Università di Pisa
Dipartimento di Informatica

Technical Report: TR-05-18

Testability of SPP Three-Level Logic Networks in Static Fault Models

Valentina Ciriani,
Department of Information Technologies
University of Milano
26013 Crema (CR), Italy
ciriani@dti.unimi.it

Anna Bernasconi,
Department of Computer Science
University of Pisa
56100 Pisa, Italy
annab@di.unipi.it

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

July 14, 2005

Address: via F. Buonarroti 2, 56127 Pisa, Italy. Tel: +39 050 2212700 Fax: +39 050 2212726

Testability of SPP Three-Level Logic Networks in Static Fault Models *

Valentina Ciriani,
Department of Information Technologies
University of Milano
26013 Crema (CR), Italy
ciriani@dti.unimi.it

Anna Bernasconi,
Department of Computer Science
University of Pisa
56100 Pisa, Italy
annab@di.unipi.it

Rolf Drechsler
Institute of Computer Science
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

July 14, 2005

Abstract

Recently introduced, three-level logic Sum of Pseudoproducts (SPP) forms allow the representation of Boolean functions with much shorter expressions than standard two-level Sum of Products (SOP) forms, or other three-level logic forms.

In this paper the testability of circuits derived from SPPs is analyzed. We study testability under static Fault Models (FMs), i.e. the Stuck-At Fault Model (SAFM) and the Cellular Fault Model (CFM). For SPP networks several minimal forms can be considered. While full testability can be proved in the SAFM for some forms, SPP networks in the CFM are shown to contain redundancies. Finally, we propose a method for transforming non-testable networks into testable ones. Experimental results are given to demonstrate the efficiency of the approach.

1 Introduction

An important aspect of logic synthesis is the problem of deriving high-quality design from the initial specifications. A given Boolean function may be realized by a large variety of circuits, very different in terms of structure. In this framework the selection of a logic network, out of all possible known models (SOP [9], Reed Muller [22], EXSOP [10, 12], OR-AND-OR [11], SPP [5, 19], ESPP [18]), is critical and depends on multiple factors. Moreover it is very difficult to define a theoretical model that captures the problem in its generality. Thus the objective is to synthesize a circuit that optimizes a cost function involving different factors. In particular we are interested in several features like, e.g., *(i)* the size of the algebraic expression, in order to estimate the area occupied by the logic gates; *(ii)* the number of levels in the network, in order to estimate the delay of the longest path through the gates; *(iii)* the implementability of the network in the

*Parts of this paper have been published in *IFIP 12-th VLSI-SOC* (2003) [7].

current technologies; *(iv)* the existence of efficient minimization algorithms; *(v)* the testability properties of the network; *(vi)* the power consuming of the network.

The standard synthesis is performed with Sum of Products (SOP) minimization procedures, leading to two-level circuits. More-than-two level minimization is much harder, but the size of the circuits can significantly decrease. In many cases three-level logic is a good trade-off among circuit speed, circuit size, and the time needed for the minimization procedure [21]. Algorithms for exact minimization have worst case exponential complexity, hence the time to attain minimal forms may become huge for increasing size of the input.

In this paper we focus on a special three-level network called *Sum of k-Pseudoproducts* (k-*SPP*) and on the more general *Sum of Pseudoproducts* (*SPP*). This choice is motivated by the fact that SPP networks often satisfy the above-mentioned properties, as observed below.

SPP expressions, introduced in [19], can be seen as a direct generalization of SOP expressions 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). In the recent paper [18] a modified version of SPP networks, called ESPP and consisting of an EXOR of pseudoproducts, has been proposed.

Among three-level networks, SPP forms are particularly compact [4, 5]. However SPP forms have two major disadvantages: *(i)* they require large computational effort for the minimization; *(ii)* they have been originally defined for EXOR gates with unbounded fan-in, but in most technologies, EXOR gates with many inputs are slow, expensive and often not easily implementable [25]. Therefore, in recent studies [4, 5, 6], *k-SPP forms* with a fixed maximum number of literals (k) in the EXOR factors have been introduced.

Experimental results [4, 5, 6] show that the size of the k-SPP minimal forms is not significantly larger than the one for unbounded fan-in, but the computational effort drastically decreases, especially when $k = 2$. Thus, 2-SPP forms are reasonable upper bounds of the exact SPP forms, and are a good trade-off between the compactness of SPP forms and the efficiency of SOP minimization. Furthermore 2-SPP forms require a reduced number of different EXOR gates and are more practicable for the current technology. Moreover, preliminary results on multipliers indicate that SPP networks are also low power consuming [8].

Beside the synthesis aspect, testability is a major aspect of the design process. For this, aspects of testability should be considered from the very beginning [26]. For several two-level forms detailed studies on testability have been performed. But, to the best of our knowledge, for three-level networks testability has not been considered so far.

In this paper the testability of SPP forms is studied from a theoretical and practical point of view. We study testability of SPP forms under two static fault models, i.e. the Stuck-At Fault Model (SAFM) and the Cellular Fault Model (CFM).

The classical stuck-at fault model (SAFM) is well-known and used throughout the industry for many years [3, 1] and still represents the state-of-the-art. In SAFM it is assumed that a defect causes a basic cell input or output to be fixed to either 0 or 1. Thus, all failures with this effect will be detected by tests for stuck-at faults. The strongest cell-based fault model that controls the correct static behavior of a combinational circuit is the CFM, which tries to completely verify the function computed by each basic cell in the circuit [15]. The investigations with respect to CFM and SAFM as well are usually based on the single fault assumption, i.e. one assumes that there is at most one fault (according to the considered fault model) in the circuit. Under the stuck-at fault model it is proved that general SPP networks, minimized with respect to the number of literals, are free of redundancies by construction. Whereas it can be shown by counter-examples that SPPs, minimized with respect to the number of pseudoproducts, are not fully testable. The same results hold for the specific class of 2-SPPs. Then, the circuits are studied with respect to the more general CFM. In this fault model SPP forms are shown to contain redundancies, and a characterization of the faulty networks is provided. Experimental results for the SAFM are given to demonstrate the efficiency of the approach.

Furthermore, the non-fully testable networks have been studied in order to improve their testability. In particular, 2-SPP networks minimal with respect to the number of 2-pseudoproducts can be transformed into minimal 2-SPP forms that are fully-testable.

The paper is structured as follows: In Section 2 notation and definitions are given. The static fault models are introduced and basics on SPP networks are reviewed. The testability results for the SAFM and the CFM are presented in Section 3. In Section 4 we analyze the (negative) results on testability, and show how it is possible to improve the testability of 2-SPP and SPP networks in the SAFM e in the CFM. Finally, in Section 5 details on the experimental setup and the practical results are given.

2 Preliminaries

2.1 Fault Models

Let C be any combinational logic circuit over a fixed library. We recall the definitions of the two classical static fault models: the cellular fault model and the stuck-at fault model.

2.1.1 Cellular Fault Model (CFM)

In CFM [15] it is assumed that a fault modifies the behavior of exactly one node v in a given circuit C and that the modified behavior is still combinational. Since this fault can be detected by observing the incorrect output values of v for one suitable input combination, it suffices to test for faults of the following kind:

Definition 1 *A* cellular fault *in C is a tuple $(v, I, X/Y)$, where v is the faulty node (= fault location), I is an input to v for which v does not behave correctly, and X (Y) is the output of the correct (faulty) node on input I.*

2.1.2 Stuck-at Fault Model (SAFM)

A fault in the SAFM [3] causes exactly one input or output pin of a node in C to have a fixed constant value (0 or 1) independently of the values applied to the primary inputs of the circuit. In the following we simply speak of stuck-at-i (s-a-i) faults.

2.1.3 Definition and Notation for SAFM and CFM

We finish the discussion of CFM and SAFM with some general definitions and remarks on the relation between the two fault models. For this, let C be a circuit and FM a fault model as defined above.

Definition 2 *An input t to C is a* test *for a fault F in* FM *iff the primary output values of C on applying t in the presence of F are different from the output values of C in the fault free case.*

Example 1 *Consider the circuit in Figure 1. A s-a-0 fault at the output of the gate $(x_1 \oplus x_2)$ can be tested by setting inputs x_1, x_3, and x_4 to 1 and x_2 to 0. With the s-a-0 fault at the output of the gate $(x_1 \oplus x_2)$ the network returns 0, while it should return 1.*

Now suppose that the upper EXOR gate is faulty in the CFM and outputs x_2 instead of $(x_1 \oplus x_2)$. Even this fault can be tested by setting x_1, x_3, and x_4 to 1 and x_2 to 0.

The goal of any test pattern generation process is a *complete* test set for the circuit under test in the considered fault model FM, i.e. a test set that contains a test for each testable fault. It easily follows from the definitions that, given a circuit C, a complete test set in CFM is also complete in SAFM. Thus, the cellular fault model is stronger than the stuck-at fault model.

The construction of complete test sets requires the determination of the faults which are not testable (= *redundant*), even though it is easy to see that in general the detection of redundancies is a *coNP-complete* problem. Redundancies have further unpleasant properties: they may invalidate tests for testable faults and often correspond to locations of the circuit where area is wasted [3]. For this, synthesis procedures which result in non-redundant circuits are desirable. 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 called *fully testable*.

2.2 2-SPP Networks

In this section we recall some basic definitions from [4, 5, 6].

In a Boolean space $\{0,1\}^n$ described by n variables x_1, x_2, ..., x_n, a 2-*EXOR factor* is an EXOR with at most 2 variables, one of which possibly complemented (an EXOR with just one literal corresponds to the literal itself). Given two Boolean variables x_1, x_2, all the possible 2-EXOR factors are essentially x_1, $\overline{x}_1$, x_2, $\overline{x}_2$, $(x_1 \oplus x_2)$, and $(x_1 \oplus \overline{x}_2)$ (in fact, $\overline{x}_1 \oplus x_2 = x_1 \oplus \overline{x}_2$, and $\overline{x}_1 \oplus \overline{x}_2 = x_1 \oplus x_2$).

Definition 3 *A* 2-pseudoproduct *is a product of 2-EXOR factors; and a* 2-SPP form *is a sum of 2-pseudoproducts.*

For example, $x_2x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4) + (x_1 \oplus x_2)(x_1 \oplus \overline{x}_3)x_4$ is a 2-SPP form. A 2-pseudoproduct P of a Boolean function f is *prime* iff no other 2-pseudoproduct P' of f exists such that $P \subseteq P'$. Observe that, unlike products, P' is not always obtained from P by deleting one or more factors. For example, the 2-pseudoproduct $P = (x_1 \oplus x_2)(x_1 \oplus x_3)(x_1 \oplus x_4)$ is contained, among others, not only in $(x_1 \oplus x_3)(x_1 \oplus x_4)$, but also in $(x_2 \oplus \overline{x}_3)(x_1 \oplus x_4)$ and $(x_2 \oplus \overline{x}_3)(x_2 \oplus \overline{x}_4)$.

Definition 4 *A subset of* $\{0,1\}^n$ *whose characteristic function can be represented as a* 2*-pseudoproduct is a* 2-pseudocube.

2-pseudocubes generalize the concept of cubes. A SOP form is a particular 2-SPP form where each EXOR factor contains only one literal.

In the space $\{0,1\}^n$ the number of different 2-EXOR factors with exactly 2 literals is $2 \cdot \binom{n}{2} = n(n-1)$. Thus in the worst case, 2-SPP forms require a quadratic number of different 2-EXOR gates.

The 2-SPP 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.

Example 2 *For the function f represented by the Karnaugh map in Figure 1, the following 2-SPP cover is a minimal expression with respect to* 2*-pseudoproducts:* $(x_1 \oplus x_2)x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$*. The 2-SPP circuit representation is on the right side of the figure. On the other hand, a 2-SPP form minimal with respect to the number of literals is* $\overline{x_2}x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$*. Finally, a minimal SOP form of such function is* $\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_1x_3x_4$*.*

We can observe that a 2-pseudoproduct corresponds to a system of linear equations, and a 2-pseudocube corresponds to the set of solutions of such a system.

Example 3 *The* 2*-pseudoproduct*

$$x_2 \cdot (x_1 \oplus x_3) \cdot (x_3 \oplus \overline{x}_5) \cdot \overline{x}_6 \cdot (x_7 \oplus x_8)$$

x1 x2 \ x3 x4	00	01	11	10
00	1	0	1	0
01	1	0	1	0
11	0	0	0	0
10	0	0	1	0

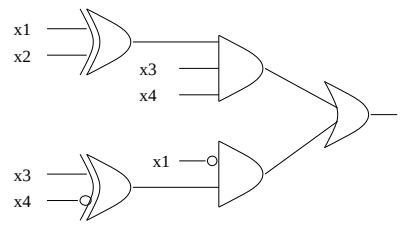

Figure 1: Karnaugh map of function f with a 2-SPP cover $(x_1 \oplus x_2)x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$, minimal with respect to the number of 2-pseudoproducts, and the corresponding 2-SPP circuit representation.

in $\{0,1\}^9$ *corresponds to the system*

$$\left\{\begin{array}{rcl} x_2 & = & 1 \\ x_1 \oplus x_3 & = & 1 \\ x_3 \oplus \overline{x}_5 & = & 1 \\ \overline{x}_6 & = & 1 \\ x_7 \oplus x_8 & = & 1 \end{array}\right. = \left\{\begin{array}{rcl} x_2 & = & 1 \\ x_1 \oplus x_3 & = & 1 \\ x_3 \oplus x_5 & = & 0 \\ x_6 & = & 0 \\ x_7 \oplus x_8 & = & 1 \end{array}\right.$$

When the 2-pseudocube is actually a cube, the system has only one variable in each equation.

A 2-pseudocube can be represented with different 2-pseudoproducts corresponding to different linear systems. For example, the three 2-pseudoproducts $x_1 \cdot x_1 \cdot (x_2 \oplus x_3) \cdot (x_2 \oplus x_4)$, $x_1 \cdot (x_2 \oplus x_3) \cdot (x_2 \oplus x_4) \cdot (x_3 \oplus \overline{x}_4)$, and $x_1 \cdot (x_2 \oplus x_3) \cdot (x_2 \oplus x_4)$ represent the same set of points (i.e., 2-pseudocube): $\{1011, 1100\}$. Of course the most convenient representation is the third one. The corresponding linear systems are:

$$\left\{\begin{array}{rl} x_1 & = 1 \\ x_1 & = 1 \\ x_2 \oplus x_3 & = 1 \\ x_2 \oplus x_4 & = 1 \end{array}\right. = \left\{\begin{array}{rl} x_1 & = 1 \\ x_2 \oplus x_3 & = 1 \\ x_2 \oplus x_4 & = 1 \\ x_3 \oplus x_4 & = 0 \end{array}\right. = \left\{\begin{array}{rl} x_1 & = 1 \\ x_2 \oplus x_3 & = 1 \\ x_2 \oplus x_4 & = 1 \end{array}\right.$$

Observe that only the third system has maximum rank, i.e. its equations are linearly independent, and indeed it corresponds to the smaller 2-pseudoproduct. Therefore minimal 2-SPP forms are sums of 2-pseudoproducts whose systems have maximum rank.

In [6] a 2-SPP minimization algorithm is proposed. As in the Quine-McCluskey approach the generation of prime 2-pseudoproducts is performed in steps by successive unions of 2-pseudoproducts. A minimal 2-SPP form is generated by choosing a minimal subset of prime 2-pseudoproducts that covers the original function (this is the classical set-covering step of Quine-McCluskey optimization).

The *Sum of Pseudoproducts* (or *SPP*) forms are a direct generalization of 2-SPP expressions, where the EXOR factors can have an unbounded number of literals. SPP networks were introduced in [19] and studied in [2, 4, 5, 8, 17]. An SPP form consists of an OR of *pseudoproducts*, where a pseudoproduct is the AND of EXOR factors. As a limit case each EXOR factor reduces to a single literal, then SOP and SPP forms coincide. For example the function $x_2(x_1 \oplus x_3 \oplus x_4 \oplus x_5)(x_1 \oplus \overline{x}_6) + x_2(x_1 \oplus x_3 \oplus \overline{x}_4)$ is an SPP form, but it is not a 2-SPP from.

An *EXOR factor* is a single literal, or a string of different literals connected by EXORs. By some known properties of EXOR gates we represent always EXOR factors with at most one complemented variable (conventionally, the variable of higher index). For example $\overline{x}_1 \oplus \overline{x}_2 \oplus x_4 \oplus x_5 = x_1 \oplus x_2 \oplus x_4 \oplus x_5$ and $\overline{x}_1 \oplus \overline{x}_2 \oplus x_3 \oplus \overline{x}_5 = x_1 \oplus x_2 \oplus x_3 \oplus \overline{x}_5$.

A product P of EXOR factors is a *pseudoproduct* of the Boolean function f if $P \subseteq f$. A pseudoproduct P of f is *prime* if no other pseudoproduct P' of f exists such that $P \subseteq P'$. A pseudoproduct is the characteristic function of a set of points denoted *pseudocube*. Similarly to the 2-SPP minimization problem, the SPP minimization problem can be stated as follows: "Given a Boolean function f, the *SPP minimization problem* is the problem of finding a sum of pseudoproducts with minimum number of factors or literals that is a characteristic function for f".

In summary, in the SPP framework pseudocubes and pseudoproducts play the same role of 2-pseudocubes and 2-pseudoproducts in the 2-SPP synthesis.

Usually SPP and 2-SPP networks are considered three-level-logic networks, in fact these forms have three levels of logic gates (EXOR, AND, OR). In some technologies (e.g., CMOS) EXOR gates are expensive, and are reduced to an OR of AND gates, e.g., $x \oplus y = \overline{x}y + x\overline{y}$. Therefore, in this case, we use a cost function where a k-input EXOR gate costs $4(k-1)$, and k-input OR/AND gates cost k. This cost corresponds to the CMOS cost described in [13]. On the other hand, if we consider FPGA [20] technologies, EXORs with a bounded number of literals are directly implemented. In [13] a different cost function is proposed for the FPGA realization of the networks, where the cost of k-input EXOR gates is, in general, the same of k-input AND/OR gates.

3 Testability

In this section we study the testability of 2-SPP and SPP networks under the SAFM and the more general CFM.

3.1 Testability in the SAFM

As observed in Section 2.2, there exist two different notions of cost function for the minimization of 2-SPP (SPP) forms:

1. The cost function is the total number of 2-pseudoproducts (pseudoproducts) in the form.
2. The cost function is the total number of literals in the form.

In both cases, the minimal forms are prime and irredundant. The full testability of 2-SPP and SPP forms is guaranteed only in the second case, as proved below, while forms minimized with respect to the number of pseudoproducts may contain redundancies.

3.1.1 2-SPP Networks

We first consider 2-SPP forms minimal w.r.t. the number of 2-pseudoproducts.

Theorem 1 *2-SPP forms minimal with respect to the number of 2-pseudoproducts are not fully testable.*

Proof. We provide a counter-example. Consider the function $f = \{0101, 0111, 1001, 1010, 1101, 1110\}$. There are three prime 2-pseudoproducts for f: $(x_1 \oplus x_2)(x_3 \oplus x_4)$, $x_2(x_3 \oplus x_4)$, and $x_1(x_3 \oplus x_4)$. The sum of any couple of them provides a 2-SPP form, prime and irredundant, minimal w.r.t. the number of 2-pseudoproducts.

Let us choose the form $f = (x_1 \oplus x_2)(x_3 \oplus x_4) + x_2(x_3 \oplus x_4)$. Suppose that there is a s-a-0 at the input x_2 of the gate $(x_1 \oplus x_2)$. In this case the output of the 2-pseudoproduct $(x_1 \oplus x_2)(x_3 \oplus x_4)$ is identical to the output of $x_1(x_3 \oplus x_4)$. Therefore the faulty network is equivalent to $x_1(x_3 \oplus x_4) + x_2(x_3 \oplus x_4)$, that is exactly the original function f. ■

We now consider 2-SPP forms minimal w.r.t. the number of literals. We first need a preliminary result. Recall that 2-SPP networks are composed of three levels of logic: a level of 2-EXORs whose inputs are the variables; a level of ANDs whose inputs are the outputs of the EXOR layer; and an OR of the outputs of the AND layer.

Lemma 1 *All possible values can be applied to the inputs of the AND layer of a minimal 2-SPP network.*

Proof. Recall that a 2-pseudoproduct can be seen as a linear system. In a minimal 2-SPP form each 2-pseudoproduct contains a number of 2-EXOR factors equal to the rank of its system. In other words the equations in the corresponding system are linearly independent. This means that the outputs of the EXOR gates are independent, i.e., the inputs to the AND layer have all the possible values. ■

We can now prove the full testability of minimal 2-SPP networks.

Theorem 2 *2-SPP forms minimal with respect to the number of literals are fully testable.*

Proof. Since 2-SPP forms are prime and irredundant, the proof of the full testability for AND and OR gates is the same as for SOP forms. In fact, as proved in Lemma 1, the inputs to the AND gates are directly controllable, i.e., all possible values can be applied. Then we are left only with the case of a s-a-fault at inputs of EXOR gates. We prove by contradiction that any fault can be tested.

Let $(x_i \oplus x_j) \cdot p + s$ be a representation of f in 2-SPP form minimal w.r.t the number of literals, where p is a 2-pseudoproduct and s is the rest of the form.

Let us consider the case $x_i \equiv 0$, i.e., s-a-0 in x_i. Then the network computes the faulty function $f_F = x_j \cdot p + s$. By contradiction suppose that $f_F \equiv f$, then

$$\begin{aligned} x_j \cdot p + s &\equiv (x_i \oplus x_j) \cdot p + s \\ x_j x_i \cdot p + x_j \overline{x}_i \cdot p + s &\equiv \overline{x}_j x_i \cdot p + x_j \overline{x}_i \cdot p + s \\ x_j x_i \cdot p + s &\equiv \overline{x}_j x_i \cdot p + s\,. \end{aligned}$$

Since the 2-pseudocubes represented by the 2-pseudoproducts $x_j x_i \cdot p$ and $\overline{x}_j x_i \cdot p$ have an empty intersection, the last equality implies that they must be both covered by s, and this implies that s covers $x_i \cdot p$. Therefore f contains $(x_i \oplus x_j) \cdot p$ and $x_i \cdot p$. We now observe that

$$x_i \cdot p + (x_i \oplus x_j) \cdot p = x_i \cdot p + x_j \cdot p\,.$$

In fact we have

$$\begin{aligned} x_i \cdot p + (x_i \oplus x_j) \cdot p &= x_j x_i \cdot p + \overline{x}_j x_i \cdot p + x_j \overline{x}_i \cdot p \\ &= x_j x_i \cdot p + \overline{x}_j x_i \cdot p + x_j x_i \cdot p + x_j \overline{x}_i \cdot p \\ &= x_i \cdot p + x_j \cdot p\,. \end{aligned}$$

Therefore we reach a contradiction to the minimality w.r.t. the number of literals of the 2-SPP form for f. The minimal 2-SPP form for f would be $x_j \cdot p + s$ instead of $(x_i \oplus x_j) \cdot p + s$.

The case of negated variables is identical. An analogous proof holds for a s-a-1 fault. ■

3.1.2 SPP Networks

SPP networks have an unbounded number of literals in the EXOR gates. If we consider forms minimal w.r.t. the number of pseudoproducts, then we have the same result as for 2-SPP networks, since the counter-example given in the proof of Theorem 1 still holds.

Consider now SPP forms minimal w.r.t. the number of literals. The result is analogous to the one for 2-SPP forms:

Theorem 3 *SPP forms minimal with respect to the number of literals are fully testable.*

Proof. Following the proof for 2-SPP forms we now have to prove the testability of general EXOR gates. Let $(x_i \oplus h) \cdot p + s$ be a representation of f in SPP form minimal w.r.t the number of literals, where h is an EXOR factor, not including x_i, p is a pseudoproduct and s is the rest of the minimal SPP form.

Let us consider the case $x_i \equiv 0$, i.e. s-a-0 in x_i. Then the network computes the faulty function $f_F = h \cdot p + s$. By contradiction suppose that $f_F \equiv f$, then

$$\begin{aligned} h \cdot p + s &\equiv (x_i \oplus h) \cdot p + s \\ hx_i \cdot p + h\overline{x}_i \cdot p + s &\equiv \overline{h}x_i \cdot p + h\overline{x}_i \cdot p + s \\ hx_i \cdot p + s &\equiv \overline{h}x_i \cdot p + s\,. \end{aligned}$$

Since the pseudocubes represented by the pseudoproducts $hx_i \cdot p$ and $\overline{h}x_i \cdot p$ have an empty intersection, the last equality implies that they must be both covered by s, and this implies that s covers $x_i \cdot p$. Therefore f contains $(x_i \oplus h) \cdot p$ and $x_i \cdot p$. We now observe that

$$x_i \cdot p + (x_i \oplus h) \cdot p = x_i \cdot p + h \cdot p\,.$$

In fact we have

$$\begin{aligned} x_i \cdot p + (x_i \oplus h) \cdot p &= hx_i \cdot p + \overline{h}x_i \cdot p + h\overline{x}_i \cdot p \\ &= hx_i \cdot p + \overline{h}x_i \cdot p + hx_i \cdot p + h\overline{x}_i \cdot p \\ &= x_i \cdot p + h \cdot p\,. \end{aligned}$$

Therefore we reach a contradiction to the minimality w.r.t. the number of literals of the SPP form for f. Indeed a minimal form for f would be $h \cdot p + s$ instead of $(x_i \oplus h) \cdot p + s$. An analogous proof holds for the s-a-1 fault. ■

However, in practice the SPP networks are defined once a variable ordering is fixed. In this case the above theorem, which refers to SPP forms minimal with respect to any possible variable ordering, does not hold any more. Moreover, as shown below, the SPP forms minimal w.r.t. a fixed variable ordering are no longer fully testable.

Let us consider minimal SPP forms depending on a variable ordering (for more details on SPP networks, see [4, 5, 19]). For example, consider the Boolean function $f = \{0011, 0100, 1000, 1111\}$, and the variable ordering $o = x_1 < x_2 < x_3 < x_4$. The function f is indeed a pseudocube, and its minimal SPP network, w.r.t. the variable ordering o, is $(x_1 \oplus x_2 \oplus x_3)(x_1 \oplus x_2 \oplus x_4)$. Meanwhile if we choose the variable ordering $x_3 < x_1 < x_2 < x_4$, then a minimal SPP form is $(x_3 \oplus x_1 \oplus x_2)(x_3 \oplus \overline{x}_4)$, which contains less literals than the former form. In the case of 2-SPP networks, the number of literals in a minimal form is independent of the variable ordering (see [6] for more details); for this reason the testability theorem holds in any case.

If we fix an ordering, then the proof of testability given above cannot be applied anymore, as the following counter-example shows: Consider the function $f = \{00011, 00100, 00110, 01001, 01011, 01110, 10001, 10011, 10110, 11011, 11100, 11110\}$. Once the variable ordering $o = x_1 < x_2 < x_3 < x_4$ is fixed, there are eleven prime pseudoproducts for f. A minimal form for f in the variable ordering o is:

$$f = (x_1 \oplus x_2 \oplus x_3 \oplus x_4)(x_3 \oplus x_5) + x_4(x_3 \oplus x_5)\,.$$

Suppose that there is a s-a-0 at the input x_4 of the gate $(x_1 \oplus x_2 \oplus x_3 \oplus x_4)$. In this case the faulty function is:

$$f_F = (x_1 \oplus x_2 \oplus x_3)(x_3 \oplus x_5) + x_4(x_3 \oplus x_5)\,.$$

It is easy to verify that $f \equiv f_F$, but the pseudoproduct $p_F = (x_1 \oplus x_2 \oplus x_3)(x_3 \oplus x_5)$ is not represented in the order o. Therefore it is not in the set of eleven prime pseudoproducts used to form the minimal expression.

In this case the fault cannot be detected because f is indeed in minimal form w.r.t. the variable ordering o and $f \equiv f_F$. Of course, if we do not fix a variable ordering then

$$(x_1 \oplus x_2 \oplus x_3 \oplus x_4)(x_3 \oplus x_5) + x_4(x_3 \oplus x_5)$$

is not a minimal form for f.

In summary, we can formally state the following

Theorem 4 *SPP forms minimal with respect to the number of literals in a fixed variable ordering are not fully testable.*

3.2 Testability in the CFM

As we have seen in Section 2.1, the CFM is a generalization of the SAFM. In this section we investigate the testability of 2-SPP networks when cellular faults are considered. Since 2-SPP networks consist of a SOP network with an additional level of EXORs, we first review some results concerning testability of the SOP networks.

3.2.1 SOP Networks

We consider separately all gates in a SOP network, i.e. NOT, AND and OR.

NOT gates These gates are always testable:

Lemma 2 *A cellular fault at a NOT gate just before the AND layer is always testable.*

Proof. A faulty NOT gate can compute one of 3 different functions; two of them are constant functions and the testability follows from the stuck-at fault result.

We are left with the case in which the gate instead of complementing its input variable, outputs the variable itself.

Let x_i be the input variable of the faulty NOT gate, and let $f = \overline{x}_i \cdot p + s$ be a minimal SOP form for f, where p is a product and s is the rest of the SOP form. The gate that should compute $\overline{x}_i$ computes x_i and the resulting faulty function is $f_F = x_i \cdot p + s$.

Note that if two functions are equivalent, OR-ing these functions gives the same function again. If the fault is not testable, then $f \equiv f_F$ and their OR gives f. Let us compute $f + f_F$:

$$f + f_F = \overline{x}_i \cdot p + x_i \cdot p + s = p + s \not\equiv f\,,$$

since f is in minimal form. ■

AND gates AND gates are not always testable in the CFM, although they are always testable in the SAFM.

Lemma 3 *Let $f = x_1 x_2 \cdots x_k + s$ be a minimal SOP form, where s is a sum of products. The faulty AND gate computes a function $g(x_1, x_2, \ldots, x_k)$ and $g \not\equiv x_1 x_2 \cdots x_k$. We have:*

1. *If a minimal SOP form for g contains $x_1 x_2 \cdots x_k$ as a prime implicant, i.e. the faulty function is $f_F = x_1 x_2 \cdots x_k + s' + s$, then the function is testable iff s does not cover s'.*
2. *Otherwise f is testable.*

Proof.

1. ($\Rightarrow$) We show that if s covers s', then f is not testable. Observe that if s covers s', we have $s + s' = s$, therefore

$$f_F = x_1 x_2 \cdots x_k + s' + s = x_1 x_2 \cdots x_k + s = f\,.$$

   ($\Leftarrow$) We show that if s' is not covered by s, then f is testable. Note that if s does not cover s', then there exists a subcube c in the variables $x_1, \ldots, x_k$ such that s' covers c, s does not cover c, and c does not cover $x_1 x_2 \cdots x_k$ because of the minimality of the SOP form $g = x_1 x_2 \cdots x_k + s'$. Therefore there exists a point of c where f_F takes the value 1, while f is false.

2. Let $f_F = s' + s$, where s' covers $x_1 x_2 \cdots x_k$, but $x_1 x_2 \cdots x_k$ is not a prime implicant of s'. By contradiction, if $f \equiv f_F$, we have that $s' \setminus \{x_1 x_2 \cdots x_k\}$ is covered by s. Since $x_1 x_2 \cdots x_k$ is covered by s', there must exist a cube q, prime for s', such that s covers $q \setminus \{x_1 x_2 \cdots x_k\}$. But then $x_1 x_2 \cdots x_k$ cannot be in a minimal SOP form for f since it is not prime.

■

OR gate OR gates are not always testable. For an intuition of this fact, consider a minimal SOP form of a function f. Since the SOP form is minimal, the number of products that cover a single point is usually small. This implies that rarely many products are simultaneously equal to 1. In other words, the input set of the final OR gate in the network is frequently a proper subset of $\{0,1\}^k$, where k is the number of products in input to the OR. If the OR gate is faulty only on the inputs that do not occur in its input set, the gate is not testable.

For example, consider the function $f = x_1 x_2 + \overline{x}_1 \overline{x}_2$. For this function we can never have $x_1 x_2 = \overline{x}_1 \overline{x}_2 = 1$. Therefore the input 11 does not occur at the OR gate. If the gate is faulty and computes an EXOR instead of an OR, we cannot observe the fault. Indeed the behavior of the gates OR and EXOR differs only on the input 11.

We summarize the result on SOP testability in the cellular fault model in the following

Theorem 5 *SOP networks are not fully testable in the CFM.*

3.2.2 2-SPP Networks

We now study the testability of 2-SPP forms, and consider only forms minimal with respect to the number of literals.

Recall that the EXOR level of a 2-SPP network is followed by a SOP. This SOP receives as inputs all possible 0 and 1 combinations. Since we have already studied the testability of the SOP forms, it is now sufficient to consider the EXOR gates only.

EXOR gates We show below that EXOR gates are not always testable.

A faulty 2-EXOR gate can compute one out of 15 different functions, which can be classified w.r.t. testability in 6 classes (4 testable and 2 possibly non-testable).

Let $f = (x_i \oplus x_j) \cdot p + s$, where p is a 2-pseudoproduct and s is a 2-SPP form. The gate that should compute $x_i \oplus x_j$ computes another function g over x_i and x_j, and $f_F = g \cdot p + s$.

Testable classes

1. g is a constant function: The testability follows from the result for stuck-at faults.

2. $g = x_i x_j$ or $g = \overline{x}_i \overline{x}_j$: The testability follows from the fact that a redundancy would imply that $(x_i \oplus x_j) \cdot p$ is covered by s, in contradiction to the minimality of the SPP for f.
3. $g = x_i \oplus \overline{x}_j$, or $g = x_i + \overline{x}_j$, or $g = \overline{x}_i + x_j$: The testability follows from the fact that a redundancy would imply that $(x_i \oplus \overline{x}_j) \cdot p$ is covered by s, and therefore p would be an implicant of f, in contradiction to the primality of $(x_i \oplus x_j) \cdot p$.
4. $g = x_i$, or $g = x_j$, or $g = \overline{x}_i$ or $g = \overline{x}_j$: The testability follows from the fact that a redundancy would imply that s covers $x_j p$, $x_i p$, $\overline{x}_j p$, $\overline{x}_i p$, respectively, in contradiction to the minimality of the SPP for f.

Non-testable classes

5. $g = x_i \overline{x}_j$ or $g = \overline{x}_i x_j$: The fault $g = x_i \overline{x}_j$ (resp. $g = \overline{x}_i x_j$) is testable iff $\overline{x}_i x_j \cdot p$ (resp. $x_i \overline{x}_j \cdot p$) is not covered by s.
6. $g = x_i + x_j$ or $g = \overline{x}_i + \overline{x}_j$: The fault $g = x_i + x_j$ (resp. $g = \overline{x}_i + \overline{x}_j$) is testable iff $x_i x_j \cdot p$ (resp. $\overline{x}_i \overline{x}_j \cdot p$) is not covered by s.

We now give a formal proof for the above statements.

Lemma 4 *The faulty 2-EXOR gates in classes 1, 2, 3, 4 are testable, and the faulty 2-EXOR gates in classes 5 and 6 can be non-testable.*

Proof. We show the testability or non-testability of faulty gates in each of the above classes.

1. g is a constant function, then the testability follows from the result for stuck-at faults.
2. Let us consider the case $g = x_i x_j$. By contradiction suppose that $f \equiv f_F$, then

$$\begin{aligned} x_i x_j \cdot p + s &\equiv (x_i \oplus x_j) \cdot p + s \\ x_i x_j \cdot p + s &\equiv \overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p + s\,. \end{aligned}$$

   Since $x_i x_j \cdot p$ and $\overline{x}_i x_j \cdot p$ have an empty intersection, as well as $x_i x_j \cdot p$ and $x_i \overline{x}_j \cdot p$, s must cover $\overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p = (x_i \oplus x_j) \cdot p$, in contradiction to the minimality of the SPP form of f.

   An analogous proof holds for the case $g = \overline{x}_i \overline{x}_j$.
3. Let us consider the case $g = \overline{x}_i + x_j$. By contradiction suppose that $f \equiv f_F$, then

$$\begin{aligned} \overline{x}_i \cdot p + x_j \cdot p + s &\equiv (x_i \oplus x_j) \cdot p + s \\ \overline{x}_i x_j \cdot p + \overline{x}_i \overline{x}_j \cdot p + x_i x_j \cdot p + s &\equiv \overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p + s \\ \overline{x}_i \overline{x}_j \cdot p + x_i x_j \cdot p + s &\equiv x_i \overline{x}_j \cdot p + s\,. \end{aligned}$$

   Since $x_i x_j \cdot p$ and $x_i \overline{x}_j \cdot p$ are disjoint, as well as $\overline{x}_i \overline{x}_j \cdot p$ and $x_i \overline{x}_j \cdot p$, $x_i x_j \cdot p + \overline{x}_i \overline{x}_j \cdot p$ must be covered by s, and this implies that s covers $(x_i \oplus \overline{x}_j) \cdot p$. We know that a minimal SPP form for f contains $(x_i \oplus x_j) \cdot p$, therefore $p = (x_i \oplus \overline{x}_j) \cdot p + (x_i \oplus x_j) \cdot p$ is a 2-pseudoproduct of f, in contradiction to the primality of the 2-pseudoproduct $(x_i \oplus x_j) \cdot p$.

   Analogous proofs hold for the cases $g = x_i + \overline{x}_j$ and $g = x_i \oplus \overline{x}_j$.
4. Let us consider the case $g = x_i$. By contradiction suppose that $f \equiv f_F$, then

$$\begin{aligned} x_i \cdot p + s &\equiv (x_i \oplus x_j) \cdot p + s \\ x_i x_j \cdot p + x_i \overline{x}_j \cdot p + s &\equiv \overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p + s \\ x_i x_j \cdot p + s &\equiv \overline{x}_i x_j \cdot p + s\,. \end{aligned}$$

Since $x_i x_j \cdot p$ and $\overline{x}_i x_j \cdot p$ are disjoint, s must cover the 2-pseudoproducts $x_i x_j \cdot p$ and $\overline{x}_i x_j \cdot p$, and this implies that s covers $x_j \cdot p$. Therefore f contains $(x_i \oplus x_j) \cdot p$ and $x_j \cdot p$. We now show that $x_j \cdot p + (x_i \oplus x_j) \cdot p = x_i \cdot p + x_j \cdot p$. In fact we have

$$x_j \cdot p + (x_i \oplus x_j) \cdot p = x_i x_j \cdot p + \overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p = x_j \cdot p + x_i \cdot p\,.$$

Therefore we reach a contradiction to the minimality (w.r.t. the number of literals) of the 2-SPP form for f.

Analogous proofs hold for all the other cases.

5. Let us consider the case $g = x_i \overline{x}_j$. The function f is non-testable if and only if $f \equiv f_F$. Therefore we have that $x_i \overline{x}_j \cdot p + s \equiv x_i \overline{x}_j \cdot p + \overline{x}_i x_j \cdot p + s$, which is equivalent to $s \equiv \overline{x}_i x_j \cdot p + s$. This finally means that $\overline{x}_i x_j \cdot p$ is covered by s.

   An analogous proof holds for the case $g = \overline{x}_i x_j$.

6. Let us consider the case $g = x_i + x_j$. The function f is non-testable if and only if $f \equiv f_F$. Therefore we have that $x_i \cdot p + x_j \cdot p + s \equiv x_i \overline{x}_j \cdot p + \overline{x}_i x_j \cdot p + s$, which is equivalent to $x_i \overline{x}_j + x_i x_j \cdot p + \overline{x}_i x_j \cdot p + s \equiv \overline{x}_i x_j \cdot p + x_i \overline{x}_j \cdot p + s$ and then to $x_i x_j \cdot p + s \equiv s$. This finally means that $x_i x_j \cdot p$ is covered by s.

   An analogous proof holds for the case $g = \overline{x}_i + \overline{x}_j$.

■

We summarize the results on 2-SPP testability in the cellular fault model in the following

Theorem 6 *2-SPP networks are not fully testable in the CFM.*

4 Improving the Testability of 2-SPP and SPP Networks

We have seen in the previous sections that 2-SPP networks minimal with respect to the number of 2-pseudoproducts, and in general SPP networks, are not fully testable in the SAFM. Moreover, 2-SPP and SPP networks, minimal with respect to the number of literals, are not testable in the CFM. We now exploit these results in order to understand the reasons why certain networks may contain redundancies, and to consequently improve their testability. As we will see, it is possible to identify some structural properties of the covers that guarantee their full testability in the SAFM, and the testability of the EXOR gates in the CFM. In summary, in this section we will show how to transform non-testable-EXOR gates into testable gates.

4.1 SAF Model

4.1.1 2-SPP Networks Minimal w.r.t. the Number of 2-Pseudoproducts

We show how to transform a minimal 2-SPP network into an equivalent fully testable network, still minimal with respect to the number of 2-pseudoproducts.

Let f be a Boolean function and C be a 2-SPP covering for f, minimal with respect to the number of 2-pseudoproducts. Our main idea is that of changing C in such a way that the deletion of any literal in an EXOR factor (operation equivalent to a s-a-0 or a s-a-1 at inputs to an EXOR gate) changes the function.

For any 2-pseudoproduct $(x_i \oplus x_j)p$, where x_i and x_j are literals, and p is a 2-pseudoproduct, consider the function g representing the cover $C \setminus \{(x_i \oplus x_j)p\}$. Now suppose that the function g covers $x_i p$. Then we can substitute $(x_i \oplus x_j)p$ with $x_j p$ in C, without changing the covered function f. In fact, as we have already observed in the proof of Theorem 2, $x_i p + (x_i \oplus x_j)p = x_i p + x_j p$.

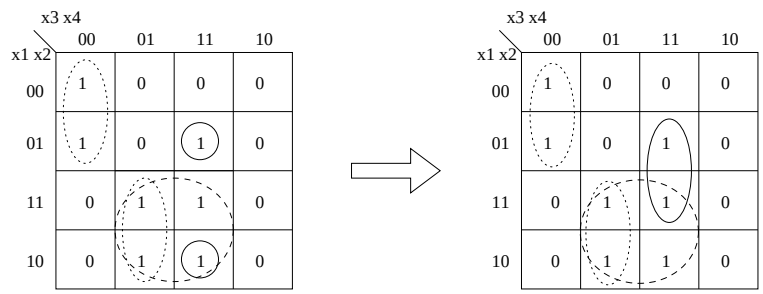

Figure 2: On the left: the Karnaugh map of a function f with a 2-SPP cover $(x_1 \oplus \overline{x}_4)\overline{x}_3 + x_1x_4 + (x_1 \oplus x_2)x_3x_4$, minimal with respect to the number of 2-pseudoproducts, but non-fully testable. On the right: the Karnaugh map of the same function with the corresponding 2-SPP cover $(x_1 \oplus \overline{x}_4)\overline{x}_3 + x_1x_4 + x_2x_3x_4$, still minimal with respect to the number of 2-pseudoproducts, but fully testable.

Suppose that we have removed all the non-testable EXORs from x_ip. The new 2-pseudoproduct x_ip could not be fully testable since it could be not prime (primality is essential to prove the testability of AND gates.) We can replace x_ip with a 2-pseudoproduct q obtained from x_ip removing the maximal number of factors keeping q in f. It is easy to verify that the EXOR and AND gates of q are now fully testable.

Otherwise, if the function g does not cover x_ip, we can check whether it covers one of the following 2-pseudoproducts: x_jp, $\overline{x}_ip$, or $\overline{x}_jp$, and we can modify the cover C replacing $(x_i \oplus x_j)p$ with (the maximal 2-pseudoproduct obtained deleting factors from) x_ip, or $\overline{x}_jp$, or $\overline{x}_ip$, respectively, without changing f, as it can be easily verified.

Observe that the overall number of 2-pseudoproducts in C do not change after such operation. Indeed we replace, if necessary, a 2-pseudoproduct with just another one.

Repeating such operation for any 2-pseudoproduct, we can transform the cover C into an equivalent cover C', which contains the same number of 2-pseudoproducts and satisfies the following property:

$$\forall\ (x_i \oplus x_j)p \in C' \quad \Rightarrow \quad x_ip \not\subseteq g \quad \textit{and} \quad x_jp \not\subseteq g \quad \textit{and} \quad \overline{x}_ip \not\subseteq g \quad \textit{and} \quad \overline{x}_jp \not\subseteq g\,, \tag{1}$$

where x_i and x_j are literals, p is a 2-pseudoproduct, and g is the function representing the cover $C' \setminus \{(x_i \oplus x_j)p\}$.

Note that a 2-SPP cover minimal with respect to the number of literals always satisfies Property 1. Indeed, the new cover C' has a smaller number of literals than C.

Example 4 *Consider the function f represented in the Karnaugh map in Figure 2 and the cover C represented on the left side: $C = (x_1 \oplus \overline{x}_4)\overline{x}_3 + x_1x_4 + (x_1 \oplus x_2)x_3x_4$. First, let us consider the 2-pseudoproduct $(x_1 \oplus \overline{x}_4)\overline{x}_3$. According to the previous observations, it is fully testable since it satisfies Property 1.*

Consider now the 2-pseudoproduct $(x_1 \oplus x_2)x_3x_4$, we can observe that it is not fully-testable. In fact, $x_1x_3x_4$ is covered by $C \setminus \{(x_1 \oplus x_2)x_3x_4\} = (x_1 \oplus \overline{x}_4)\overline{x}_3 + x_1x_4$, in particular by the product x_1x_4, thus Property 1 is not satisfied. In other words a s-a-0 of x_2 in $(x_1 \oplus x_2)x_3x_4$ would produce a non-testable fault.

We can now replace the 2-pseudocube $(x_1 \oplus x_2)x_3x_4$ with $x_2x_3x_4$ (corresponding to the points $0111, 1111$), as represented in the cover on the right side of the figure. It is easy to verify that $x_2x_3x_4$ cannot be further reduced into x_3x_4, or x_2x_4, or x_2x_3. Therefore the new cover $(x_1 \oplus \overline{x}_4)\overline{x}_3 + x_1x_4 + x_2x_3x_4$ is minimal with respect to the number of 2-pseudoproducts, and it is fully testable.

We finally prove that the new cover is fully testable.

Lemma 5 *Let f be a Boolean function, and C a 2-SPP cover for f satisfying Property 1. Then C is fully testable.*

Proof. We prove by contradiction that any fault in an EXOR gate of C can be tested. Let $(x_i \oplus x_j)p$ be a 2-pseudoproduct in C and s be the rest of the cover in 2-SPP form, i.e., $C = (x_i \oplus x_j)p + s$. We must consider four cases: a s-a-0 in x_i, a s-a-1 in x_i, a s-a-0 in x_j, and a s-a-1 in x_j.

Let us consider the case $x_i \equiv 0$, i.e., s-a-0 in x_i. Then the network computes the faulty function $f_F = x_j \cdot p + s$. By contradiction suppose that $f_F \equiv f$, then

$$\begin{aligned} x_j p + s &\equiv (x_i \oplus x_j)p + s \\ x_i x_j p + \overline{x}_i x_j p + s &\equiv x_i \overline{x}_j p + \overline{x}_i x_j p + s \\ x_i x_j p + s &\equiv x_i \overline{x}_j p + s\,. \end{aligned}$$

Since the two 2-pseudoproducts $x_i x_j p$ and $x_i \overline{x}_j p$ have an empty intersection, the last equality implies that s must cover both of them, which in turn implies that s covers $x_i p$, in contradiction with the fact that C satisfies Property 1.

Let us consider the case $x_i \equiv 1$, i.e., s-a-1 in x_i. Then the network computes the faulty function $f_F = \overline{x}_j p + s$. By contradiction suppose that $f_F \equiv f$, then

$$\begin{aligned} \overline{x}_j p + s &\equiv (x_i \oplus x_j)p + s \\ x_i \overline{x}_j p + \overline{x}_i \overline{x}_j p + s &\equiv x_i \overline{x}_j p + \overline{x}_i x_j p + s \\ \overline{x}_i \overline{x}_j p + s &\equiv \overline{x}_i x_j p + s\,. \end{aligned}$$

Since the two 2-pseudoproducts $\overline{x}_i \overline{x}_j p$ and $\overline{x}_i x_j p$ have an empty intersection, the last equality implies that s must cover both of them, which in turn implies that s covers $\overline{x}_i p$, in contradiction with the fact that C satisfies Property 1.

The remaining two cases are analogous. ■

In summary, we can formally state the following

Theorem 7 *2-SPP forms minimal with respect to the number of 2-pseudoproducts can always be transformed into fully testable 2-SPP forms, still minimal with respect to the number of 2-pseudoproducts.*

4.1.2 SPP Networks

The technique described in the previous paragraph can be generalized in order to make general SPP networks fully testable. The idea is still that of making the network sensitive to the deletion of literals in its EXOR factors.

Let f be a Boolean function, C be an SPP covering for f, minimal with respect to the number of pseudoproducts, and $(x_i \oplus h)p$ be a pseudoproduct in C, where x_i is a literal, h an EXOR factor not including x_i, and p a pseudoproduct. Consider the function g representing the cover $C \setminus \{(x_i \oplus h)p\}$. If g covers $x_i p$ (or $\overline{x}_i p$), we replace $(x_i \oplus h)p$ with (the maximal 2-pseudoproduct obtained deleting factors from) hp (or $\overline{h}p$) in C, without changing the covered function, nor the number of pseudoproducts in the cover.

Repeating the same operation for any other literals in h, for any other EXOR factor in p, and finally for any other pseudoproduct in the cover, we can transform C into an equivalent cover C', which contains the same number of pseudoproducts and satisfies the following property:

$$\forall\ ep \in C',\ \ \forall\ x_i \in e \ \ \Rightarrow \ \ x_i p \not\subseteq g \quad \textit{and} \quad \overline{x}_i p \not\subseteq g\,, \tag{2}$$

where x_i is a literal, e is an EXOR factor, and g is the function representing the cover $C' \setminus \{ep\}$.

Generalizing the proof of Lemma 5 to this context, it is easy to verify that an SPP cover satisfying Property 2 is fully testable.

In summary, we can formally state the following

Theorem 8 *SPP forms minimal with respect to the number of pseudoproducts can always be transformed into fully testable SPP forms, still minimal with respect to the number of pseudoproducts.*

Note that while the minimality is hard to check, the properties guaranteeing the testability of the network (Properties 1 and 2) can be easily verified.

As a final remark, we can observe that Property 2 can also be exploited in order to make testable SPP expressions minimal with respect to the number of literals in a fixed variable ordering. Indeed, given such an SPP expression, we can fix all possible redundancies by changing its pseudoproducts in order to derive a new expression satisfying Property 2. Observe that the overall size of the new expression, measured by the number of literals, decreases; anyway this new expression is not represented in the fixed ordering anymore. This is not a problem since we fix the order of the variables only for decreasing the computational time of our algorithms.

4.2 CF Model

2-SPP networks minimal with respect to the number of literals, and more in general SPP networks, are not fully testable in the CFM. This is a consequence of the non-testability of SOP networks in this model, and of the non-testability of the EXOR gates.

Following the approach described for the SAFM, we describe here how the faulty EXOR gates in a 2-SPP network can be made testable.

Let $f = (x_i \oplus x_j) \cdot p + s$, where p is a 2-pseudoproduct and s is a 2-SPP form. As we have seen in Section 3.2.2, if the EXOR gate is faulty, and computes $x_i\overline{x}_j$, $\overline{x}_i x_j$, $x_i + x_j$, or $\overline{x}_i + \overline{x}_j$, then the fault may not be testable. More precisely, the fault is not testable if and only if $\overline{x}_i x_j \cdot p$, $x_i\overline{x}_j \cdot p$, $x_i x_j \cdot p$, $\overline{x}_i\overline{x}_j \cdot p$, respectively, are covered by s.

Thus the idea is that of changing the cover C of a function is such a way that the new cover, C', satisfies the following property:

$$\forall\ (x_i \oplus x_j)p \in C' \quad \Rightarrow \quad x_i\overline{x}_j p,\ \overline{x}_i x_j p,\ x_i x_j p,\ \overline{x}_i\overline{x}_j p \not\subseteq g\,, \tag{3}$$

where x_i and x_j are literals, p is a 2-pseudoproduct, and g is the function representing the cover $C' \setminus \{(x_i \oplus x_j)p\}$.

Note that Property 3 implies Property 1. It is not difficult to verify that the 2-SPP cover C can be transformed into a 2-SPP cover C' satisfying Property 3 in the following way:

- if g covers $x_i\overline{x}_j p$, we replace $(x_i \oplus x_j)p$ with (the maximal 2-pseudoproduct obtained deleting factors from) $\overline{x}_i x_j p$;
- if g covers $\overline{x}_i x_j p$, we replace $(x_i \oplus x_j)p$ with (the maximal 2-pseudoproduct obtained deleting factors from) $x_i\overline{x}_j p$;
- if g covers $x_i x_j p$, we replace $(x_i \oplus x_j)p$ with (the maximal 2-pseudoproducts obtained deleting factors from) $x_i p$ and $x_j p$;
- if g covers $\overline{x}_i\overline{x}_j p$, we replace $(x_i \oplus x_j)p$ with (the maximal 2-pseudoproducts obtained deleting factors from) $\overline{x}_i p$ and $\overline{x}_j p$.

Observe that in the first two cases the number of 2-pseudoproducts does not change, and the cost in literals decreases if the 2-pseudoproducts that replace $(x_i \oplus x_j)p$ are reduced in order to maintain the AND-testability; while in the last two cases, the number of 2-pseudoproducts increases.

Table 1: *Costs for benchmark functions in 2-SPP, SOP and SPP forms and minimization times*

	2-SPP			**SOP**			**SPP**			
name	μ	#E	time	μ'	μ/μ'	time	μ''	#E	μ/μ''	time
9sym	168	18	242.67	588	0.29	5.32	188	30	0.89	147.58
addm4	694	34	50.96	1407	0.49	0.87	*	*	*	*
adr4	105	5	6.69	415	0.25	0.10	118	10	0.89	88.22
clip	402	26	1662.27	769	0.52	0.38	*	*	*	*
dist	471	26	924.10	879	0.54	0.14	636	50	0.74	8196.00
f51m	232	19	64.00	402	0.58	0.23	243	23	0.95	443.00
life	180	16	120.40	756	0.24	0.03	180	16	1.00	262.00
m4	735	28	890.94	1214	0.61	0.67	835	48	0.88	9929.40
max512	620	35	341.24	1032	0.60	0.53	*	*	*	*
mlp4	500	25	339.51	869	0.58	1.62	524	32	0.95	1423.74
newcond	161	11	1485.01	239	0.67	0.01	*	*	*	*
radd	105	5	15.20	415	0.25	0.08	118	10	0.89	144.00
rd53	64	6	0.10	175	0.37	0.01	66	7	0.97	0.20
rd73	212	11	24.10	903	0.23	0.03	187	15	1.13	114.00
root	281	21	272.32	376	0.75	0.08	366	31	0.77	1597.70
squar5	101	6	0.42	120	0.84	0.01	112	8	0.90	0.64
xor5	24	2	0.05	96	0.25	0.01	18	1	1.33	0.02
z4	91	6	5.30	311	0.29	0.04	100	10	0.91	6.75

5 Experimental Results

In this section experimental results for the SAFM are reported. The methods described above have been implemented in C. The experiments have been run on a Pentium III 450MHz CPU with 128 MByte of main memory. The three-level forms have been optimized using the tools described in [6] and the generated networks have been written as BLIF files. The correctness of the synthesis process and the testability analysis have been studied using SIS [23]. The benchmarks are taken from LGSynth93 [27].

To give an impression on the cost savings in comparison to SOPs in a first series of experiments the quality of SPP forms (optimized by different criteria) is compared to two-level approaches. We count the number of literals and gates (AND and EXOR) of an expression. In the multi-level context the cost function is the total number of literals in all gates (see [13, 16]). The problem is that in many technologies EXOR and OR (or AND) gates have different costs. In [16] the authors consider a 2-input EXOR gate as $x \oplus y = \overline{x} \cdot y + x \cdot \overline{y}$. Thus the cost in literals of a 2-input EXOR gate is 4, while the cost of the 2-input OR and AND gates is 2. This is also proportional to the number of transistors used in the CMOS technology mapping (i.e., 4 transistors for AND/OR gates and 8 transistors for the EXOR gate). More in general, 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. Therefore, we can use a function μ where a k-input EXOR gate costs $4(k-1)$, and k-input OR/AND gates cost k. This cost function corresponds to the CMOS cost described in [13].

The costs and the run times for the minimization algorithms are reported in Table 1. We compare the costs of minimal 2-SPP, SOP and SPP forms (2-SPP and SPP networks are minimized with respect to the number of literals in the expressions). In the first column the *name* of the benchmark is given. In the next column the costs are given for 2-SPP, SOP and SPP forms. Here, μ is the cost for the 2-SPP

Table 2: *Number of redundancies*

name	**original**	**2-SPP**	**SOP**	**SPP**
9sym	0	0	0	0
addm4	24	0	0	*
adr4	24	0	0	0
clip	0	0	0	*
dist	0	0	0	0
f51m	56	0	0	0
life	0	0	0	0
m4	22	0	0	3
max512	4	0	0	*
mlp4	24	0	0	2
newcond	0	0	0	*
radd	0	0	0	0
rd53	0	0	0	0
rd73	0	0	0	0
root	0	0	0	1
squar5	12	0	0	1
xor5	0	0	0	0
z4	12	0	0	0

network, while μ' is the cost for the SOP network. The cost for the SPP network is μ''. #E is the number of different EXOR gates in 2-SPP and SPP forms. The run times are given in column *time* The star * indicates that the SPP algorithm did not terminate after 172800 seconds (corresponding to 2 CPU days). The minimization algorithms are designed for exact synthesis of 2-SPP and SPP forms. Indeed the set of prime 2-pseudoproducts (pseudoproducts) is exactly computed. Since we used some heuristics [14, 24] in solving the set-covering problem, the numbers of literals in the expressions in Table 1 are upper bounds for the minimal solutions. We note that 2-SPP and SPP forms are much more compact than the corresponding SOP expressions, 2-SPP minimization is also faster than SPP minimization with the exceptions of $9sym$ and $xor5$. This is due to the fact that the SPP minimization algorithm takes advantage of some regularities of functions (see [2]), which cannot be exploited by the 2-SPP synthesis.

For all forms, the number of redundancies under the SAFM is given in Table 2. If SOPs are minimized, i.e. they are prime and irredundant, the corresponding networks are also fully testable. But compared to 2-SPP forms they are significantly larger in size (see above). As predicted by the theoretical results in Section 3, we can observe that 2-SPPs are fully testable in the SAFM (see Theorem 2), while SPPs may contain redundancies. Indeed the redundancies in SPP networks are due to the heuristic used for their synthesis, and to the fact that the variable ordering in the minimization algorithms is fixed (see Theorem 4). But also in this case, the number of redundancies is low.

In summary, the experiments show that 2-SPP forms provide a very good compromise between compact representation, complexity of the minimization process and testability. Beside being more efficient than SOP regarding number of literals, they are so far the only three-level form that ensures full testability of the resulting circuit by construction.

6 Conclusion

In this paper we studied for the first time the testability of the resulting networks for two static fault models, i.e. the stuck-at fault model and the cellular fault model. For specific classes, i.e. 2-SPPs and SPPs minimal w.r.t. the number of literals in any variable ordering, full testability has been proved for the SAFM, while for other classes counter-examples were provided. Networks that are not fully testable have been studied in order to improve their testability. In particular, 2-SPP networks minimal with respect to the number of 2-pseudoproducts can be transformed into minimal 2-SPP forms that are fully-testable.

References

[1] M. Abramovici and M. Breuer. *Digital Systems Testing and Testable Design*. IEEE, 1994.

[2] A. Bernasconi, V. Ciriani, F. Luccio, and L. Pagli. Three-Level Logic Minimization Based on Function Regularities. *IEEE Transactions on CAD*, 22(8):1005–1016, 2003.

[3] M. Breuer and A. Friedman. *Diagnosis & reliable design of digital systems*. Computer Science Press, 1976.

[4] V. Ciriani. Synthesis of SPP Three-Level Logic Networks using Affine Spaces. *IEEE Transactions on CAD*, 22(10):1310–1323, 2003.

[5] V. Ciriani. *Three-Level Logic Synthesis: Algebraic Approach and Minimization Algorithms*. PhD thesis, Dipartimento di Informatica, University of Pisa, 2003.

[6] V. Ciriani and A. Bernasconi. 2-SPP: a Practical Trade-Off between SP and SPP Synthesis. In *5th International Workshop on Boolean Problems (IWSBP2002)*, pages 133–140, 2002.

[7] V. Ciriani, A. Bernasconi, and R. Drechsler. Testability of SPP Three-Level Logic Networks. In *IFIP 12-th International Conference on Very Large Scale Integration, (VLSI-SOC)*, pages 331–336, 2003.

[8] V. Ciriani, F. Luccio, and L. Pagli. Synthesis of Integer Multipliers in Sum of Pseudoproducts Form. *Integration - the VLSI journal*, 36(3):103–118.

[9] O. Coudert. Two-Level Logic Minimization: an overview. *INTEGRATION*, 17:97–140, 1994.

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

[11] D. Debnath and Z. Vransic. A Fast Algorithm for OR-AND-OR Synthesis. *IEEE Transactions on Computer Aided Design*, 22(9):1166–1176, 2003.

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

[13] M. Eggerstedt, N. Hendrich, and K. von der Heide. 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*, pages 142–146, 1993.

[14] M. S. Fiorenzo-Catalano and F. Malucelli. Parallel Randomized Heuristics For The Set Covering Problem. *International Journal of Computer Research*, 10(4), 2001.

[15] A. Friedman. Easily testable iterative systems. *IEEE Transactions on Computers*, C-22:1061–1064, 19733.

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

[17] R. Ishikawa, T. Hirayama, G. Koda, and K. Shimizu. New Three-Level Boolean Expression Based on EXOR Gates, journal = IEICE Transactions on Information and Systems, pages = 1214-1222, year = 2004, volume = e87-d, number = 5.

[18] R. Ishikawa, T. Igarashi, T. Hirayama, and K. Shimizu. Pseudocube-based expressions to enhance testability. In *IEEE Asia-Pacific Conference on Circuits and Systems*, volume 2, pages 305–310, 2002.

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

[20] J. Rose, A. El Gamal, and A. Sangiovanni-Vincentelli. Architecture of Field-Programmable Gate Arrays. *Proceedings of the IEEE*, 81(7):1013–1029, 1993.

[21] T. Sasao. On the Complexity of Three-Level Logic Circuits. In *Int. Workshop on Logic Synthesis*, 1989.

[22] T. Sasao. AND-EXOR Expressions and their Optimization. In T. Sasao, editor, *Logic Synthesis and Optimization*. Kluwer Academic Publisher, 1993.

[23] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[24] J. Tebboth and R. Daniel. A Tightly Integrated Modelling and Optimisation Library. *Annals of Operations Research*, 104:313–333, 2001.

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

[26] T. Williams and K. Parker. Design for Testability - A Survey. *IEEE Transactions on Computers*, 31(1):2–15, 1982.

[27] S. Yang. Synthesis on Optimization Benchmarks. User guide, Microelectronic Center, 1991. Benchmarks available at ftp://ftp.sunsite.org.uk/computing/general/espresso.tar.Z.