Efficient Minimization of Fully Testable 2-SPP Networks

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

Valentina Ciriani
Department of Information Technologies
University of Milano, Italy
ciriani@dti.unimi.it

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

Tiziano Villa
DIEGM
University of Udine, Italy
villa@uniud.it

Abstract

The paper presents a heuristic algorithm for the minimization of 2-SPP networks, i.e., three-level EXOR-AND-OR forms with EXOR gates restricted to fan-in 2. Previous works had presented exact algorithms for the minimization of unrestricted SPP networks and of 2-SPP networks. The exact minimization procedures were formulated as covering problems as in the minimization of SOP forms and had worst-case exponential complexity. Extending the expand-irredundant-reduce paradigm of the ESPRESSO *heuristic, we propose a minimization algorithm for 2-SPP networks that iterates local minimization and reshape of a solution until further improvement. We introduce also the notion of EXOR-irredundant to prove that OR-AND-EXOR irredundant networks are fully testable and guarantee that our algorithm yields OR-AND-EXOR irredundant solutions. We report a large set of experiments showing impressive high-quality results with affordable run times, handling also examples whose exact solutions could not be computed.*

1. Introduction

Mainstream logic synthesis concentrates on two extremes: two-level logic and unrestricted multi-level logic. The former has been studied in great depth both from the theoretical and practical viewpoint, resulting in exact and heuristic automatic minimizers of industrial quality, such as ESPRESSO [2]. For the latter we do not have yet a complete exact characterization, but a robust theory of algebraic and Boolean division triggered the development of efficient heuristic tools, such as SIS [13].

In-between there are interesting restricted forms of multi-level logic, of which three-level logic attracted the attention of many researchers, as surveyed in [4]. Here we continue the investigation of three-level EXOR-AND-OR forms, introduced in [9, 4]. They are a direct generalization of AND-OR forms, obtained generalizing cubes to pseudocubes where literals in cubes may be replaced by EXOR factors in pseudocubes. Pseudocubes have been shown in [4] to correspond to affine spaces over the Boolean vector space B^n, $B = \{0, 1\}$. The repeated union of pseudocubes yields prime pseudocubes, an extension of primes for SOP; once prime pseudocubes are computed, exact minimization of EXOR-AND-OR forms is reduced to the solution of a covering table, as in case of SOP forms. To be technologically feasible, EXOR-AND-OR forms are restricted to EXOR factors with at most k literals. In this paper we will discuss only forms with $k = 2$, called 2-SPP forms [5].

Although exact methods for SPP minimization perform well on many examples [4, 5], they are not affordable for all industrial benchmarks, therefore we must give up exact minimization for heuristic one, mirroring what has been done for SOP minimization [11, 2]. In SOP heuristic minimization the solution of the covering table is replaced by the iteration of a sequence of EXPAND, IRREDUNDANT COVER, REDUCE operations: EXPAND replaces each implicant in the cover with a largest prime containing it and eventually other cubes of the cover (or parts of them); IRREDUNDANT COVER removes a maximal set of redundant implicants; REDUCE replaces each prime implicant by a smallest implicant that covers all the relatively essential vertices of the prime implicant (the vertices not contained in any other cube of the given representation). These operations are performed heuristically, i.e., the order of expanding and reducing implicants matters with respect to the final quality. The REDUCE operation is an uphill move which adds literals and enables the optimization process

to climb out of a local minimum and move closer to the global minimum during the next EXPAND and IRREDUNDANT steps. Both the EXPAND and IRREDUNDANT operations remove literals or cubes. Iterating the ESPRESSO loop on a given representation yields a final representation that satisfies primality and irredundancy and whose cost is at a local minimum. ESPRESSO guarantees also that the final cover is fully testable. This is easy for single-output functions for which a prime and irredundant cover is fully testable for all single stuck-at faults. For multi-output functions the result is obtained by applying an iterative procedure, MAKE_SPARSE, that makes each output function prime and irredundant separately.

In this paper we present a heuristic minimization procedure for 2-SPP forms based on the iteration of a suite of operations that generalize the expansion-irredundant-reduction cycle of heuristic SOP minimization. In particular we introduce the operations MERGE, EXOR-EXPAND that are specific to the 2-SPP forms and then we describe the iterative loop to improve the solution. The proposed procedure has been implemented with good results on industrial benchmarks, enabling us to minimize 2-SPP forms for which we cannot afford to compute an exact solution.

Beside synthesis, testability is a major aspect of the design process. In this paper the testability of the 2-SPP forms derived from our heuristics is studied from a theoretical point of view under the Stuck-At Fault Model (SAFM). In [6] is shown that a 2-SPP network *minimal* with respect to the number of literals is fully testable under the SAFM. The proof of full testability presented in [6] exploits the properties of a minimal network (i.e., primality, irredundancy and minimality w.r.t. the number of literals), where *minimal* means that it is a 2-SPP network representing the function with the *minimum* number of literals (and there may be more than one with such minimum number of literals). In this paper we prove that primality and minimality are not necessary for guaranteeing full testability. Indeed weaker properties are sufficient for obtaining fully testable 2-SPP networks. Therefore we introduce the notion of AND-irredundancy and EXOR-irredundancy to prove that our heuristic algorithm yields fully testable solutions although we cannot guarantee their minimality.

2. Preliminaries

Stuck-at Fault Model (SAFM) Let C be any combinational logic circuit over a fixed library. 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-0 (s-a-0) and stuck-at-1 (s-a-1) faults.

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 *coNP-complete*. 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 *fully testable*.

2-SPP Networks In this paragraph we recall some basic definition from [4, 5]. In a Boolean space $\{0,1\}^n$ described by n variables $x_1, x_2, \ldots, x_n$, 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 $\overline{x}_1 \oplus \overline{x}_2 = x_1 \oplus x_2$).

A 2-*pseudoproduct* is a product of 2-EXOR factors; and a 2-*SPP form* is a sum of 2-pseudoproducts. 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 more details see [9, 4]). 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 $(x_2 \oplus \overline{x}_3)(x_2 \oplus \overline{x}_4)$.

A set of points whose characteristic function can be represented as a 2-pseudoproduct is a 2-*pseudocube*. This is a generalization of 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 the minimum number of literals or with the minimum number of 2-pseudoproducts.

Remark 1 *In [7] the authors consider a 2-input EXOR gate as* $x \oplus y = x\overline{y} + \overline{x}y$*. Thus the cost in literals of a 2-input EXOR gate is 4, when it is introduced for the first time in the network, while the cost of new 2-input AND and OR gates is 2. This is also proportional to the number of transistors used for the CMOS technology mapping. An EXOR, AND or OR gate that is already used in the network has no cost. Each factor of each product costs 1, and each product of the cover costs 1.*

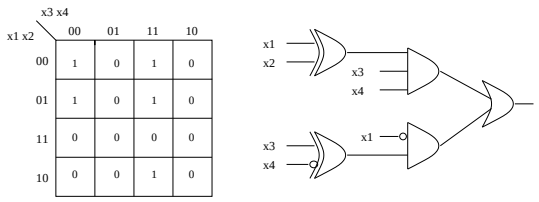

Figure 1. Karnaugh map of function f with a 2-SPP cover $(x_1 \oplus x_2)x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$, and the corresponding 2-SPP circuit representation.

For example consider 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 $\overline{x}_2x_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_1x_3x_4$. In [5] a 2-SPP minimization algorithm is proposed. A minimal 2-SPP form is generated by choosing a minimal subset of prime 2-pseudoproducts that covers the original function.

3. 2-SPP Fully Testable Networks

A 2-SPP network minimal with respect to the number of literals is fully testable under the SAFM [6]. The proof of full testability presented in [6] exploits three properties of the network: *primality*, *irredundancy* and *minimality w.r.t. the number of literals*. In this section we prove that primality and minimality are not necessary for guaranteeing full testability. Indeed weaker properties are sufficient for obtaining fully testable 2-SPP networks. Since minimality is no longer necessary, we can then design testing-oriented heuristics. Let f be a Boolean function. A 2-pseudoproduct p in f is *AND-irredundant* if deleting any factor from it, the resulting 2-pseudoproduct is no longer contained in f. A 2-SPP cover for f is *AND-irredundant* iff it is composed by AND-irredundant 2-pseudoproducts.

Note that when a 2-pseudoproduct is indeed a product, this definition coincides with the notion of primality for products. As already pointed out in Section 2, the primality of 2-pseudoproducts is a stronger property than AND-irredundancy. As for SOP forms, a 2-SPP form is *irredundant* if deleting any 2-pseudocube from the expression, the function changes. For SOP forms primality and irredundancy are sufficient for proving the full testability of the expressions under the SAFM. We will show that the full testability of 2-SPP forms is guaranteed by AND-irredundancy, irredundancy and the following additional property.

Definition 1 *Let f be a Boolean function and C be a 2-SPP covering for f. The cover C is* EXOR-irredundant *if* $\forall\ (x_i \oplus x_j)p \in C$ *we have*

$$x_ip \not\subseteq g \quad \text{and} \quad x_jp \not\subseteq g \quad \text{and} \quad \overline{x}_ip \not\subseteq g \quad \text{and} \quad \overline{x}_jp \not\subseteq g$$

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

We can observe that the AND-irredundancy guarantees that the deletion of a factor in any 2-pseudoproduct changes the function, as well as irredundancy guarantees that the deletion of any 2-pseudoproduct changes the function. In an analogous way, the EXOR-irredundancy guarantees that the deletion of any literal in an EXOR factor changes the function: e.g., suppose on the contrary that $x_ip = x_ix_jp + x_i\overline{x}_jp \subseteq g$ and $(x_i \oplus x_j)p \in C$, then $(x_i \oplus x_j)p = \overline{x}_ix_jp + x_i\overline{x}_jp$ can be replaced by $x_jp = x_ix_jp + \overline{x}_ix_jp$, i.e., literal x_i can deleted without changing the function.

An irredundant, AND-irredundant and EXOR irredundant 2-SPP form is called *OR-AND-EXOR-irredundant*. Finally we have the following results (see [1] for the proofs):

Theorem 1 *OR-AND-EXOR-irredundant 2-SPP forms are fully testable.*

Theorem 2 *2-SPP forms minimal w.r.t. the number of literals are OR-AND-EXOR irredundant.*

4. 2-SPP Heuristics

The major problem with 2-SPP forms is the huge minimization time required for their exact synthesis (see Section 5). To overcome this problem, in this section we describe a heuristic algorithm for the synthesis of OR-AND-EXOR-irredundant 2-SPP forms. In this way we sacrifice the minimality of the forms to obtain reduced synthesis time, but experiments show that the overhead is very small, and theoretical results show that we still obtain fully testable networks. The basic operations used by our minimization algorithm are direct generalizations of classical two-level heuristic minimization (see [2, 8, 12]). Our basic operations are listed below. While some are straightforward generalizations from previous approaches, the new operators are described in detail in the following sections.

MERGE replaces two adjacent 2-pseudoproducts, contained in the cover, by their union.

EXPAND tries to remove each literal x_i of a 2-pseudoproduct x_ip in order to obtain a smaller cover of the function.

EXOR-EXPAND tries to remove each EXOR factor $(x_i \oplus x_j)$ of a 2-pseudoproduct $(x_i \oplus x_j)p$ in order to obtain a smaller cover of the function. Otherwise it tries to replace $(x_i \oplus x_j)p$ with x_ip, x_jp,

$\overline{x}_i p$, or $\overline{x}_j p$. Observe that the cover obtained with EXPAND and EXOR-EXPAND is EXOR-irredundant and AND-irredundant but not necessarily prime.

IRREDUNDANT deletes redundant 2-pseudoproducts from a given cover. This operation guarantees the irredundancy of the cover.

REDUCE takes a 2-pseudoproduct p and reduces the set it represents by adding some literal to p.

It is easy to see that the EXPAND and EXOR-EXPAND operations guarantee the AND-irredundancy of the obtained expression, while IRREDUNDANT guarantees the irredundancy. We will later explicitly show that the EXOR-EXPAND operation also guarantees the EXOR-irredundancy.

4.1. MERGE

MERGE intuitively replaces two adjacent 2-pseudoproducts of the same cover by their union. To implement MERGE we need the notion of *adjacency* of 2-pseudoproducts. We first recall some definitions.

The *structure* of a 2-pseudoproduct is the 2-pseudoproduct without complementation. Given a 2-pseudoproduct p we call *literal part* of p the product of single literals in it, while the *EXOR part* is the remaining product of 2-EXORs. For example the structure of the 2-pseudoproduct $x_1\overline{x}_2x_6(x_3 \oplus x_4)(x_3 \oplus \overline{x}_5)(x_7 \oplus \overline{x}_8)$ is $x_1x_2x_6(x_3 \oplus x_4)(x_3 \oplus x_5)(x_7 \oplus x_8)$, its literal part is $x_1\overline{x}_2x_6$ and its EXOR part is $(x_3 \oplus x_4)(x_3 \oplus \overline{x}_5)(x_7 \oplus \overline{x}_8)$.

Definition 2 *Two 2-pseudoproducts with the same structure are* adjacent *(1) if their EXOR parts are identical, or (2) if their literal parts are identical and the EXOR parts differ in complementation only on some EXOR factors all having a variable in common.*

Example 1 *Consider the following 2-pseudoproducts having the same structure:*

$$\begin{aligned} p_1 &= x_1\overline{x}_2x_6(x_3 \oplus x_4)(x_3 \oplus \overline{x}_5)(x_3 \oplus x_9)(x_7 \oplus \overline{x}_8) \\ p_2 &= \overline{x}_1x_2\overline{x}_6(x_3 \oplus x_4)(x_3 \oplus \overline{x}_5)(x_3 \oplus x_9)(x_7 \oplus \overline{x}_8) \\ p_3 &= \overline{x}_1x_2\overline{x}_6(x_3 \oplus \overline{x}_4)(x_3 \oplus x_5)(x_3 \oplus \overline{x}_9)(x_7 \oplus \overline{x}_8) \\ p_4 &= x_1\overline{x}_2x_6(x_3 \oplus \overline{x}_4)(x_3 \oplus \overline{x}_5)(x_3 \oplus x_9)(x_7 \oplus x_8)\,. \end{aligned}$$

The 2-pseudoproducts p_1 and p_2 are adjacent since they have the same EXOR part; p_2 and p_3 are adjacent since they have the same literal part and differ in the EXOR parts only on EXORs having in common the variable x_3; p_1 and p_4 are not adjacent since they differ in EXORs without a common variable.

We can always merge two adjacent 2-pseudoproducts p_1 and p_2 as follows:

Case 1: p_1 and p_2 differ in their literal parts. Let x_i be the variable with the lowest index that has different complementation in p_1 and p_2, and let p be the 2-pseudoproduct where x_i is complemented. The union of p_1 and p_2 is obtained from p by deleting $\overline{x}_i$, and substituting each literal l_j, having different complementation in p_1 and p_2, with $(x_i \oplus l_j)$.

Case 2: p_1 and p_2 differ in their EXOR parts. Let x_i be the common variable in the EXOR factors with different complementation, and let x_j, $j \neq i$, be the variable with the lowest index in these EXOR factors. Let p be the 2-pseudoproduct where $(x_i \oplus x_j)$ is complemented. The union of p_1 and p_2 is obtained from p by deleting $(x_i \oplus \overline{x}_j)$, and substituting each EXOR factor $(x_i \oplus l_k)$, having different complementation in p_1 and p_2, with $(x_j \oplus l_k)$.

For example consider the 2-pseudoproducts of Example 1. The union of p_1 and p_2 is given by $(x_1 \oplus x_2)(x_1 \oplus \overline{x}_6)(x_3 \oplus x_4)(x_3 \oplus \overline{x}_5)(x_3 \oplus x_9)(x_7 \oplus \overline{x}_8)$. The union of p_2 and p_3 is $\overline{x}_1x_2\overline{x}_6(x_4 \oplus x_5)(x_4 \oplus \overline{x}_9)(x_7 \oplus \overline{x}_8)$. Observe that in the first case the union has less factors, but its EXOR part increases, while in the second case some EXOR factors change, but their number decreases.

Since in many technologies EXOR gates are expensive, the union is not always convenient. By counting the number of literals as explained in Remark 1, we can state:

Theorem 3 *Let C be a 2-SPP cover for a function f, let p_1 and p_2 be two adjacent 2-pseudoproducts in C, and let p be their union. The cost of the cover $C' = C \cup p \setminus \{p_1, p_2\}$ is less than the cost of C if*

1. *$2 + k - 4 * E_n \geq 0$ if p_1 and p_2 differ on their literal part (case 1);*
2. *$2 + k + 4 * E_v - 4 * E_n \geq 0$ if p_1 and p_2 differ on their EXOR part (case 2);*

where

k *denotes the number of factors in p_1, or p_2,*

E_n *is the number of new EXORs in the union of p_1 and p_2 introduced for the first time in the network, and*

E_v *is the number of EXOR factors in p_1 and p_2, which are not factors of the union of p_1 and p_2 and of any other pseudoproduct in the network.*

Our heuristic algorithm performs the union only when it is convenient according to the previous considerations. It is important to notice that the choice of not merging two 2-pseudoproducts does not change the testability of the obtained network.

4.2. EXOR-EXPAND

The operation EXOR-EXPAND tries to remove each EXOR factor $(x_i \oplus x_j)$ of a 2-pseudoproduct $(x_i \oplus x_j)p$ in order to obtain an AND-irredundant (and smaller) cover

of the function. If an EXOR factor $(x_i \oplus x_j)$ can not be removed without changing the function, EXOR-EXPAND tries to replace $(x_i \oplus x_j)p$ with x_ip, x_jp, $\overline{x}_ip$, or $\overline{x}_jp$ in order to guarantee the EXOR-irredundancy of the 2-pseudoproduct. Note that EXOR-EXPAND does not subsume the EXPAND operation. For example, consider the 2-SPP cover $(x_1 \oplus x_2)x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$ of the function f in Figure 1. Since we cannot remove $(x_1 \oplus x_2)$ from the 2-pseudoproduct $(x_1 \oplus x_2)x_3x_4$ without changing the function, we try to replace it with x_1, x_2, $\overline{x}_1$, or $\overline{x}_2$. Observe that we can replace $(x_1 \oplus x_2)x_3x_4$ with $\overline{x}_2x_3x_4$ without changing the function. The resulting 2-SPP form $\overline{x}_2x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$ is now EXOR-irredundant. In general, if $(x_i \oplus x_j)p$ cannot be changed with x_ip, x_jp, $\overline{x}_ip$, or $\overline{x}_jp$, without changing the function, it means that $(x_i \oplus x_j)p$ is EXOR-irredundant, as stated in in the following

Proposition 1 *After the application of EXOR-EXPAND the resulting 2-SPP is EXOR-irredundant.*

4.3. Heuristic Algorithm

We now present a simple heuristic algorithm based on the previous operators. The input to the algorithm is a cover of the function. The loop consists of successive calls to MERGE, EXPAND, EXOR-EXPAND and IRREDUNDANT. The cost is measured after the cover is made irredundant. A new cycle tries to further minimize the cover calling first the REDUCE operator in order to escape from a local minimum. The overall structure of our heuristic algorithm is shown in Figure 2. Observe that the successive calls to the operators MERGE, EXPAND, EXOR-EXPAND and IRREDUNDANT guarantee the EXOR-AND-OR-irredundancy of the resulting cover. Therefore, by Theorem 1, we can conclude that the 2-SPP forms minimized with our heuristics are fully testable.

Finally, note that the cycle EXPAND, EXOR-EXPAND and IRREDUNDANT is sufficient for the synthesis of 2-SPP networks. The MERGE operator is a useful local optimization to replace pair of adjacent 2-pseudoproducts in the cover with their more cost-advantageous union. We have implemented the heuristic algorithm, and the experimental results are shown in the next section.

5. Experimental Results

In this section experimental results for the 2-SPP synthesis heuristics are reported. The methods described above have been implemented in C, using the CUDD library for BDDs and ZDDs [10]. In particular, we have used BDDs to represent Boolean functions and perform the Boolean operations required by the EXPAND, EXOR-EXPAND and IRREDUNDANT procedures. We have used ZDDs to represent 2-SPP covers. The experiments have been run on a Pentium III 850MHz CPU with 256 MByte of main memory. The input benchmarks are PLAs taken from LGSynth93 [14]. We have compared the performances of our heuristics with those of the exact algorithms for 2-SPP and SOP synthesis. The exact 2-SPP forms have been optimized using the tools described in [5], while the SOP forms have been derived using ESPRESSO EXACT. The comparison of synthesis times and network costs are shown in Table 1. The cost is measured according to the CMOS metric described in Remark 1. As expected, the cost of our heuristic solution is still smaller than the SOP costs, but larger than the cost of the optimal exact 2-SPP forms. However the synthesis time is widely reduced with respect to the exact 2-SPP minimization time on average. We have noticed that the synthesis time of the heuristic algorithm is larger than the exact synthesis time only for functions easy to minimize in the 2-SPP framework.

```
C = input cover;
C = MERGE(C);
C = EXPAND(C);
C = EXOR-EXPAND(C);
C = IRREDUNDANT(C);
do {
    μ = cost of C;
    C = REDUCE(C);
    C = MERGE(C);
    C = EXPAND(C);
    C = EXOR-EXPAND(C);
    C = IRREDUNDANT(C);
    μ' = cost of C;
} while (μ' < μ);
```

Figure 2. Heuristic Algorithm for 2-SPP minimization

Our main experimental result consists in the synthesis of new difficult benchmarks in 2-SPP form. This is shown in Table 2, where we compare area, delay and synthesis time of 2-SPP and SOP forms for some benchmarks whose exact 2-SPP form is not known. To this aim we have run our experiments using the SIS system with the MCNC library for technology mapping. Note how areas and delays of the 2-SPP networks are always smaller than those of the corresponding SOP networks, with the exception of *al2* for the delay. From the last row of Table 2 we can observe that the total area of the 2-SPP circuits is about one half of the total area of the SOP forms. On the other hand, since two-level minimization is easier than multilevel synthesis the computational time for the synthesis of SOP forms is much less than the one for the synthesis of 2-SPP forms. To save space we only report costs in Table 1 and mapped areas in Table 2, as reliable indicators of our experiments.

We have finally conducted a testability analysis, under

	2-SPP		Exact 2-SPP		Exact SOP	
Name	**Cost**	**Time**	**Cost**	**Time**	**Cost**	**Time**
9sym	471	29.50	168	93.97	588	3.29
addm4	1126	138.90	694	2928.53	1407	0.02
adr4	174	15.74	105	14.88	415	0.06
clip	651	51.28	402	745.73	769	0.21
dist	749	40.31	471	688.87	879	0.08
f51m	304	30.75	232	26.86	402	0.13
life	293	33.64	180	166.61	756	0.02
m4	1087	81.04	735	561.56	1214	0.39
max512	987	133.70	620	1242.45	1032	0.30
mlp4	665	42.62	500	211.98	869	0.95
newcond	186	18.77	161	520.99	239	0.01
radd	192	7.39	105	17.95	415	0.03
rd53	72	0.50	64	0.10	175	0.01
rd73	272	7.03	212	23.36	903	0.02
root	370	26.47	281	156.05	376	0.05
squar5	106	0.68	101	0.23	120	0.01
xor5	24	0.29	24	0.05	96	0.01
z4	109	4.02	91	2.80	311	0.02
Total	7838	662.63	5146	7402.97	10966	5.61

Table 1. Synthesis times and network costs of 2-SPP, exact 2-SPP and exact SOP forms

	2-SPP			SOP		
Name	**Area**	**Delay**	**Time**	**Area**	**Delay**	**Time**
al2	252	15.3	347.03	340	15.1	14.34
alu2	169	16.1	30.75	176	16.4	0.15
alu3	155	13.3	30.87	187	16.8	0.16
apla	289	17.7	69.09	299	24.5	0.07
bench1	1337	46.2	150.47	1670*	55.6*	0.33*
dk17	140	15.2	44.27	204	18.3	0.04
dk27	48	12.3	10.68	79	12.7	0.03
max1024	1052	39.1	478.60	1690*	53.7*	1.32*
p3	266	22.1	18.83	447	26.7	0.16
prom1	9671	160.0	874.35	19828	399.6	81.67
tial	2294	64.3	1677.19	2376	68.7	14.42
Total	15673	421.6	3732.13	27296	708.1	112.69

Table 2. Area, delay and synthesis time of 2-SPP and SOP forms for benchmarks whose exact 2-SPP form is not known. (A star indicates that the SOP form has been derived with ESPRESSO instead of ESPRESSO EXACT.)

the SAFM, of the 2-SPP networks obtained with our heuristic, using SIS [13]. As already predicted by our theoretical results, the synthesized 2-SPP networks have no redundancies.

6. Conclusions

We presented a heuristic minimization procedure for 2-SPP forms based on the iteration of a suite of operations that generalize to 2-SPP forms the expansion-irredundant-reduction cycle of heuristic SOP minimization, generating by construction a cover that is fully testable for single stuck-at faults. Future work includes addition of new techniques to escape from local mimina, an extension to multiple-output functions, and the investigation of multi-fault testability.

Acknowledgments We are in debt to our student Marco Bordigoni who implemented the heuristic algorithm and carried out the experiments.

References

[1] A. Bernasconi, V. Ciriani, R. Drechsler, and T. Villa. Efficient Minimization of Fully Testable 2-SPP Networks. Technical Report TR-05-23, University of Pisa, 2005.

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

[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 TCAD*, 22(10):1310–1323, 2003.

[5] 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.

[6] 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.

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

[8] T. Kozlowski, E. L. Dagless, and J. M. Saul. An Enhanced Algorithm for the Minimization of Exclusive-Or Sum-Of-Products for Incompletely Specified Functions. In *The Proceedings of the International Conference on Computer Design*, pages 244–249, 1995.

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

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

[11] R. Rudell and A. Sangiovanni-Vincentelli. Multiple-valued minimization for PLA optimization. *IEEE Transactions on Computer-Aided Design*, CAD-6:727–750, Sept. 1987.

[12] T. Sasao. EXMIN2: A Simplification Algorithm for Exclusive-OR-Sum-of Products Expressions for Multiple-Valued-Input Two-Valued-Output Functions. *IEEE Transactions on Computer-Aided Design*, 12:621–632, 1993.

[13] 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.

[14] S. Yang. Synthesis on Optimization Benchmarks. User guide, Microelectronic Center, 1991.