Testability of SPP Three-Level Logic Networks

Valentina Ciriani Anna Bernasconi

Department of Computer Science
University of Pisa
56100 Pisa, Italy
{ciriani, annab}@di.unipi.it

Rolf Drechsler

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

Abstract

Sum of Pseudoproducts (SPPs) are three-level network structures that give a good compromise between compact representation and small depth of the resulting circuit. In this paper the testability of circuits derived from SPPs is studied. For SPPs several restricted forms can be considered. While full testability can be proved for some classes, others are shown to contain redundancies. Experimental results are given to demonstrate the efficiency of the approach.

1. Introduction

The standard synthesis of Boolean functions is aimed at designing circuits of reduced if not minimal cost, according to given cost criteria.

Synthesis is often 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.

Several different logic frameworks have been defined and studied, e.g., two-level logic: SOP [7], Reed Muller [14]; and three level logic: EXSOP [8, 9] (EXOR of ORs of ANDs). A different three-level form called *Sum of Pseudoproducts* (or *SPP*) was introduced in [13]. SPP expressions 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).

Among three-level networks, SPP forms are particularly compact [3, 4]. 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 and, in most technologies, EXOR gates with many inputs are slow, expensive and often impractical [17]. Therefore, in recent studies [6, 5], k*-SPP forms* with a fixed maximum number of literals (k) in the EXOR factors have been introduced. Experimental results 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.

Beside the synthesis aspect, testability is a major aspect of the design process. Up to 40% of the overall design costs are due to testing. For this, aspects of testability should be considered from the very beginning [18]. For several two-level forms detailed studies on testability have been performed. But 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. 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 products, are not fully testable. The same result holds for the specific class of 2-SPPs. Experimental results are given to demonstrate the efficiency of the approach.

The paper is structures as follows: In Section 2 notation and definitions are given. The stuck-at fault model is introduced and basics on SPP networks are reviewed. The testability results are presented in Section 3. In Section 4 details on the experimental setup and the practical results are given.

2. Preliminaries

2.1. Fault Model

Let C be any combinational logic circuit over a fixed library. A fault in the stuck-at fault model [2] 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.

Definition 1 *A* stuck-at fault *with fault location v is a tuple $(v[i], \epsilon)$ or $([i]v, \epsilon)$. $v[i]$ ($[i]v$) denotes the i-th input (output) pin of v, $\epsilon \in \{0, 1\}$ is the fixed constant value.*

For brevity, in the following we simply speak of stuck-at-i (s-a-i) faults, if the context is clear.

Definition 2 *An input t to C is a* test *for a fault F, 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.*

A fault is *testable*, iff there exists a test for this fault. The goal of any test pattern generation process is a *complete* test set for the circuit under test, i.e., a test set that contains a test for each testable fault.

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 [2]. 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*.

For example consider the circuit in Figure 1. A s-a-0 fault at input x_2 can be tested by setting inputs x_3 and x_4 to 1. This is needed to ensure the propagation along the upper AND-gate. Since the EXOR of x_3 and x_4 then becomes 0, the output of the lower AND-gate becomes also 0, ensuring the propagation of the faulty value along the OR-gate at the output. The test is independent of the value of input x_1.

2.2. 2-SPP Networks

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.*

A 2-pseudoproduct P of a Boolean function f is *prime* iff no other 2-pseudoproduct P' of f exists such that $P \subseteq P'$.

Definition 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. In particular, a SOP form is a particular 2-SPP form where each EXOR factor contains only one literal.

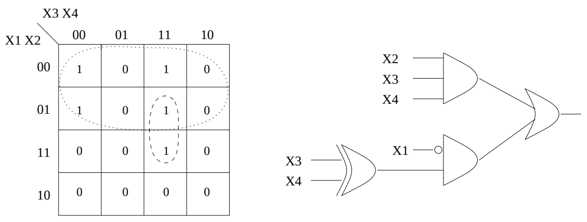

Figure 1. Karnaugh map of function f with 2-SPP cover $x_2x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$, and its 2-SPP circuit representation

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 minimal number of literals or with minimal number of 2-pseudoproducts.

For example, the function f represented by the Karnaugh map in Figure 1, has the following 2-SPP cover minimal with respect to both literals and 2-pseudoproducts: $x_2x_3x_4 + \overline{x}_1(x_3 \oplus \overline{x}_4)$. The 2-SPP circuit representation is on the right side of the figure. A minimal SOP form of such function is $x_2x_3x_4 + \overline{x}_1\overline{x}_3\overline{x}_4 + \overline{x}_1x_3x_4$.

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. For example, the 2-pseudoproduct

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

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. For example, the product $x_1 \cdot \overline{x}_2 \cdot \overline{x}_4 \cdot x_6$ in $\{0, 1\}^9$, corresponds to the system

$$\left\{ \begin{array}{rcl} x_1 & = & 1 \\ x_2 & = & 0 \\ x_4 & = & 0 \\ x_6 & = & 1 \end{array} \right.$$

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:

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

Observe that only the third system has maximum rank, i.e., its equations are linearly independent, and indeed it corresponds to the smaller 2-pseudoproducts. 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 SPP forms, proposed and studied in [3, 13], are a direct generalization of 2-SPP expressions, where the EXOR factors can have an unbounded number of literals.

3. Testability

In this section we study the testability of 2-SPP and general SPP networks.

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 property 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 are not in general fully testable.

3.1. Testability of 2-SPP Networks

3.1.1. Non-Testability of 2-SPP Networks for Product Minimization

We 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. ■

3.1.2. Testability of 2-SPP Networks for Literal Minimization

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 among 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. We are then left only with the case of 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 minimal 2-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 $f_F = x_j \cdot p + s$, 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 $x_j x_i \cdot p \cap \overline{x}_j x_i \cdot p = \emptyset$, we have that $x_j x_i \cdot p \subseteq s$ and $\overline{x}_j x_i \cdot p \subseteq s$, which implies that $x_i \cdot p \subseteq s$. Therefore f contains $(x_i \oplus x_j) \cdot p$ and $x_i \cdot p$. We now show that $x_i \cdot p + (x_i \oplus x_j) \cdot p = x_i \cdot p + x_j \cdot p$. In fact we have

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

Therefore we reach a contradiction to the minimality w.r.t. the number of literals of the 2-SPP form for f. In fact 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. The same proof holds for a s-a-1 fault. ■

3.2. Testability of General 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 products, 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 variables are fully testable.*

Proof. Following the proof for 2-SPP forms we have now 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 $hx_i \cdot p \cap \overline{h}x_i \cdot p = \emptyset$, we have that $hx_i \cdot p \subseteq s$ and $\overline{h}x_i \cdot p \subseteq s$, which implies that $x_i \cdot p \subseteq s$. Therefore f contains $(x_i \oplus h) \cdot p$ and $x_i \cdot p$. We now show that $x_i \cdot p + (x_i \oplus h) \cdot p = x_i \cdot p + h \cdot p$. In fact we have

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

Therefore we reach a contradiction to the minimality w.r.t. the number of literals of the SPP form for f. 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 more fully testable. Let us consider minimal SPP forms depending on a variable ordering (for more details on SPP networks, see [3, 13]). 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 the minimal forms is instead 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.

Therefore 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.*

4. Experimental Results

All 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 carried out in SIS [15]. The benchmarks are taken from LGSynth93 [19].

In a first series of experiments the quality of SPP forms (optimized by different criteria) are compared to two-level approaches. By this, an impression on the quality of the approaches is provided for a set of benchmarks. To this end 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 [10, 12]). The problem is that in many technologies EXOR and OR (or AND) gates have different costs. In [12] the authors consider a 2-input EXOR gate

Table 1. *Costs for benchmark functions in 2-SPP, SOP and SPP forms*

	2-SPP		SOP		SPP		
name	μ	#E	μ'	μ/μ'	μ''	#E	μ/μ''
9sym	168	18	588	0.29	188	30	0.89
addm4	694	34	1407	0.49	*	*	*
adr4	105	5	415	0.25	118	10	0.89
clip	402	26	769	0.52	*	*	*
dist	471	26	879	0.54	636	50	0.74
f51m	232	19	402	0.58	243	23	0.95
life	180	16	756	0.24	180	16	1.00
m4	735	28	1214	0.61	835	48	0.88
max512	620	35	1032	0.60	*	*	*
mlp4	500	25	869	0.58	524	32	0.95
newcond	161	11	239	0.67	*	*	*
radd	105	5	415	0.25	118	10	0.89
rd53	64	6	175	0.37	66	7	0.97
rd73	212	11	903	0.23	187	15	1.13
root	281	21	376	0.75	366	31	0.77
squar5	101	6	120	0.84	112	8	0.90
xor5	24	2	96	0.25	18	1	1.33
z4	91	6	311	0.29	100	10	0.91

as $x \oplus y = xy + \overline{xy}$. 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 for 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 [10].

Table 1 compares 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 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 star * indicates that the SPP algorithm did not terminate after 172800 seconds (corresponding to 2 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 [11, 16] in solving the set covering problem, the number of literals in the expressions in Table 1 are upper bounds for the minimal solutions. The corresponding minimization times are given in Table 2. 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 [1]), which cannot be exploited by the 2-SPP synthesis.

For all forms, the number of redundancies under the stuck at fault model are given in Table 3. 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). Corresponding to the theoretical results in Section 3, it can be observed that 2-SPPs are fully testable (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).

In summary, the experiments have shown 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.

5. Conclusion

Several approaches for three-level synthesis have recently been proposed. The resulting circuits have small delay but are more compact than two-level forms. The algorithmic complexity of the minimization algorithms are moderate. This makes them a promising candidate for synthesis.

In this paper we studied for the first time the testabil-

Table 2. *Minimization times (in seconds)*

name	2-SPP	SOP	SPP
9sym	242.67	5.32	147.58
addm4	50.96	0.87	*
adr4	6.69	0.10	88.22
clip	1662.27	0.38	*
dist	924.10	0.14	8196.00
f51m	64.00	0.23	443.00
life	120.40	0.03	262.00
m4	890.94	0.67	9929.40
max512	341.24	0.53	*
mlp4	339.51	1.62	1423.74
newcond	1485.01	0.01	*
radd	15.20	0.08	144.00
rd53	0.10	0.01	0.20
rd73	24.10	0.03	114.00
root	272.32	0.08	1597.70
squar5	0.42	0.01	0.64
xor5	0.05	0.01	0.02
z4	5.30	0.04	6.75

Table 3. *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

ity of the resulting networks. 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, while for other classes counter-examples were provided. Experimental results demonstrated the efficiency of the approach.

It is focus of current work to study more complex fault models, like path-delay faults or bridging faults.

References

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

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

[3] V. Ciriani. Logic Minimization Using Exclusive OR Gates. In *ACM/IEEE 38th Design Automation Conference (DAC)*, pages 115–120, 2001.

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

[5] V. Ciriani. Synthesis of SPP Three-Level Logic Networks using Affi ne Spaces. *IEEE Transactions on TCAD*, October 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] O. Coudert. Two-Level Logic Minimization: an overview. *INTEGRATION*, 17:97–140, 1994.

[8] 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.

[9] 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.

[10] 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.

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

[12] G. Hachtel and F. Somenzi. *Logic Synthesis and Verifi cation Algorithms*. Kluwer Academy Publishers, 1996.

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

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

[15] 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.

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

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

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

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