in input to the Gauss-Jordan elimination algorithm.

In the former example, the first product can be represented by 000000 and 001000, and the second by 010001, 001000 and 000010. Thus, the input to the Gauss-Jordan elimination step is given by the set of vectors: $\{000000, 001000, 010001, 000010\}$. Note that the vector $e_3 = 001000$ has been written only once.

We finally observe that the contraction of the set GE is necessary since we cannot set generic products as input of the Gauss-Jordan elimination. Indeed, the Gauss-Jordan elimination procedure needs as inputs completely specified points.

THEOREM 2 **Correctness**. *If a Boolean function f, represented by a SOP form, is D-reducible, the algorithm shown in Figure 6 computes the smallest affine space A containing f (i.e., A is the associated affine space of f).*

PROOF. By the proof of Theorem 1 we only need to prove that also this algorithm inserts in V_A the smallest vector space containing $v \oplus f$. Recall that the Gauss-Jordan elimination of a generic set of points Q computes the smallest vector space covering Q. We have to show that the smallest vector space containing $v \oplus f$ and the smallest vector space containing GE are equivalent. For this purpose it is sufficient to prove that 1) any minterm of $v \oplus f$ can be generated from the points of GE, and that 2) any point of GE can be generated from the minterms of $v \oplus f$.

1) let q be a minterm of $v \oplus f$, and let p a product in the PLA $v \oplus P$ that covers q. Observe that p always exists since P covers f. Let E be the subset of the standard basis generated by the algorithm starting from p (i.e., e_i is in E if and only if p contains a don't care "-" in position i); and let p_0 be the point generated from the product p setting all the don't cares "-" to 0 (as generated by the algorithm). The point q can be easily generated by the EXOR of p_0 with the points of E that correspond to the "-"s of p, which are in turns 1s in q. Note that $\{p_0\}$ and E are subsets of GE.

2) let q be a point of GE, we have two cases i) q is a point of the standard basis, i.e., $q = e_i$. In this case, the algorithm inserts q in GE since there exists a product p

```
Synthesis Algorithm

INPUT: Function $f$ in $\{0,1\}^n$
OUTPUT: A DRedSOP for $f$, if $f$ is D-reducible. A SOP for $f$, otherwise

(DRed, $f_A$, $A$) = D-reducibility Test($f$)
if (DRed == "no")
then // $f$ is not D-reducible
    return SOP($f$)
else // $f$ is D-reducible
    $\chi_A$ = CEX ($A$) // CEX expression of the affine space $A$
    return $\chi_A \cdot$ SOP($f_A$)
```

Fig. 7. Synthesis Algorithm for completely specified functions.

in g containing a don't care in position i. Let p' be the point derived from p setting all the don't cares of p to 0, and let p'' be the point derived from p setting all the don't cares of p to 0, but the one corresponding to i that is set to 1. First, note that both p' and p'' are covered by p and therefore are in $v \oplus f$. Second, note that p' and p'' differ only for the bit i, thus their EXOR is the point e_i. ii) q corresponds to a product p of g where all the don't cares are set to 0. In this case q is a point of p and thus is in $v \oplus f$. $\square$

4.3 Synthesis Algorithm

The synthesis strategy of a D-reducible function $f = \chi_A \cdot f_A$ is described in Figure 7. After the testing phase, we represent χ_A using its CEX, getting an EXOR-AND network, and we then minimize f_A in SOP form [Brayton et al. 1984; Coudert 1995]. (Observe that f_A can be synthesized in any logical framework, e.g., in three-level-logic form [Debnath and Sasao 1997a; 1997b; 1998; Dubrova et al. 1995; 1999; Perkowski 1995; Sasao 1995].) The synthesis of f_A could be easier than the synthesis of f, since f_A depends on $\dim A < n$ variables. Moreover, the size of the network for f_A can be smaller than the size of the corresponding network for f. Indeed f and f_A have the same number of minterms, but f_A is defined in a smaller space and its minterms are less sparse.

For example consider the function $f = \{0010, 0100, 0110, 1011, 1101\}$, whose Karnaugh map is shown on the left side of Figure 1. f is D-reducible, and its associated affine space is described by the CEX expression $(x_1 \oplus \overline{x}_4)$. We can project f onto the Boolean space of dimension 3, represented with circles in the Karnaugh map on the left side of Figure 1. We can therefore study the function f_A, represented in the Karnaugh map on the right side of the figure. f_A depends only on the canonical variables of A, i.e., x_1, x_2, and x_3.

Notice that we have the same number of minterms, but these are now compacted in a smaller space, i.e., the minterms of the function are more adjacent and we have more chance to merge them into cubes. Suppose we want to synthesize f and f_A in the classical SOP framework. We have $f = \overline{x}_1 x_3 \overline{x}_4 + \overline{x}_1 x_2 \overline{x}_4 + x_1 \overline{x}_2 x_3 x_4 + x_1 x_2 \overline{x}_3 x_4$, and $f_A = \overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3$. The new form for f is then $f = (x_1 \oplus \overline{x}_4)(\overline{x}_2 x_3 + \overline{x}_1 x_2 + x_2 \overline{x}_3)$. Figure 2 shows the resulting network for the function f.

```
Synthesis Algorithm for incompletely specified functions

INPUT: Function $f = (f^{on}, f^{dc})$ in $\{0,1\}^n$
OUTPUT: A DRedSOP for $f$, if $f$ is D-reducible. A SOP for $f$, otherwise
NOTATION: $f^{on}$ is the on set of $f$ and $f^{dc}$ is the don't care set of $f$

(DRed, $f_A^{on}$, $A$) = D-reducibility Test($f^{on}$)
if (DRed == "no")
then // $f^{on}$ is not D-reducible
    return SOP($f$)
else // $f^{on}$ is D-reducible
    $\chi_A$ = CEX ($A$) // CEX expression of the affine space $A$
    $Var_c$ = canonical variables of $A$
    $f_A^{dc}$ = Project($A \cap f^{dc}$, $Var_c$) // deletes the non canonical variables
    $f_A = (f_A^{on}, f_A^{dc})$
    return $\chi_A \cdot$ SOP($f_A$)
```

Fig. 8. Synthesis Algorithm for incompletely specified functions.

4.4 Synthesis of Incompletely Specified D-reducible Functions

Let us now consider f as a incompletely specified function, i.e., f is a Boolean function such that $f : \{0,1\}^n \to \{0,1,-\}$. A point p of the Boolean space $\{0,1\}^n$ such that $f(p) = -$ is called don't care. Thus, the don't care set (or DC set) of f contains all the don't cares for f, while the on set contains all the points p of f such that $f(p) = 1$. Let us now briefly discuss how to extend the notion of D-reducibility to functions with don't care points. The extension to incompletely specified functions is important because synthesis techniques usually benefit from the presence of don't cares.

Our current approach to the synthesis of completely specified D-reducible functions projects onto the affine space A only the on set of the function. However, we think that the synthesis of D-reducible functions would be greatly improved by projecting onto A also the don't care set. Therefore, we propose the following approach.

In order to keep the dimension of A as small as possible, we still define A as the smallest affine space covering only the on set of a function:

Definition 10. *An incompletely specified function $f : \{0,1\}^n \to \{0,1,-\}$ is* D-reducible on an affine space *if its on set can be covered by an affine space of dimension strictly smaller than n.*

Once A has been derived, we project onto A not only the ones of f, but also its *don't care set.* The points of the don't care set that are not covered by A are not considered (see the algorithm in Figure 8).

For example, consider the incompletely specified function $f = (f^{on}, f^{dc})$, with on set $f^{on} = \{0011, 0100, 0101, 1000, 1110\}$ and don't care set $f^{dc} = \{0010, 0111, 1001, 1101\}$. The affine space is derived exclusively from f^{on}, and is $A = \{0011, 0010, 0100, 0101, 1000, 1001, 1110, 1111\}$. We then project onto A both $f^{on} \cap A$ $(= f^{on})$ and $f^{dc} \cap A = \{0010, 1001\}$. The CEX expression of the affine space A is $(x_1 \oplus x_2 \oplus x_3)$, thus its canonical variables are x_1, x_2 and x_4, while the non canonical variable (which will be deleted in the projection phase) is x_3. The projected function $f_A = (f_A^{on}, f_A^{dc})$ is thus composed by the on set $f_A^{on} = \{001, 010, 011, 100, 110\}$ and the don't care set $f_A^{dc} = \{000, 101\}$.

Also for incompletely specified functions, we can improve the synthesis strategy. Observe that, when a function $f = (f^{on}, f^{dc})$ is not D-reducible, the test must be performed on the new function $f' = (\overline{f}^{on} \setminus f^{dc}, f^{dc})$.

5. RELATIONS WITH DEGENERATE AND AUTOSYMMETRIC FUNCTIONS

Let us now discuss the relationship among the class of D-reducible functions, and two other classes of Boolean functions: *degenerate functions* and *autosymmetric functions.*

Recall that a Boolean function is said to be degenerate if it does not depend on all its input variables. D-reducible functions depend in general on all their n input variables, even if we can study them in a space of dimension strictly smaller than n. Thus, in general D-reducible functions are not degenerate. Moreover, degenerate functions are not in general D-reducible. For instance, the function $f(x_1, x_2, x_3) = x_2 \vee x_3$ is degenerate, but not D-reducible; while the function $f(x_1, \ldots, x_n) = x_1 \oplus x_2 \oplus \ldots \oplus \overline{x}_n$ is D-reducible since it is contained in the $n-1$ dimensional space of vectors with even Hamming weight, and it is not degenerate.

Autosymmetric functions, introduced in [Luccio and Pagli 1999] and studied in [Bernasconi et al. 2003], exhibit a regular structure that can be exploited by synthesis algorithms. We recall here their definition.

Definition 11. *A Boolean function f in $\{0,1\}^n$ is* closed under $\boldsymbol{\alpha}$*, with $\boldsymbol{\alpha} \in \{0,1\}^n$, if for each $\boldsymbol{w} \in \{0,1\}^n$, $f(\boldsymbol{w} \oplus \boldsymbol{\alpha}) = f(\boldsymbol{w})$.*

Each function is obviously closed under the zero vector $\mathbf{0}$. As proved in [Luccio and Pagli 1999], if a function f is closed under two different vectors $\boldsymbol{\alpha}_1$, $\boldsymbol{\alpha}_2 \in \{0,1\}^n$, it is also closed under $\boldsymbol{\alpha}_1 \oplus \boldsymbol{\alpha}_2$. Therefore the set L_f of all the vectors $\boldsymbol{\beta}$ such that f is closed under $\boldsymbol{\beta}$ is a vector subspace of $\{0,1\}^n$. L_f is called the *linear space* of f, and k is its *dimension.*

Definition 12. *A Boolean function f is* k-autosymmetric*, or equivalently f has* autosymmetry degree k*, $0 \leq k \leq n$, if its linear space L_f has dimension k.*

For $k \geq 1$ the synthesis of f can be reduced to the synthesis of a *smaller* function f_k, which can be identified in polynomial time. The function f_k is called a *restriction* of f; indeed f_k is "equivalent" to, but smaller than f, it depends only on $n-k$ variables, and has $|f|/2^k$ minterms only, where $|f|$ denotes the number of minterms of f. The new $n-k$ variables are EXOR combinations of some of the original ones.

These results imply that k-autosymmetric functions can be studied in a smaller space whose $n-k$ variables are EXOR combinations of the original ones. However, this fact does not mean that an autosymmetric function is necessarily D-reducible. In fact, there exist D-reducible functions that are not autosymmetric, and autosymmetric functions that are not D-reducible. For instance, the Boolean function $f(x_1, x_2, x_3) = (x_1 \oplus x_2) \vee x_3$ is autosymmetric but not D-reducible, and the Boolean function$f(x_1, x_2, x_3) = x_1x_2x_3$ is D-reducible but non autosymmetric. However, the intersection between the two sets of autosymmetric and D-reducible functions is not empty: for instance the parity function is both autosymmetric and D-reducible.

As we have seen, D-reducibility and autosymmetry represent different, though similar, regularities. Nevertheless, D-reducible functions have an interesting connection with autosymmetric functions, which can be understood by looking at their *Walsh Transform.*

The Walsh transform[4] of a Boolean function f is the rational valued function $\hat{f}$ that defines the coordinates of f with respect to the basis $\{Q_w(x) = (-1)^{w^T x} \mid w \in \{0,1\}^n\}$, where

$$w^T x = \sum_{i=1}^{n} w_i x_i \mod 2$$

denotes the inner product between the two vectors $w, x \in \{0,1\}^n$. More precisely, for any $w \in \{0,1\}^n$ we have

$$\hat{f}(w) = 2^{-n} \sum_{x \in \{0,1\}^n} Q_w(x) f(x) = 2^{-n} \sum_{x \in \{0,1\}^n} (-1)^{w^T x} f(x)\,.$$

Then

$$f(x) = \sum_{w \in \{0,1\}^n} Q_w(x) \hat{f}(w) = \sum_{w \in \{0,1\}^n} (-1)^{w^T x} \hat{f}(w)$$

is the *Walsh expansion* of f.

The connection between autosymmetric and D-reducible functions is expressed by the following theorems.

THEOREM 3. *Let f be a D-reducible function, and let A be its associated affine space. The function defined by the absolute value of the Walsh transform of f, $|\hat{f}|$, is a k-autosymmetric function, with $k = n - \dim A$, and its linear space is $L_{|\hat{f}|} = V_A^{\perp}$, where V_A denotes the vector space corresponding to A.*

PROOF. Let $\alpha \in A$ and $V_A = \alpha \oplus A$. Recall that $V_A^{\perp}$ is defined as follows

$$V_A^{\perp} = \{u \in \{0,1\}^n \mid \forall\, v \in V_A, \quad u^T v = 0\}\,,$$

and its dimension k is given by $k = n - \dim V_A = n - \dim A$. To prove the theorem, we must show that for any $u \in V_A^{\perp}$ and for any $w \in \{0,1\}^n$, $|\hat{f}(w \oplus u)| = |\hat{f}(w)|$. By the definition of Walsh transform, we have

$$\hat{f}(w) = 2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{w^T v} f(v) = 2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{w^T (v \oplus \alpha)} f(v \oplus \alpha)\,.$$

Since $\alpha \in A$ and f is D-reducible with associate affine space A, we have that for any $v \in \{0,1\}^n$

$$f(v \oplus \alpha) = 1 \iff v \oplus \alpha \in A \iff v \in \alpha \oplus A \iff v \in V_A\,.$$

[4]This transform is also called *Abstract Fourier Transform* of a Boolean function.

Moreover, for any $u \in V_A^\perp$ and any $v \in V_A$, $u^T v = 0$. Thus we get

$$\begin{aligned}
\hat{f}(w) &= 2^{-n} \sum_{v \in V_A} (-1)^{w^T(v \oplus \alpha)} f(v \oplus \alpha) \\
&= 2^{-n} \sum_{v \in V_A} (-1)^{u^T v} (-1)^{u^T \alpha} (-1)^{u^T \alpha} (-1)^{w^T(v \oplus \alpha)} f(v \oplus \alpha) \\
&= 2^{-n} \sum_{v \in V_A} (-1)^{u^T(v \oplus \alpha)} (-1)^{u^T \alpha} (-1)^{w^T(v \oplus \alpha)} f(v \oplus \alpha) \\
&= (-1)^{u^T \alpha}\, 2^{-n} \sum_{v \in V_A} (-1)^{(w \oplus u)^T(v \oplus \alpha)} f(v \oplus \alpha) \\
&= (-1)^{u^T \alpha} \hat{f}(w \oplus u)\,,
\end{aligned}$$

and the thesis immediately follows. □

THEOREM 4. *Let f be a k-autosymmetric function. Then the characteristic function $\chi_{\hat{f}}$ of the support of its Walsh transform $\hat{f}$ is a D-reducible function, whose associated affine space is $L_f^\perp$.*

PROOF. We must show that for any $w \notin L_f^\perp$, $\hat{f}(w) = 0$, where

$$L_f^\perp = \{u \in \{0,1\}^n \mid \forall\, w \in L_f, \quad u^T w = 0\}\,.$$

Let $w \notin L_f^\perp$. Thus, there exists $u \in L_f$ such that $u^T w = 1$. By the definition of Walsh transform, we have

$$\hat{f}(w) = 2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{v^T w} f(v) = 2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{(v \oplus u)^T w} f(v \oplus u)\,.$$

Since f is autosymmetric, and $u \in L_f$, $f(v \oplus u) = f(v)$ and we get

$$\hat{f}(w) = 2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{v^T w} (-1)^{u^T w} f(v) = -2^{-n} \sum_{v \in \{0,1\}^n} (-1)^{v^T w} f(v)\,,$$

that implies $\hat{f}(w) = 0$. □

As these two theorems show, a D-reducible Boolean function has an autosymmetric Walsh transform, and *vice versa*, an autosymmetric function has a D-reducible Walsh transform. In other words, D-reducibility is the spectral counterpart of autosymmetry.

6. EXPERIMENTAL RESULTS

In this section we compare the size of the networks described in Section 4.3 (in short DRedSOPs) with the size of the corresponding minimum SOPs. To this end we count the number of literals and the number of gates (OR, AND and EXOR) of an expression. In the multi-level context the cost function is the number of literals in each different gate (see [Eggerstedt et al. 1993; Hachtel and Somenzi 1996]). We observe that in many technologies EXOR and OR (or AND) gates have different costs. In [Hachtel and Somenzi 1996] the authors consider a 2-input EXOR gate as $x \oplus y = \overline{x}y + x\overline{y}$. Thus the cost of a 2-input EXOR gate is 4, while the cost of the 2-input OR and AND gates is 2. Generally, by the associative property of the EXOR operator, we can always see a k-input EXOR gate as the composition of $(k-1)$ 2-input EXOR gates. (The realization is a tree of EXOR gates. Note that an EXOR tree for a k-input EXOR can always be balanced, thus its height is $\lceil \log_2 k \rceil$.) Therefore, we can use a cost function μ where a k-input EXOR gate costs $4(k-1)$, and k-input OR/AND gates cost k. With these measures we compare DRedSOP and SOP expressions. Note that, for SOP expressions the cost μ (that we call μ_{SOP}) corresponds to the sum of the number of literals (L) and different AND gates (A) in the SOP expression, i.e., $\mu_{SOP} = L + A$. For the DRedSOP form of a function f, the cost is $\mu_{DRedSOP} = \mu_{SOP'} + AD + E$, where E is the total cost of the EXOR gates, $\mu_{SOP'}$ is the cost of the SOP of the projected function f_A, and AD is the cost of the final AND gate. In fact $AD = nE + 1$, where nE is the number of EXORs, and 1 is the output of the SOP. For example for the DRedSOP: $(x_1 \oplus x_2)(x_1 \oplus x_3)(x_1x_4 + x_6)$ we have $AD = 2 + 1$ and $\mu_{DRedSOP} = 5 + 3 + 2 * 4$.

Our minimization method has been tested on a range of functions taken from the Espresso benchmark suite [Yang 1991]. CPU times are reported in seconds on a Pentium III $800MHz$ machine with $512MB$ of RAM. The Gauss-Jordan elimination is computed with *Mathematica* 5.0.

In our experiments, we have first computed the number of functions that have at least one D-reducible output in the benchmark suite. The number of such functions is about 70% of the total. The overall number of outputs that are reduceble is 48%. We have then synthesized these functions in order to evaluate whether their DRedSOP network is indeed more compact than the classical minimum SOP form. We have minimized both SOP and DRedSOP forms using Espresso Exact [Brayton et al. 1984]. The size of the resulting networks has been compared using the cost function μ.

Table I reports a cost-oriented comparison among the original optimal SOP forms determined by Espresso exact and the DRedSOP forms for a significant subset of our results. The first column reports the name of the instance, the second column reports the number of inputs and output of the benchmark, the third column shows the number of D-reducible outputs ($\mathbf{O_D}$) and the average number of variables reduced in the D-reducible outputs ($\mathbf{A_D}$). The forth column reports the number of products in each benchmark. The following columns report the cost of the networks together with the computational time in seconds. The cost of the PLA for the SOP form is reported in the fifth column (μ_{SOP}) of the table, while the overall cost of the DRedSOP network is in the eighth column (μ_{DRed}).

We can note that the DRedSOP is not always smaller than the minimum SOP form. This is due to different reasons. First, the EXOR part of the network can be expensive in the CMOS technology. In other technologies where EXOR, AND and OR have the same cost, DRedSOPs are clearly more convenient. Moreover, the EXOR gates considered can have an unbounded number of inputs that give rise to a tree of EXORs with fan in 2. Finally, some functions benefit from the multi-output minimization; after the projection of some outputs, it can happen that the common products are reduced in number. Nevertheless, some benchmarks deeply benefit from the decomposition, see for example *rd84* and *sao2*.

To evaluate the obtained circuits, we ran our benchmarks using the SIS system

Table I. *Synthesis times and network costs of DRedSOPs, and exact SOP forms*

Benchmark				Network Cost					Synthesis time	
Name	**I/O**	**O_D/A_D**	$\|P\|$	μ_{SOP}	$\mu_{SOP'}$	**AD+E**	μ_{DRed}	**gain**	**SOP**	**DRed**
addm4	9/8	4/1.5	189	1407	1380	27	1407	0.00	0.49	0.41
adr4	8/5	1/1.0	255	415	410	13	423	-1.93	0.07	0.05
alu1	12/8	4/1.5	19	60	51	15	66	-10.00	0.23	0.34
b12	15/9	3/1.3	431	233	200	14	214	8.15	1.85	16.90
b2	16/17	1/1.0	104	1970	1998	19	2017	-2.36	0.81	0.79
chkn	29/7	6/2.0	153	1744	1544	27	1571	9.92	0.33	0.40
co14	14/1	1/1.0	14	210	169	81	250	-19.05	0.01	0.01
f51m	8/8	2/1.0	76	402	396	17	413	-2.74	0.11	0.12
in7	26/10	4/2.8	54	427	500	22	522	-22.25	2.38	7.04
intb	15/7	1/1.0	629	5911	5259	9	5268	10.88	11.76	9.13
m181	15/9	3/1.3	430	235	202	14	216	8.06	2.17	21.40
misex2	25/18	17/5.3	29	213	89	140	239	-12.21	0.01	0.01
mlp4	8/8	2/2.0	121	869	846	14	860	1.04	0.95	0.30
mp2d	14/14	6/5.2	123	201	173	64	237	-17.91	1.00	4.97
newtpla	15/5	5/4.6	23	199	112	36	148	25.63	0.01	0.01
rd84	8/4	1/1.0	255	2070	1104	48	1152	44.35	0.16	0.14
sao2	10/4	4/2.0	58	495	289	55	344	30.51	0.05	0.06
t3	12/8	6/1.5	33	251	207	29	236	5.98	0.01	0.03
table3	14/14	1/1.0	175	2643	2737	28	2765	-4.62	0.24	0.23
table5	1715/	4/4.0	158	2503	2588	92	2680	-7.07	0.25	0.30
vg2	25/8	4/7.8	110	914	586	118	704	22.98	0.79	0.86
vtx1	27/6	4/7.8	110	1074	670	116	786	26.82	0.62	2.71
x6dn	39/5	5/1.0	121	818	737	11	748	8.56	0.57	0.60
x9dn	27/7	5/8.8	120	1258	704	130	834	33.70	0.69	2.23
xor5	5/1	1/1.0	16	96	1	26	27	71.88	0.01	0.01

with the MCNC library for technology mapping and the SIS command `map -W -f 3 -s`. Moreover, we have run the `rugged` script of SIS in order to compare the results with a multi-level minimizer. In Table II we compare mapped area and delay of DRedSOPs, the two-level SOPs, and the multi-level networks obtained with the `rugged` script of SIS, for a significant subset of the benchmarks. The first column reports the name of the benchmarks, and the following ones report, by groups of two, the areas and delays estimated by SIS. The first group, labeled DRedSOP, concerns the DRedSOP circuits, the next group refers to two level SOP forms generated by ESPRESSO EXACT. The last group of two columns refers to the multilevel networks generated by SIS.

Consistently with the classical results of multi-level minimization, we can observe that the multi-level always gains in area with respect to bounded-level forms (SOP and DRedSOP), paying with an unbounded number of levels in the resulting circuits that is reflected in the higher delay results.

We can note that the gain measured by our metric is not always consistent with the area measured by SIS, this is meanly due to the technology mapping phase performed by SIS before the measure. Clearly, the technology mapping algorithm can substantially modify the algebraic form.

As we can observe from Table I, a significant number of benchmark functions have a reduced size for their DRedSOP form (the functions that have a positive value in the gain column of the table). Therefore we propose our algorithm as a preprocessing step before the logic synthesis process.

Table II. *Area/delay costs of DRedSOPs, and exact SOP forms*

Benchmark			**DRedSOP**		**Two-level SOP**		**Multi-level SIS**	
Name	**n**	$\lvert P\rvert$	**area**	**delay**	**area**	**delay**	**area**	**delay**
addm4	9	189	1276	42.90	1363	47.90	529	93.70
adr4	8	255	115	12.20	267	19.20	81	25.70
alcom	15	47	217	11.20	220	10.80	160	11.70
alu1	12	19	81	6.80	81	6.80	81	6.80
b12	15	431	226	19.10	206	18.40	133	21.00
chkn	29	153	866	47.10	819	43.60	561	42.90
f51m	8	76	310	20.70	563	31.60	143	51.30
m181	15	430	232	19.70	205	18.40	141	18.90
mlp4	8	121	901	38.00	902	36.40	421	58.60
mp2d	14	123	313	19.70	417	26.00	115	25.10
newapla	12	17	103	16.40	111	16.80	56	20.50
newtpla	15	23	130	19.70	111	19.70	107	16.30
sao2	10	58	344	27.60	332	27.10	202	40.40
t3	12	33	191	16.90	198	21.50	118	35.20
vg2	25	110	395	22.40	354	18.60	149	18.70
vtx1	27	110	384	25.90	344	21.30	156	20.10
x6dn	39	121	899	34.40	1217	36.80	543	37.40
x9dn	27	120	439	26.90	404	23.00	178	20.10
xor5	5	16	16	9.10	16	9.10	16	12.20

7. CONCLUSION

In this paper we have introduced the notion of D-reducibility of a Boolean function f. For a D-reducible function f, depending on n variables, a new function f_A, depending on less than n variables, can be defined and built in polynomial time. This approach supplies a new tool for efficient minimization, based on the idea of exploiting a Boolean function regularity to get more compact expressions.

Our experiments have confirmed the foreseen size reduction, and have also shown that a great number of functions of practical importance are indeed D-reducible, thus validating the overall interest of our approach.

To further increase the size reduction, it would be very interesting to study affine spaces that are represented by a product of 2-EXORs only (a 2-EXOR is an EXOR with two inputs, or a single literal). In this way, in fact, we could partially overcome the unbounded fan in EXORs cost of the CMOS technology.

Future work also includes the study of functions whose minterms can be projected onto subsets of the Boolean space that are not necessarily affine spaces, but whose characteristic functions have compact algebraic expressions.

Finally, it would be interesting to generalize our approach designing an algorithm that computes the smallest affine space A, of dimension strictly smaller than n, containing the majority of minterms of a non reducible function f. In this way, we could represent the *regular* part of f as a DredSOP form, and the other minterms of f as a SOP form.

REFERENCES

Aloul, F., Ramani, A., Markov, I., and Sakallah, K. 2002. Solving Difficult SAT Instances in the Presence of Symmetry. In *ACM/IEEE 39th Design Automation Conference (DAC).*

731–736.

Bernasconi, A. and Ciriani, V. 2006. DRedSOP: Synthesis of a New Class of Regular Functions. In *DSD*. 377–384.

Bernasconi, A., Ciriani, V., and Cordone, R. 2008. On Projecting Sums of Products. In *11th Euromicro Conference on Digital Systems Design: Architectures, Methods and Tools*.

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

Bernasconi, A., Ciriani, V., Luccio, F., and Pagli, L. 2002b. Implicit Test of Regularity for Not Completely Specified Boolean Functions. In *IEEE/ACM 11th International Workshop on Logic & Synthesis (IWLS)*. 345–350.

Bernasconi, A., Ciriani, V., Luccio, F., and Pagli, L. 2003. Three-Level Logic Minimization Based on Function Regularities. *IEEE Trans. on CAD of Integrated Circuits and Systems 22,* 8, 1005–1016.

Brayton, R., Hachtel, G., McMullen, C., and A.L.Sangiovanni-Vincentelli. 1984. *Logic Minimization Algorithms for VLSI Synthesis*. Kluwer Academic Publishers.

Caruso, G. 1991. Near Optimal Factorization of Boolean Functions. *IEEE Transactions on CAD 10,* 8, 1072–1078.

Ciriani, V. 2003. Synthesis of SPP Three-Level Logic Networks using Affine Spaces. *IEEE Trans. on CAD of Integrated Circuits and Systems 22,* 10, 1310–1323.

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

Coudert, O. 1994. Two-Level Logic Minimization: an Overview. *INTEGRATION 17*, 97–140.

Coudert, O. 1995. Doing Two-Level Logic Minimization 100 Times Faster. In *SODA*. 112–121.

Czajkowski, T. and Brown, S. 2008. Functionally Linear Decomposition and Synthesis of Logic Circuits for FPGAs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems 27,* 12, 2236–2249.

Debnath, D. and Sasao, T. 1997a. An Optimization of AND-OR-EXOR Three-Level Networks. In *Asia and South Pacific Design Automation Conference*. 545–550.

Debnath, D. and Sasao, T. 1997b. Minimization of AND-OR-EXOR Three-Level Networks with AND Gate Sharing. *IEICE Trans. Information and Systems E80-D,* 10, 1001–1008.

Debnath, D. and Sasao, T. 1998. A Heuristic Algorithm to Design AND-OR-EXOR Three-Level Networks. In *Asia and South Pacific Design Automation Conference*. 69–74.

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

Debnath, D. and Vranesic, Z. 2003. A Fast Algorithm for OR-AND-OR Synthesis. *IEEE Transactions on CAD 22,* 9, 1166–1176.

Dubrova, E. and Ellervee, P. 1999. A Fast Algorithm for Three-Level Logic Optimization. In *Int. Workshop on Logic Synthesis*. 251–254.

Dubrova, E., Miller, D., and Muzio, J. 1995. Upper Bounds on the Number of Products in AND-OR-XOR Expansion of Logic Functions. *Electronic Letters 31,* 7, 541–542.

Dubrova, E., Miller, D., and Muzio, J. 1999. AOXMIN-MV: A Heuristic Algorithm for AND-OR-XOR Minimization. In *4th Int. Workshop on the Applications of the Reed Muller Expansion in circuit Design*. 37–54.

Eggerstedt, M., Hendrich, N., and von der Heide, K. 1993. Minimization of Parity-Checked Fault-Secure AND/EXOR Networks. In *IFIP WG 10.2 Workshop on Applications of the Reed-Muller Expansion in Circuit Design*. 142–146.

Fišer, P. and Toman, D. 2009. A Fast SOP Minimizer for Logic Funcions Described by Many Product Terms. In *12th Euromicro Conference on Digital Systems Design (DSD)*.

Hachtel, G. and Somenzi, F. 1996. *Logic Synthesis and Verification Algorithms*. Kluwer Academy Publishers.

Ishikawa, R., Hirayama, T., Koda, G., and Shimizu, K. 2004. New Three-Level Boolean Expression Based on EXOR Gates. *IEICE Transactions on Information and Systems* 5, 1214–1222.

Kravets, V. and Sakallah, K. 2000. Generalized Symmetries of Boolean Functions. In *International Conference on Computer-Aided Design (ICCAD)*. 526–532.

Kravets, V. and Sakallah, K. 2001. Constructive Library-Aware Synthesis Using Symmetries. In *Design, Automation and Test in Europe Conference & Exhibition (DATE)*. 208–213.

Liebler, R. 2003. *Basic Matrix Algebra with Algorithms and Applications*. Chapman & Hall/CRC.

Luccio, F. and Pagli, L. 1999. On a New Boolean Function with Applications. *IEEE Transactions on Computers 48,* 3, 296–310.

McGeer, P., Sanghavi, J., Brayton, R., and Sangiovanni-Vincentelli, A. 1993. Espresso-Signature: A New Exact Minimizer for Logic Functions. *IEEE Transactions on VLSI 1,* 4, 432–440.

Perkowski, M. 1995. A New Representation of Strongly Unspecified Switching Functions and its Application to Multi-Level AND/OR/EXOR Synthesis. In *IFIP WG 10.5 Workshop on Applications of the Reed-Muller Expansion*. 143–151.

Sasao, T. 1993. Logic Synthesis with EXOR Gates. In *Logic Synthesis and Optimization*, T. Sasao, Ed. Kluwer, Norwell/MA, USA, Chapter 12, 259–286.

Sasao, T. 1995. A Design Method for AND-OR-EXOR Three Level Networks. In *Int. Workshop on Logic Synthesis*. 8:11–8:20.

Sasao, T. 1999. *Switching Theory for Logic Synthesis*. Kluwer Academic Publishers.

Yang, S. 1991. Logic synthesis and optimization benchmarks user guide version 3.0. User guide, Microelectronic Center.

Responses to Reviewers

Title: Dimension-reducible Boolean Functions based on Affine Spaces
Authors: Anna Bernasconi, Valentina Ciriani

The comments of the reviewers have been incorporated in the paper. Below we give detailed answers to the reviewers comments.

Review Number 1.
The paper is generally well-written, but should replace all instances of "Gauss elimination" with "Gaussian elimination", as http://en.wikipedia.org/wiki/Gaussian_elimination

As suggested, we have replaced all instances of "Gauss elimination" with "Gaussian elimination". We have also introduced and used the term "Gauss-Jordan elimination" to refer to the complete procedure to reduce a matrix in reduced row echelon form. Indeed, this procedure consists in first reducing the matrix to row echelon form, applying "Gaussian elimination", and then working from right to left forcing zeros above each leading coefficient. The complete process is called "Gauss-Jordan elimination".
We have also added a definition to better explain the procedure.

The following proposition explicits the relation.
The following proposition makes explicit the relation
At the end of Section 5 As these two theorems show, a D-reducible Boolean function as an auto-symmetric Walsh transform "as" should probably be "has"

We have corrected the typos.

Please replace the jargon "points" with the more common word "minterm" (I am assuming that they mean the same thing).

We have replaced "point" with "minterm" whenever we refer to a function, we still use the term "point" as a synonymous of "vector" when we refer to vector and affine spaces, as indicated in the notation of the paper.

Also, what about functions whose maxterms are contained in a subspace ? Is it possible to build some kind of duality theory for the notion D-reducibility ?

Of course, whenever a function f is not D-reducible we could study and try to reduce the complement of the function f, i.e., we could compute the smallest affine spaces covering the zeros of f. If the complement $\overline{f}$ of f is D-reducible, then we could compute a DRedSOP for $\overline{f}$ and obtain a network for f adding a final NOT gate. We have mentioned this possibility in the paper.
Applying the concept of D-reducibility to maxterms appears less simple, as a maxterm denotes not a single vector of the Boolean space $\{0,1\}^n$, but a set of vectors.

Empirical comparisons are interesting, they show some wins and some losses. I am not concerned about seeing some losses — modern multicore synthesis tools may *have the resources to run several different algorithms in parallel and either stop when the first reasonable solution is produced, or let all threads finish and choose the best result. However, I am concerned that all comparisons are to SOPs (Table II which is claimed to compare to SIS does not seem to show multilevel circuits produced by SIS).*

We have added a column in Table II containing the multi-level SIS results for comparison. As explained in the paper:
"Consistently with the classical results of multi-level minimization, we can observe that the multi-level always gains in area with respect to bounded-level forms (SOP and DRedSOP), paying with an unbounded number of levels in the resulting circuits that is reflected in the higher delay results."
Moreover, we have explained that Table II contains the results of the technology mapping performed with the command "map -W -f 3 -s" of SIS.

The following paper seems closely related to the reviewed work, but not discussed. Czajkowski, T.S. Brown, S.D. Functionally linear decomposition and synthesis of logic circuits for FPGAs DAC 2008

We have discussed the suggested paper at the end of Section 3.

Review Number 2.
One major concern is the title of this paper and the definition of "D-reducible function." Maybe the author creates this terminology, but it is too general for the concept which is discussed in the paper. This paper only deals with "D-reducible function based on affine space," but we might have more general (and possibly strong) method to obtain D-reduced sub-functions using non-linear Boolean functions. The author mentions it in the conclusion section as future work, but it is not sufficient. The author should append the words "based on affine space" to the title and appropriate points in the paper, e.g. Definition 7.

We have followed your suggestion, and appended the words "based on affine space" in the title, in Definition 8 (the previous Definition 7) and in some more points in the paper.

Experimental part of this paper may have some problems. The author should clarify the following points.
The author reports that 70% of benchmark circuits have at least one D-reducible outputs. How many outputs out of all outputs were reducible? and how many variables were reducible in such D- reducible outputs? It is important information to compare the network cost uSOP and uSOP', so the author had better show it in the table.

We have inserted the required information in the text of the experimental results Section and in Table I.

The proposed method is based on AND-EXOR decomposition as shown in Fig.2, so, if one variable is reducible, the number of minterms in f should be less than 50% of all input combinations. If two variables is reducible, the number of minterms *should be less than 25%. Is it correct? If so, the ratio of minterms gives a bound of D- reducibility in this method. So, the author had better show the such data of each benchmark circuit.*

Yes, it is correct. We have added a paragraph explaining this fact at the end of Section 3 instead of Section 6, since it a structural property of the D-reducible functions. Moreover, since the benchmarks are multi-output functions, is not very clear how to show this property in the experimental results.

In table II, area and delay of the functions done by SIS tool. The experimental setting is not clear. The synthesis result SOPs are factorized by SIS and then technology mapped? Or, the SOPs are not factorized and just technology mapped? If SOP factorization is applied before technology mapping, SIS may find similar decompositions which may be done by DRedSOP, so the experimental setting of SIS command script should be written correctly in detail.

We have added a column in Table II containing the multi-level SIS results for comparison. As explained in the paper:
"Consistently with the classical results of multi-level minimization, we can observe that the multi-level always gains in area with respect to bounded-level forms (SOP and DRedSOP), paying with an unbounded number of levels in the resulting circuits that is reflected in the higher delay results."
Moreover, we have explained that Table II contains the results of the technology mapping performed with the command "map -W -f 3 -s of SIS".

Minor typo: p.2, l.13 from the bottom, We than propose − > We then propose

We have corrected the typo.

Review Number 3.
Comments to the Author The sentence from the beginning of Definition 6 to the end of Example 4. is a dead-copy of the cited paper [Ciriani 2003] (Definition 10 and Example 12). Furthermore, This definition is no well written. Without reading the following example, readers does not understand how to derive the CEX expression. For example, there is no definition of 'connected variable', which is described in Definition 9 of [Ciriani 2003].

We have changed the definition, and we have added more details in the example. The procedure is now better explained and it should be easier to understand.