Compact and Testable Circuits for Regular Functions

Anna Bernasconi
Department of Computer Science
Università di Pisa, Italy
annab@di.unipi.it

Valentina Ciriani
Department of Information Technologies
Università degli Studi di Milano, Italy
valentina.ciriani@unimi.it

Abstract

We propose a new synthesis approach based on the SPP three-level logic minimization of D-reducible Boolean functions. This approach supplies a new tool for efficient minimization, based on the idea of exploiting a Boolean function regularity to get more compact expressions. D-reducible functions can be efficiently synthesized giving rise to new four-level logic forms called DRedSPP. These forms are often smaller than the corresponding minimum SPP forms, and are fully testable under the Stuck-At Fault Model. Moreover, the computational time needed to derive a DRedSPP form for a D-reducible function $f$ is nearly always less than the time required to derive an SPP representation of $f$.

Keywords: logic synthesis, EXOR gates, regular functions, testability.

1 Introduction

The design flow of logic circuits always includes a phase of Boolean function synthesis. In this phase, reduced, and possibly minimal, algebraic forms are determined for the functions, in order to reduce the final size of the corresponding circuits.

As logic minimization is computationally a very hard problem, different strategies have been studied in an attempt to reduce the computational time and, at the same time, to derive more compact forms. Among these strategies, the ones based on the idea of exploiting the “regularity” of functions for their synthesis appear to be very promising. The reason, we might argue, is that non-random functions encoding “real life” problems, as are in general the functions we are interested in minimizing, must exhibit a regular structure that can be reflected in some degree of regularity [1, 2, 3, 4].

In general, it is not always clear whether a Boolean function is “regular”, and which type of regularity could be exploited for its synthesis.

For example, the regularity of a Boolean function has been expressed in [2, 5] by its autosymmetry degree for decreasing the time needed for logic synthesis, moreover, regular Dimension-reducible (shortly, D-reducible) functions have been described in [6, 7] for obtaining compact algebraic forms. Both types of regularity are sufficiently common to make the case interesting, and easy to be tested.

In this paper we further study D-reducible functions and propose a new efficient minimization method for their logic synthesis, resulting in a new four-level logic form called DRedSPP.

Informally, a Dimension-reducible function $f$ is a function whose minterms are contained in an affine space $A$ strictly smaller than the whole Boolean space $\{0, 1\}^n$. Therefore, the function can be represented as $f = \chi_A \cdot f_A$, where $\chi_A$ is the characteristic function of $A$ and $f_A$ is the projection of $f$ onto $A$. As shown in [6], this kind of regularity if quite common: a large percentage (about 70%) of the benchmark functions have at least a D-reducible output.

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 $f$. Note that $f_A$ depends on less variables than $f$, therefore its minimization is easier. This approach thus requires two steps: (i) deriving the space $A$, its characteristic function $\chi_A$, and the projection $f_A$; (ii) minimizing $f_A$ in a given logic framework.

The characteristic function $\chi_A$ can be represented with a product of EXORs (called pseudoproduct), while $f_A$ can be synthesized in any framework of logic minimization, e.g., two level logic, three-level logic, or general multi-level minimization.

In the SOP framework this method turned out to be particularly convenient because if we project a function onto a smaller Boolean space we have the chance of reducing the Hamming distances among its minterms in order to merge them in bigger cubes in the final SOP form [6, 7]. In this paper we instead focus on the three-level form called Sum of PseudoProducts (or SPP), and we prove how our approach to the synthesis of D-reducible functions often turns out to be convenient even in this framework.

Recall that more-than-two level minimization is much harder, but the size of the circuits can significantly de-
increase [8, 9, 10, 11, 12, 13]. In many cases three-level logic is a good trade-off among circuit speed, circuit size, and the time needed for the minimization procedure [14].

SPP networks are three-level EXOR-AND-OR forms introduced in [5] as a direct generalization of SOP forms using EXOR gates. An SPP form consists of the OR of pseudoproducts, where a pseudoproduct is the AND of EXOR factors, i.e., EXOR of literals. For example \((x_1 \oplus x_2) \cdot x_3 + (x_1 \oplus x_3 \oplus x_4) \cdot (x_3 \oplus x_5) + x_2 \cdot x_3\) is an SPP form.

Experimental results show that the average size of SPP forms is approximately half the size of the corresponding SOP. This is obtained paying an increased computational time in the synthesis procedure. As a limit case each EXOR factor reduces to a single literal in SPP, and the SOP and SPP forms coincide.

To be technologically feasible, SPP networks can be restricted to EXOR-AND-OR forms whose EXOR gates have fan-in bounded by a chosen constant \(k\); these forms are called \(k\)SPP [15]. In this paper we will consider only forms with \(k = 2\), i.e., 2SPP forms [16].

Let us start here by giving an intuitive presentation of D-reducibility and of the new algebraic forms we propose for representing D-reducible functions. Consider the function \(f = \{0000, 0110, 1001, 1111\}\) shown in the Karnaugh map on the left side of Figure 1. The function \(f\) is D-reducible, i.e., it can be projected 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, whose description requires less variables (in this case three instead of four). If we synthesize \(f\) and \(f_A\) in the classical SOP framework we obtain \(f = \overline{x_1}x_2\overline{x_3}x_4 + \overline{x_1}x_2x_3\overline{x_4} + x_1\overline{x_2}x_3 + x_1x_2x_3\), and \(f_A = x_1x_2 + x_2x_3 + x_2\overline{x_3}\), respectively.

Observe that the overall number of products has decreased from 4 to 3. As proposed in [6], if we multiply the SOP for \(f_A\) by the characteristic function \(\chi_A\) of the space \(A\), we can derive an algebraic form, called DRedSOP, describing the original function \(f\). For the current example, as \(A\) is represented by the EXOR \((x_1 \oplus x_3)\), we get the form \(f = (x_1 \oplus x_3)(x_1x_2 + x_2x_3 + x_2\overline{x_3})\). This form contains 8 literals, while the SOP for \(f\) contains 14 literals.

If we represent \(f_A\) in SPP form, as we propose in this paper, we get \(f_A = (x_2 \oplus x_3) + x_1x_2\), and multiplying by the characteristic function of \(A\), we get a DRedSPP form for \(f\): \(f = (x_1 \oplus x_3)((x_2 \oplus x_3) + x_1x_2)\). Observe that this new DRedSPP form is more compact than the corresponding DRedSOP form: indeed the number of literals decreases from 8 to 6.

Let us now compare our new DRedSPP form with a minimal SOP form for \(f\): \(f = (x_1 \oplus x_3)(x_2 \oplus x_3) + x_1x_2x_3\).

We can observe that the number of pseudoproducts in this case does not decrease: both \(f\) and \(f_A\) contains two pseudoproducts in their minimal SPP forms. This is a general property, as we will prove in the following sections.

Nevertheless, the new DRedSPP form is more compact than the SOP for \(f\) if we consider the number of literals, that is in general a better measure for the cost of the networks. Indeed, while the minimal SOP for \(f\) contains 7 literals, the DRedSPP form contains 6 literals. This is due to the fact that \(f_A\) depends on less variables, and its pseudoproducts can be described with less literals than the corresponding pseudoproducts for the original function \(f\).

Thus, as shown by this simple example, and confirmed by the experimental results reported in this paper, the approach we propose for the synthesis of D-reducible regular functions, based on the “compression” of the function onto a smaller space, turns out to be useful and convenient even in a three-level logic framework.

Moreover, the experiments show how the proposed algebraic forms can be efficiently synthesized since the computational time needed to derive a DRedSPP form for a D-reducible function \(f\) is nearly always less than the time required to derive an SOP representation of \(f\).

Finally we prove that DRedSPP forms are fully testable in the Stuck-At Fault Model.

The paper is organized as follows. Preliminaries and previous work are summarized in Section 2. Section 3 describes the new DRedSPP circuits based on SOP minimization of \(f_A\), outlines the synthesis procedure, and discusses some properties of these forms. The full testability of DRedSPP circuits under the Stuck-At Fault Model is proven in Section 4. Experimental results are reported in Section 5 and, finally, the paper is summarized in Section 6.

## 2 Preliminaries

We briefly review some basic notions on affine spaces [15], D-reducible functions [6, 7], and SPP and 2SPP networks [15, 16].

### 2.1 Affine Spaces

We work in a Boolean space \(\{0, 1\}^n\) described by \(n\) variables \(x_1, x_2, \ldots, x_n\), where each point is represented by a binary vector of \(n\) components. In this space, 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). Recall that, by known properties of EXORs, if the number of complementations in an EXOR factor \(\hat{x}_1 \oplus \cdots \oplus \hat{x}_m\) (where \(\hat{x_i}\) denote a literal) is even we have that \(\hat{x}_1 \oplus \cdots \oplus \hat{x}_m = x_1 \oplus \cdots \oplus x_m\), while if the number of complementations is odd, we have that \(\hat{x}_1 \oplus \cdots \oplus \hat{x}_m = x_1 \oplus \cdots \oplus \overline{x}_m\). Therefore, we can always represent EXOR factors with at most one complemented variable (conventionally, the variable of higher index).

Let us now extend the symbol \(\oplus\) to denote the elementwise EXOR between two vectors. Then, \(\alpha \oplus \beta\) is the vector
obtained from \( \beta \) complementing in it the elements corresponding to the 1’s of \( \alpha \). For example \( 001 \oplus 101 = 100 \).

\[
\begin{array}{cccc}
x_1 & x_2 & x_3 & x_4 \\
00 & 1 & 0 & 0 \\
01 & 0 & 0 & 0 \\
11 & 0 & 1 & 0 \\
10 & 0 & 1 & 0 \\
\end{array}
\]

\[
\begin{array}{cccc}
x_1 & x_2 & x_3 & \\
00 & 1 & 0 & 0 \\
01 & 0 & 0 & 0 \\
11 & 0 & 1 & 0 \\
10 & 0 & 1 & 0 \\
\end{array}
\]

Figure 1: Karnaugh maps of a D-reducible function \( f \) (on the left) and its corresponding projection \( f_A \) (on the right).

A vector subspace \( V \) of the vector space \( (\{0, 1\}^n, \oplus) \) is a subset of \( \{0, 1\}^n \) containing the zero vector \( 0 = 00 \ldots 0 \), such that for each \( v_1 \) and \( v_2 \) in \( V \) we have that \( v_1 \oplus v_2 \) is still in \( V \). Note that \( V \) is a vector subspace of a vector space if and only if it is itself a vector space.

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 \( \dim V = k \)). 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 \( V = \{000, 010, 100, 110\} \) has three different bases, namely \( \{010, 100\}, \{010, 110\}, \text{ and } \{100, 110\} \).

Given a vector subspace \( V \) of \( (\{0, 1\}^n, \oplus) \), and a point \( \alpha \) in \( \{0, 1\}^n \), we derive an affine space performing the EXOR between \( \alpha \) and each point of \( V \).

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

For example, consider the vector space \( V = \{000, 010, 100, 110\} \) and the vector \( \alpha = 101 \in \{0, 1\}^3 \).

The set \( A = \alpha \oplus V = 101 \oplus V = \{101, 111, 001, 011\} \) is an affine space over \( V \). Note that if we choose \( \alpha \) as any vector of \( A \), we obtain the same result. In this example \( A = 101 \oplus V = 111 \oplus V = 001 \oplus V = 011 \oplus V \).

An interesting property of affine spaces is that \( (\alpha \oplus V \equiv V) \iff \alpha \in V \), i.e., if we choose as \( \alpha \) a point of \( V \) then the affine space \( A \) is the vector space \( V \) itself. 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 \( \alpha \) in \( A \), \( A = \alpha \oplus V \). Such space can be computed as \( V = \alpha \oplus A \), where \( \alpha \) is any point of \( A \). The dimension of \( A = \alpha \oplus V \) is therefore defined as the dimension of the unique corresponding vector space \( V \). Moreover, if \( A \) and \( A' \) are affine spaces over the same vector space \( V \), then \( A \) and \( A' \) either coincide or are disjoint.

Since a vector space can be represented by one of its bases, an affine space \( A \) is characterized by a translation point \( \alpha \) and a basis \( B \) of the corresponding vector space \( V \), i.e., \( A = (\alpha, B) \). The characteristic function of an affine space can be represented by a simple expression (called pseudo-product) consisting in an AND of EXOR factors [15]. In particular, an affine space \( A \) of dimension \( k \) can be represented with a pseudoproduct containing \( (n-1) \) \( \text{EXOR factors} \).

In fact, observe that we can partition the variables into two sets:

1. the set of \( k = \dim A \) canonical variables that are the truly independent variables onto the space \( A \), in the sense that they can assume all possible combinations of 0-1 values;
2. the set of the remaining \( n-k \) non-canonical variables that are not independent onto \( A \), as they are either constant or equal to EXOR combinations of the canonical ones.

Intuitively, the role of each EXOR factor in the pseudoproduct describing \( A \) is to define a non-canonical variable in terms of the canonical ones (see [15] for more details).

For example, \( x_1 x_2 (x_3 \oplus x_4) (x_3 \oplus x_6 \oplus x_7) \) is a pseudoproduct representing the affine space \( A = (\alpha, B) = (0100001, \{0000011, 0000100, 0011001\}) \) of dimension 3. The non canonical variables are \( x_1 \) (always equal to 0 on \( A \)), \( x_2 \) (always equal to 1 on \( A \)), \( x_4 \) (always equal to \( x_3 \) on \( A \)), and \( x_7 \) (always different from \( x_3 \oplus x_6 \) on \( A \)); the remaining variables, \( x_3, x_5, x_6, \) are the canonical ones.

The characteristic function of an affine space can be expressed in various ways as a pseudoproduct. Among these forms, a canonical expression (CEX) was defined in [5].

2.2 D-reducible Functions over Affine Spaces

D-reducible functions, introduced in [6] and further studied in [7], are functions whose minterms are completely contained in an affine space strictly smaller than the whole Boolean cube \( \{0, 1\}^n \).

Definition 2 A Boolean function \( f : \{0, 1\}^n \to \{0, 1\} \) is D-reducible if \( f \subseteq A \), where \( A \subseteq \{0, 1\}^n \) is an affine space of dimension strictly smaller than \( n \).

Let \( f \) be a D-reducible function. The smallest affine space containing \( f \) is unique and is called its associated affine space.

For example, the function \( f = \{0000, 0110, 1001, 1111\} \) shown on the left side of Figure 1, is D-reducible since it is entirely contained in the three-dimensional space \( A = \{0000, 0010, 0100, 0110, 1001, 1011, 1101, 1111\} \).
A D-reducible function \( f \) can be represented through its projection \( f_A \) onto its associated affine space \( A \). In fact, 
\[
\begin{align*}
    f &= \chi_A \cdot f_A,
\end{align*}
\]
where \( \chi_A \) 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 \( f \cap A \). Moreover, \( \chi_A \) is represented with a pseudoproduct containing \((n - \dim A)\) EXOR factors.

Consider again the function \( f \) whose Karnaugh map is shown on the left side of Figure 1. The function \( f \) is D-reducible, and its associated affine space can be described by the pseudoproduct \( (x_1 \oplus x_3) \). If we project \( f \) onto the Boolean space of dimension 3, represented in the Karnaugh map on the left side of Figure 1 with dotted circles, we obtain the function 
\[
    f_A = \{000, 011, 100, 110, 111\},
\]
represented in the Karnaugh map on the right side of the figure, which depends only on the variables of \( A \) (i.e., \( x_1, x_2, \) and \( x_3 \)).

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 test that establishes whether a function \( f \) is D-reducible and the computation of the smallest affine space containing \( f \) can be easily performed in polynomial time [6], by finding the reduced row echelon form [17, 18] of a matrix derived from any SOP representation of \( f \). 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, in polynomial time, starting from any SOP representation of \( f \) without generating all its minterms [6].

### 2.3 SPP and 2SPP Networks

SPP networks are three-level EXOR-AND-OR forms introduced in [5] as a direct generalization of SOP forms. They are obtained generalizing cubes to pseudocubes where literals in cubes may be replaced by EXOR factors in pseudocubes. The repeated union of pseudocubes yields prime pseudocubes, which extend primes of SOPs; once prime pseudocubes are computed, exact minimization of EXOR-AND-OR forms is reduced to the solution of a covering table, as for SOP minimization.

The idea of introducing EXORs in the synthesized networks comes from the observation that EXOR gates could be sensitive to some structural regularities of Boolean functions that are difficult to express using just AND and OR gates, and that should be exploited in the minimization process in order to derive more compact expressions.

Although SPP forms are compact, they have been defined for EXOR gates with unbounded fan-in that seem to be impractical [19] in the current technology. It follows that SPP-like forms with a fixed maximum number of literals in the EXOR factors are much more interesting. Therefore, 2SPP forms, where the number of literals in the EXOR factors is upper bounded by 2, have been proposed and studied [15, 16].

More formally, in a Boolean space \( \{0, 1\}^n \) a 2-EXOR factor is an EXOR with at most two variables, one of which possibly complemented. A 2-pseudoproduct is a product of 2-EXOR factors; a 2SPP form is a sum of 2-pseudoproducts. Moreover, a set of points whose characteristic function can be represented as a 2-pseudoproduct is a 2-pseudocube.

For example, the set of points \( \{1001, 1010, 1101, 1110\} \) in the Karnaugh map in Figure 2 forms a 2-pseudocube, since its characteristic function is the 2-pseudoproduct 
\[
    x_1(x_3 \oplus x_4).
\]
Observe that this 2-pseudoproduct contains the two products \( x_1x_3x_4 \) and \( x_1x_3 \), grouped by the two doted lines in the figure. Moreover, note that any product is a particular 2-pseudoproduct containing only 2-EXOR factors formed by single variables. Thus, any cube is a particular 2-pseudocube.

The 2SPP synthesis problem can be stated as: **given a set of points in the Boolean space \( \{0, 1\}^n \), find its minimal cover composed of 2-pseudocubes**, where a minimal cover is represented by a sum of 2-pseudoproducts with a minimal number of literals or with a minimal number of 2-pseudoproducts.

For example, for the function \( f \) represented by the Karnaugh map in Figure 2, the following 2SPP cover is an expression minimal with respect to both the number of 2-pseudoproducts and the number of literals: 
\[
    x_1(x_3 \oplus x_4) + x_2x_3.
\]
A minimal SOP form of such function is 
\[
    x_1x_3x_4 + x_1x_3 + x_2x_3.
\]
In the exact 2SPP minimization, as in the Quine-McCluskey approach, the generation of prime 2-pseudoproducts is performed in steps by successive unions of 2-pseudoproducts. A minimal 2SPP form is generated by choosing a minimal subset of prime 2-pseudoproducts that covers the original function. Although such an exact method performs well on many examples, it is not affordable for all industrial benchmarks. Therefore, mirroring what has been done for SOP minimization [20, 21], a heuristic minimization procedure for 2SPP was presented in [16].

This procedure, based on the iteration of a suite of operations that generalize the expansion-irredundant-reduction cycle of heuristic SOP minimization, has been
implemented with very good results on industrial benchmarks.

3 DRedSPP forms

In this section we describe the new circuits based on SPP minimization of \( f_A \), we propose a synthesis strategy, and we discuss some properties of these new forms.

3.1 DRedSPP and DRed2SPP Circuits

In [6, 7] the projection of \( f \) onto \( A \), i.e., \( f_A \), is synthesized as a sum of products and the corresponding network is called DRedSOP.

In this paper we minimize the function \( f_A \) in SPP or 2SPP form, and we call the obtained networks DRedSPP or DRed2SPP, respectively.

A D-reducible function \( f \) contained in the affine space \( A \) can be written as \( f = \chi_A \cdot f_A \), and the corresponding circuits is then \( P_A \cdot SPP|_{f_A} \), where \( P_A \) is a pseudoproduct representing \( A \), and \( SPP|_{f_A} \) is an SPP form of the projection of \( f \) onto \( A \). We call DRedSPP form the expression \( P_A \cdot SPP|_{f_A} \). Note that a DRedSPP is a four-level logic form since its SPP block, in input to the final AND gate, has three levels of logic.

If we represent \( f_A \) with a 2SPP form \( 2SPP|_{f_A} \), we then obtain the DRed2SPP form \( P_A \cdot 2SPP|_{f_A} \). Since a 2SPP form is also an SPP expression, a DRed2SPP is indeed a DRedSPP.

When we minimize \( P_A \) and \( SPP|_{f_A} \) (or \( 2SPP|_{f_A} \)), we obtain a minimal DRedSPP (or minimal DRed2SPP) representation.

Definition 3 Let \( f \) be a Boolean function, a minimal DRedSPP form for \( f \) is a circuit denoted by the expression

\[ P_A^{\text{min}} \cdot SPP|_{f_A}^{\text{min}} \]

where \( A \) is the minimal affine space containing \( f \), \( P_A^{\text{min}} \) is a minimal pseudoproduct representing \( A \), and \( SPP|_{f_A}^{\text{min}} \) is a minimal SPP form for the projection of \( f \) onto \( A \).

An analogous definition can be given for minimal DRed2SPPs posing \( 2SPP|_{f_A}^{\text{min}} \) instead of \( SPP|_{f_A}^{\text{min}} \), where \( 2SPP|_{f_A}^{\text{min}} \) is a minimal 2SPP form for \( f_A \). Figure 3 shows the logic network of a DRedSPP. Note that if a minimal SPP form contains 2-EXOR factors only, the minimal DRedSPP form is also a minimal DRed2SPP.

3.2 2DRedSPP and 2DRed2SPP Circuits

Among three-level networks, SPP forms are particularly compact [5, 22]. However SPP forms have the disadvantage to have been originally defined for EXOR gates with unbounded fan-in and, in most technologies, EXOR gates with many inputs are slow, expensive and often impractical [19].

Therefore, 2SPP forms with a fixed maximum number of literals in the EXOR factors have been introduced. For the same reason we can impose a bound (e.g., 2) on the number of literals in the EXOR factors used to represent the affine space \( A \) [7]. Thus, instead of considering the smallest affine space \( A \) covering the minterms of a function \( f \), we can look for the smallest affine space \( A_2 \) covering \( f \) and representable with a 2-pseudoproduct. Since not all affine spaces can be represented by 2-pseudoproducts, in general \( A_2 \supseteq A \).

Mirroring the definition of DRedSPP and DRed2SPP circuits, we can define 2DRedSPP and 2DRed2SPP circuits in an analogous way using only 2-EXOR factors for the representation of the affine space \( A_2 \) covering \( f \).

Hereafter, where not explicitly specified, we will describe properties and synthesis algorithms for DRedSPP and DRed2SPP forms that can be easily extended to 2DRedSPP and 2DRed2SPP circuits.

3.3 Compact Pseudoproducts

We can observe that the main building blocks of DRedSPP and DRed2SPP forms are pseudoproducts (i.e., products of EXOR factors). In fact, \( P_A \) is the pseudoproduct representation of an affine space, moreover an SPP (2SPP) form is a sum of pseudoproducts (2-pseudoproducts, resp.), i.e., we can see these forms as ORs of affine spaces.

In general, the representation of an affine space is not unique, and different pseudoproducts can be characteristic functions of the same affine space. A canonical form (called CEX) has been introduced for giving a unique representation for affine spaces [5], but the CEX expression is not always the more compact representation in terms of number of literals.

For example, the CEX expression \( \pi_1(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_2 \oplus x_4) \) is equivalent to the pseudoproduct \( \pi_1(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_4) \), which is more compact. We need then to find a compact representation of affine spaces in an efficient way. However, finding a minimal pseudoproduct representing an affine space is an NP-hard problem as shown in [7].
The case of 2-pseudoproducts is much simpler, since the CEX representation of sum of 2-EXOR factors is minimal with respect to the number of literals [23]. For example the 2-pseudoproducts $\text{P}_1$, $x_2(x_3 \oplus x_4)(x_3 \oplus x_5)(x_4 \oplus x_5)(x_7 \oplus x_8)$ and $\text{P}_2$, $x_2(x_3 \oplus x_4)(x_4 \oplus x_5)(x_5 \oplus x_6)(x_7 \oplus x_8)$ are equivalent and have the same number of literals. There are no equivalent 2-pseudoproducts containing less literals.

However, the 2SPP minimization is still a hard problem (as SPP minimization) and in practice the heuristic minimization strategies result to be a good compromise between network cost and synthesis time [16, 22]. Therefore, in the next section we propose a heuristic strategy for finding a compact, but not provably optimal, representation for both DRedSPP and DRed2SPP forms. Here we concentrate on pseudoproduct representation in SPP forms, since pseudoproducts are an important part of DRedSPP expressions. A greedy heuristic for computing a compact representation of a pseudoproduct is described in [7] for the DRedSOP. Here we use this approach also for the pseudoproducts in the SPP form. The intuition behind the approach is the following: suppose we have a pseudoproduct $p$ containing two EXOR factors $e_i$ and $e_j$ (i.e., $p = e_i \cdot e_j \cdot r$, where $r$ is the remaining product of EXORs), if $e_i$ and $e_j$ differ in one literal only (i.e., $e_i = e \oplus x_i$ and $e_j = e \oplus x_j$ where $e$ is a non-empty common EXOR term) we have that $p = e_i \cdot (x_i \oplus \pi_j) \cdot r$ and $p = e_j \cdot (x_i \oplus \pi_j) \cdot r$. We can note that if $e$ contains more than a single literal, then the new expression for $p$ is more compact than the first one. For example, let $e_i = x_1 \oplus x_2 \oplus x_3$ and $e_j = x_1 \oplus x_2 \oplus x_4$, thus $e = x_1 \oplus x_2$ and we can write $p$ as $(x_1 \oplus x_2 \oplus x_3)(x_3 \oplus \pi_4)$ that is more compact than $(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_2 \oplus x_4)$. The greedy heuristic considers couples of EXOR factors of a pseudoproduct and tries to find a more compact equivalent couple (for more details see [7]).

In order to use the same approach for SPP forms we have to first note that SPP minimization strategies [22, 15] depend on the canonical CEX representation of the pseudoproducts. Therefore, the canonical representation of a pseudoproduct $p$ should be used during the synthesis, but the, possibly, more compact equivalent form of $p$ is used for computing the real size of the pseudoproduct in terms of literals. For instance, while the CEX expression of the pseudoproduct in the previous example is $(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_2 \oplus x_4)$ the number of literals considered is 5, since the more compact equivalent form is $(x_1 \oplus x_2 \oplus x_3)(x_3 \oplus \pi_4)$. Finally, when the final SPP form is computed all the CEX expressions composing it will be transformed into the more compact ones.

### 3.4 Synthesis Algorithms

We can now describe the entire synthesis procedure (described in Figure 4) for obtaining DRedSPP and DRed2SPP forms of D-reducible functions. The first step is the D-reducible test that computes the affine space $\chi_A$ covering $f$ and the projection $f_A$ of $f$ onto $\chi_A$.

If $f$ is not D-reducible, else case in the algorithm in Figure 4, then $\chi_A$ is the entire Boolean space 1, and $f_A$ is $f$ itself. In this case the algorithm performs the SPP minimization of $f$. If $f$ is D-reducible, if case, the function MinPseudoproduct($\chi_A$) computes the pseudoproduct representing $\chi_A$ in the heuristic way. The MinSPP($f_A$) function computes a minimal SPP form for $f_A$ reducing the CEX expressions where possible (as described in Section 3.3). In this case the DRedSPP is $P_A \cdot SPP_A$.

**Synthesis of DRedSPP**

**INPUT:** A function $f$.

**OUTPUT:** A DRedSPP circuit for $f$, if $f$ is D-reducible.

```plaintext
(\chi_A, f_A) = D-ReducibleTest(f);
if (\chi_A \neq 1) // if $f$ is D-reducible
    P_A = MinPseudoproduct(\chi_A);
    SPP_A = MinSPP(f_A);
    return P_A \cdot SPP_A;
else // if $f$ is not D-reducible
    SPP = MinSPP(f);
    return SPP;
```

**Figure 4:** Algorithm for the synthesis of DRedSPP.

The synthesis strategy of DRed2SPP is analogous, where we perform a 2SPP minimization (Min2SPP) instead of SPP minimization (MinSPP).

### 3.5 Properties

In DRedSOP and 2DRedSOP synthesis, if we project a function onto a smaller Boolean space $A$ we have the chance of reducing the Hamming distance among its minterms in order to merge them in bigger cubes in the final SOP form. For example, consider the minterm 0110 in the Karnaugh map on the left side of Figure 1, its corresponding product $\pi_1 \pi_2 \pi_3$ is prime since no other minterm can be merged with it. If we project the function onto the new space $(x_1 \oplus \pi_4)$, see the right side of Figure 1, its corresponding minterm 011 can be merged with 111 forming the prime product $x_2 x_3$.

This property does not hold anymore if we perform SPP or 2SPP synthesis of the projected function $f_A$. Thus, for the new expressions DRedSPP, 2DRedSPP, DRRed2SPP, and 2DRed2SPP we have no chance to form bigger pseudocubes in $f_A$, as shown in the following theorem. Therefore, the new forms do not have less pseudoproducts, but can have less literals with respect to the corresponding SPP (or 2SPP) forms, since the projected function $f_A$ does not depend on all the variables. For instance, consider again the example of Figure 1. Minimal SOP forms of $f$ and $f_A$ are $f = \pi_1 \pi_2 \pi_3 \pi_4 + \pi_2 x_2 \pi_3 + \pi_3 x_1 \pi_4 + x_1 x_2 x_4$, and $f_A = x_1 x_2 x_3 + \pi_2 \pi_3 \pi_4$. Note that the number of products in $f_A$ and $f$ is 3 and 4, respectively. If we perform the minimization in the SPP (or 2SPP) framework we obtain $f = (x_1 \oplus \pi_4)(x_2 \oplus \pi_3) + x_1 x_2 x_4$, and
Given a $D$-reducible Boolean function $f = \chi_A \cdot f_A$, let us denote with $SPP_f$ its minimal SPP form.

**Theorem 1**

$$|SPP_f| = |SPP_{f_A}|,$$

where $|SPP_f|$ (resp. $|SPP_{f_A}|$) is the number of pseudoproducts of $SPP_f$ (resp. $SPP_{f_A}$).

**Proof.** We first prove that $|SPP_f| \leq |SPP_{f_A}|$. Suppose, by contradiction, that $|SPP_f| > |SPP_{f_A}|$. Since $f$ can be covered by $P_A \cdot SPP_{f_A}$, we can construct an SPP form for $f$ by multiplying by $P_A$ each single pseudoproduct $p$ composing $SPP_{f_A}$. We observe that $P_A \cdot p$ is a pseudoproduct since $P_A$ is a pseudoproduct itself, and then $P_A \cdot p$ is simply the product of two pseudoproducts. Therefore, we obtain an SPP form with $|SPP_f|$ pseudoproducts; this is in contradiction with the facts that $|SPP_f| > |SPP_{f_A}|$ and that $SPP_f$ is minimal.

We now prove that $|SPP_f| \geq |SPP_{f_A}|$. Suppose, by contradiction, that $|SPP_f| < |SPP_{f_A}|$. Since $f$ is $D$-reducible (i.e., $f$ is entirely contained in $\chi_A$), we have that $\chi_A \cdot f = f$. Therefore, $P_A \cdot SPP_f$ and $SPP_f$ are two covers of the same function $f$. If we can note that $SPP(f)$ covers the function $f$ in the space $A$, thus covers $f_A$. We have then the contradiction with $|SPP_f| < |SPP_{f_A}|$. With a similar proof, Theorem 1 holds also for 2SPP minimization.

### 4 Testability

Redundancies in a circuit often correspond to parts of the circuit where area is wasted and may also invalidate tests for testable faults. For this reason, synthesis procedures which result in non-redundant circuits are desirable. In this section, the testability of DRedSPP and 2DRedSPP networks is studied from a theoretical point of view under the Stuck-At Fault Model (SAFM) [24].

Let $C$ be any combinational logic circuit. Recall that a fault in the SAFM fixes exactly one input or output pin of a node in $C$ to constant value (0 or 1) independently of the values applied to the primary inputs of the circuit. A node $v$ in $C$ is called **fully testable**, if there does not exist a redundant fault with fault location $v$. If all nodes in $C$ are fully testable, then $C$ is **fully testable**.

Let us first review some results concerning the testability of SPP and 2SPP networks in the SAFM. A SPP network **minimal** with respect to the number of literals is fully testable under the SAFM, as proven in [25]. The proof of full testability exploits the properties of a minimal network (i.e., primality, irredundancy and minimality with respect to the number of literals).

In the special case of 2SPP networks, minimality is not necessary for guaranteeing full testability: indeed weaker properties are sufficient to obtain fully testable 2SPP networks. More precisely, as shown in [16], a 2SPP network is fully testable in the SAFM if and only if the corresponding algebraic expression is (i) $AND$-irredundant, i.e., the deletion of a factor in any 2-pseudoproduct changes the represented function; (ii) $OR$-irredundant, i.e., the deletion of any 2-pseudoproduct changes the represented function; and (iii) EXOR-irredundant, i.e., the deletion of any literal in a 2-EXOR factor changes the represented function. The 2SPP networks (possibly not optimal) derived with the heuristic presented in [16] satisfy these properties and thus are still fully testable under the SAFM. An $OR$-irredundant, AND-irredundant, and EXOR-irredundant 2SPP form is called EXOR-AND-OR irredundant.

We now study the testability of the proposed networks. We first consider minimal DRedSPP and 2DRedSPP networks, where the projection $f_A$ is represented with an SPP form minimal with respect to the number of literals.

**Theorem 2** DRedSPP and 2DRedSPP networks, whose SPP component is minimal with respect to the number of literals, are fully testable in the Stuck-At Fault Model.

**Proof.** The proof of full testability of DRedSPP and 2DRedSPP networks essentially exploits two properties of these networks: 1) the SPP form of $f_A$ is minimal with respect to the number of literals, thus fully testable in the SAFM; 2) a Stuck-At Fault in an EXOR factor describing the affine space $A$ would change this space, consequently the output function would change making the fault testable. Let us start studying the testability of DRedSPP networks. Let $C$ be a DRedSPP network computing a function $f$. Recall that we can partition the input variables into two sets: the set of $A$ canonical variables, that are the variables on which $f_A$ depends, and the set of the remaining $n - \dim A$ non-canonical variables that are either constant or equal to EXOR combinations of the canonical ones.

To prove the full testability of $C$ we must consider the following cases.

1. **A fault occurs in the circuit representation of $f_A$.** Since $f_A$ is realized with a fully testable circuit (an SPP form minimal with respect to the number of literals), it is fully testable in the SAFM by definition. We only have to show that the fault can be propagated to the output of $C$, in order to be tested. This can be done by setting the non-canonical variables to values satisfying each EXOR factor in the characteristic function for $A$. In this way, all input to the final AND gate, but the one corresponding to $f_A$, are equal to 1, so $C$ computes exactly $f_A$. Moreover all possible values can be applied to its inputs (remember that $f_A$ does not depend on the non-canonical variables).

2. **A fault occurs at the input of one EXOR gate.** Suppose that a stuck-at-1 or a stuck-at-0 occurs at
the input of an EXOR gate. This implies that the affine space \( A \) described by the network changes. In fact, the block of the network that defines \( A \), computes now another affine space \( A' \) such that \( A' \not= A \) and \( \dim A' = \dim A \) (observe that the fault changes the output of the EXOR gate, without eliminating it from \( C \)). Since the smallest affine space containing the on-set of a function is unique, there must exist a point \( x \in \{0,1\}^n \) such that \( f(x) = 1 \), \( x \in A \) and \( x \not\in A' \). Thus, the fault can be tested as \( C \) computes \( 0 \) instead of \( 1 \) on \( x \).

3. A fault occurs at the input of the final AND gate.

Such a fault can be a consequence of a stuck-at fault at the output of the final OR gate of the 2SPP for \( f_A \), (ii) a stuck-at fault at the output of one EXOR gate, (iii) a stuck-at fault at a single literal, corresponding to a non-canonical variable that is constant on \( A \), in input to the final AND gate. The first case has been already considered (see case 1). The last two cases can be handled in the same way. Suppose that, without loss of generality, \( f \not= 0 \), and that a stuck-at-0 occurs. Then, \( C \) computes the constant \( 0 \) function, and the fault can be easily tested by applying in input to \( C \) a configuration of values in \( \{0,1\}^n \) such that \( f \) is equal to \( 1 \). Now suppose that a stuck-at-1 occurs. Since a stuck-at-1 in this case is equivalent to the removal of an EXOR factor from the pseudoproduct describing \( A \), the affine space described by \( C \) changes into a bigger space \( A' \supset A \). Thus, the fault can be tested applying in input to \( C \) a configuration of values in \( \{0,1\}^n \) such that \( f \) takes the value \( 1 \) in all points in the on-set of the function \( f \) computed by \( C \), \( x_i \) and \( x_j \) are different. Now, suppose that a stuck-at-fault fixes the variable \( x_i \) to the constant value \( 0 \). This implies that the EXOR gate computes \( x_j \) instead of \( (x_j \oplus x_i) \); thus the function computed by the faulty circuit \( C \) takes the value \( 1 \) only on points where \( x_j = 1 \). Then, the fault can be tested by applying in input to \( C \) a configuration of values \( x \in \{0,1\}^n \) such that \( x_j = 0 \) and \( f(x) = 1 \).

4. A fault occurs at the output of the final AND gate.

Suppose that a stuck-at-1 (0) occurs at the output of the final AND gate. Suppose that, without loss of generality, \( f \not= 1 \) \( (f \not= 0) \). In this case, the network \( C \) computes the constant \( 1 \) (0) function, and the fault can be easily tested by applying in input to \( C \) a configuration of values in \( \{0,1\}^n \) such that \( f \) is equal to \( 0 \) (1).

The full testability of 2DRedSPP networks can be proved analogously. There is only a case that must be handled differently: the case of a fault occurring at the input of a 2-EXOR gate in the circuit block describing the affine space. Indeed, the affine space \( A_2 \) used in the construction of 2DRedSPP networks is not necessarily the smallest space covering the on-set of the function, therefore we cannot use its unicity to show how the fault can be tested. So, consider a 2-EXOR gate in the circuit \( C \), say \( (x_i \oplus x_j) \). First of all observe that the presence of this EXOR implies that \( (i) \) neither \( x_i \) nor \( x_j \) occur as single literals in the compact 2-pseudoproduct describing the affine space; \( (ii) \) in all points in the on-set of the function \( f \) computed by \( C \), \( x_i \) and \( x_j \) are different. Now, suppose that a stuck-at-fault fixes the variable \( x_i \) to the constant value \( 0 \). This implies that the EXOR gate computes \( x_j \) instead of \( (x_j \oplus x_i) \); thus the function computed by the faulty circuit \( C \) takes the value \( 1 \) only on points where \( x_i = 1 \). Then, the fault can be tested by applying in input to \( C \) a configuration of values \( x \in \{0,1\}^n \) such that \( x_j = 0 \) and \( f(x) = 1 \).

<table>
<thead>
<tr>
<th>Bench</th>
<th>in/out (I)</th>
<th>2SPP area</th>
<th>delay</th>
<th>DRed2SPP area</th>
<th>delay</th>
</tr>
</thead>
<tbody>
<tr>
<td>add5</td>
<td>12/7 (1)</td>
<td>292</td>
<td>25.0</td>
<td>292</td>
<td>25.0</td>
</tr>
<tr>
<td>alu2</td>
<td>10/8 (1)</td>
<td>167</td>
<td>16.7</td>
<td>167</td>
<td>16.7</td>
</tr>
<tr>
<td>and</td>
<td>14/24 (16)</td>
<td>776</td>
<td>34.7</td>
<td>755</td>
<td>34.3</td>
</tr>
<tr>
<td>b12</td>
<td>15/9 (3)</td>
<td>128</td>
<td>13.1</td>
<td>128</td>
<td>13.1</td>
</tr>
<tr>
<td>bench</td>
<td>8/6 (5)</td>
<td>139</td>
<td>15.2</td>
<td>138</td>
<td>14.7</td>
</tr>
<tr>
<td>br1</td>
<td>12/8 (6)</td>
<td>401</td>
<td>28.4</td>
<td>361</td>
<td>30.4</td>
</tr>
<tr>
<td>br2</td>
<td>12/8 (7)</td>
<td>273</td>
<td>25.3</td>
<td>256</td>
<td>24.5</td>
</tr>
<tr>
<td>dc2</td>
<td>4/7 (2)</td>
<td>206</td>
<td>18.7</td>
<td>213</td>
<td>18.8</td>
</tr>
<tr>
<td>dekoder</td>
<td>4/7 (1)</td>
<td>50</td>
<td>10.0</td>
<td>49</td>
<td>10.3</td>
</tr>
<tr>
<td>in2</td>
<td>19/10 (9)</td>
<td>824</td>
<td>39.4</td>
<td>822</td>
<td>37.8</td>
</tr>
<tr>
<td>in7</td>
<td>26/10 (2)</td>
<td>308</td>
<td>25.6</td>
<td>308</td>
<td>25.6</td>
</tr>
<tr>
<td>inc</td>
<td>15/9 (5)</td>
<td>188</td>
<td>18.4</td>
<td>180</td>
<td>18.2</td>
</tr>
<tr>
<td>in1</td>
<td>6/12 (11)</td>
<td>151</td>
<td>18.5</td>
<td>137</td>
<td>15.8</td>
</tr>
<tr>
<td>in18</td>
<td>15/9 (3)</td>
<td>129</td>
<td>13.1</td>
<td>129</td>
<td>13.1</td>
</tr>
<tr>
<td>in3</td>
<td>8/16 (15)</td>
<td>576</td>
<td>33.9</td>
<td>583</td>
<td>31.2</td>
</tr>
<tr>
<td>in4</td>
<td>8/16 (5)</td>
<td>936</td>
<td>37.0</td>
<td>915</td>
<td>42.4</td>
</tr>
<tr>
<td>mp2d</td>
<td>14/14 (2)</td>
<td>204</td>
<td>20.3</td>
<td>204</td>
<td>20.4</td>
</tr>
<tr>
<td>prom2</td>
<td>9/21 (1)</td>
<td>4749</td>
<td>104.1</td>
<td>4659</td>
<td>103.3</td>
</tr>
<tr>
<td>radd</td>
<td>8/5 (1)</td>
<td>119</td>
<td>16.1</td>
<td>119</td>
<td>16.1</td>
</tr>
<tr>
<td>rcki</td>
<td>32/7 (6)</td>
<td>341</td>
<td>49.9</td>
<td>341</td>
<td>49.7</td>
</tr>
<tr>
<td>t1</td>
<td>21/23 (11)</td>
<td>424</td>
<td>21.6</td>
<td>409</td>
<td>22.1</td>
</tr>
<tr>
<td>t4</td>
<td>12/8 (8)</td>
<td>110</td>
<td>13.5</td>
<td>112</td>
<td>14.9</td>
</tr>
<tr>
<td>tms</td>
<td>8/16 (14)</td>
<td>409</td>
<td>29.1</td>
<td>421</td>
<td>30.7</td>
</tr>
<tr>
<td>wim</td>
<td>4/7 (1)</td>
<td>46</td>
<td>9.8</td>
<td>45</td>
<td>10.1</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>498</td>
<td>26.6</td>
<td>488</td>
<td>26.6</td>
</tr>
</tbody>
</table>

Indeed, \( C \) computes \( 0 \) instead of \( 1 \) on input \( x \). Finally observe that such input can always be found. In fact, if \( x_j = 1 \) in all minterms of \( f \), a compact representation of the affine space would contain \( x_j \) instead of \( (x_j \oplus x_i) \).

The presence of a stuck-at-1 in \( (x_i \oplus x_j) \) and the presence of any stuck-at fault in \( (x_i \oplus \overline{x_j}) \) can be tested in a similar way.

We finally consider the testability of DRed2SPP and 2DRed2SPP networks, where the projection \( f_A \) is represented with an EXOR-AND-OR irredundant 2SPP form, heuristically computed.

**Theorem 3** DRed2SPP and 2DRed2SPP networks, whose 2SPP component is EXOR-AND-OR irredundant, are fully testable in the Stuck-At Fault Model.

**Proof.** The proof is similar to the proof of Theorem 2. ■
5 Experimental Results

In this section we compare 2DRed2SPPs networks with the corresponding minimal 2SPPs. We compare these two forms since 2SPP minimization results to be the more convenient minimization strategy for technological reasons, as explained in Section 3.

Our minimization method has been tested on a range of functions taken from the ESPRESSO benchmark suite [26]. We report in the following a significant subset of the functions as representative indicators of our experiments. The methods have been implemented in C, and the experiments have been run on a Pentium 1.6GHz CPU with 1 GByte of RAM.

Table 2: Synthesis times (in seconds) of 2DRed2SPP and 2SPP forms.

<table>
<thead>
<tr>
<th>Bench</th>
<th>in/out (D)</th>
<th>2SPP time</th>
<th>DRed2SPP time</th>
</tr>
</thead>
<tbody>
<tr>
<td>addb</td>
<td>12/7 (1)</td>
<td>47.3</td>
<td>41.4</td>
</tr>
<tr>
<td>alu2</td>
<td>10/6 (1)</td>
<td>1.8</td>
<td>1.8</td>
</tr>
<tr>
<td>amd</td>
<td>14/24/16</td>
<td>119.4</td>
<td>95.3</td>
</tr>
<tr>
<td>bl2</td>
<td>15/9 (3)</td>
<td>0.2</td>
<td>0.4</td>
</tr>
<tr>
<td>br1</td>
<td>12/8 (8)</td>
<td>5.9</td>
<td>0.9</td>
</tr>
<tr>
<td>br2</td>
<td>12/8 (7)</td>
<td>4.7</td>
<td>1.0</td>
</tr>
<tr>
<td>dc2</td>
<td>4/7 (2)</td>
<td>0.7</td>
<td>0.2</td>
</tr>
<tr>
<td>dekoder</td>
<td>4/7 (1)</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>m2</td>
<td>19/10 (9)</td>
<td>544.7</td>
<td>281.8</td>
</tr>
<tr>
<td>m7</td>
<td>26/10 (2)</td>
<td>835.1</td>
<td>780.7</td>
</tr>
<tr>
<td>inc</td>
<td>7/9 (5)</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>m1</td>
<td>6/12 (11)</td>
<td>0.4</td>
<td>0.6</td>
</tr>
<tr>
<td>m181</td>
<td>15/9 (3)</td>
<td>33.3</td>
<td>27.3</td>
</tr>
<tr>
<td>m3</td>
<td>8/16/15</td>
<td>7.2</td>
<td>4.0</td>
</tr>
<tr>
<td>m4</td>
<td>8/16 (5)</td>
<td>18.0</td>
<td>13.1</td>
</tr>
<tr>
<td>mp2d</td>
<td>14/14 (2)</td>
<td>14.6</td>
<td>13.7</td>
</tr>
<tr>
<td>prom2</td>
<td>9/21 (1)</td>
<td>39.3</td>
<td>54.7</td>
</tr>
<tr>
<td>radd</td>
<td>8/5/1</td>
<td>0.7</td>
<td>0.8</td>
</tr>
<tr>
<td>rckl</td>
<td>32/7 (6)</td>
<td>1683.0</td>
<td>938.1</td>
</tr>
<tr>
<td>tl</td>
<td>21/23 (11)</td>
<td>2636.3</td>
<td>2545.9</td>
</tr>
<tr>
<td>t4</td>
<td>12/8 (8)</td>
<td>8.0</td>
<td>3.0</td>
</tr>
<tr>
<td>tms</td>
<td>8/16/14</td>
<td>2.3</td>
<td>1.4</td>
</tr>
<tr>
<td>wmn</td>
<td>4/7 (1)</td>
<td>0.1</td>
<td>0.2</td>
</tr>
<tr>
<td>Average</td>
<td></td>
<td>262.77</td>
<td>200.31</td>
</tr>
</tbody>
</table>

We have minimized both 2SPP expressions and the projected part of 2DRed2SPP forms using the 2SPP heuristic minimization strategy described in [16].

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 reducible is 48%. We have then synthesized these functions in order to evaluate whether their 2DRed2SPP network is indeed more compact than the minimum 2SPP form. In Table 1 we compare mapped area and estimated delay, of 2SPP and 2DRed2SPP circuits for the considered benchmarks. To evaluate the obtained circuits after technology mapping, we ran our benchmarks using the SIS system with the MCNC library for technology mapping and the SIS command map \(-W -f 3 -s\).

The first column reports the name of the considered benchmarks. The second column contains the number of inputs and outputs, in/out, and the number of D-reducible outputs, (D), of the benchmarks. The following ones report, by groups of two, the areas and delays estimated by SIS after technology mapping. While the first group concerns the circuits synthesized in 2SPP form, the next group refers to 2DRed2SPP circuits.

We can note that 2DRed2SPP circuits are not always smaller than the minimal 2SPP forms. This is due to different reasons. First, the EXORs in the network representing A can be expensive in the CMOS technology. In other technologies where EXOR, AND and OR have the same cost, 2DRed2SPP are clearly more convenient.

Moreover, 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 have an interesting benefit from the decomposition, see for example br1 (10% of gain in area) and m1 (9% of gain in area).

In Table 2 we compare computational times (in seconds) required to synthesize 2SPP and 2DRed2SPP circuits for the same subset of benchmarks.

Again, the first column reports the name of the considered benchmarks, and the second column contains the number of inputs and outputs, in/out, and the number of D-reducible outputs, (D). The following two columns report the computational times required by the minimization algorithms for 2SPP and 2DRed2SPP synthesis.

We can note that the minimization time nearly always benefits from the decomposition in 2DRed2SPPs forms. Just some small benchmarks show a greater synthesis time for 2DRed2SPPs due to the time overhead of the D-Reducible Test. The time gain can be quite relevant: for example, the 2DRed2SPP form exhibits a 44% time gain on instance rckl.

6 Conclusion

A new synthesis approach based on the SPP three-level logic minimization of D-reducible functions has been proposed. This approach supplies a new tool for efficient minimization, based on the idea of exploiting a Boolean function regularity to get more compact expressions.

As future work, it would be interesting to generalize our approach to 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.

Another interesting future direction is the study of new function regularities, and the corresponding efficient synthesis strategies, in order to find compact and regular multilevel logic circuits.
References


