A Single–Path–Oriented Fault–Effect Propagation in Digital Circuits Considering Multiple–Path Sensitization

M. Henftling H. C. Wittmann K. J. Antreich
Department of Electrical Engineering
Institute of Electronic Design Automation
Technical University of Munich, 80290 Munich, Germany

Abstract

Various satisfiability problems in combinational logic blocks as, for example, test pattern generation, verification, and netlist optimization, can be solved efficiently by exploiting the fundamental concepts of propagation and justification. Therefore, fault effect propagation gains further importance. For the first time, we provide the theoretical background for a single path oriented fault effect propagation considering both single and multiple path sensitization. We call this approach SPOP. Furthermore, we formulate necessary and sufficient sensitization conditions for SPOP. From these conditions the best suited algebra for propagation can be derived. Experimental results for stuck–at test pattern generation demonstrate that the new approach is orthogonal to D–frontier based methods. We achieve substantial improvements with respect to test pattern generation time and quality.

1 Introduction

In recent years, a lot of improvement has been achieved in design automation for highly integrated circuits. Approaches for logic synthesis [1, 2, 3] and verification [4, 5] achieved a considerable progress. In these approaches modifications of the circuit structure are performed. To verify that these modifications do not change the circuit functionality, it has to be proven that there is no assignment to the inputs of the circuit such that an effect of the modifications is propagated to an output. To perform this proof, test pattern generators for stuck–at faults are used [4, 5].

Many different methods for justification of internal logic values have been developed [6, 7, 8, 9, 10, 11, 12]. However, less attention has been paid to propagation [6, 7]. Most approaches drive a D-frontier or split the circuit into a faulty and a fault–free part.

These methods can be improved with respect to two main aspects. First, in existing approaches the propagation is not directed. The information about the progress is derived from an analysis of the current signal assignment. Second, if the unique sensitization can be performed without conflict, then redundancy identification is very expensive. This paper proposes an approach that overcomes these difficulties.

The approach and its advantages are described in Section 2. In Section 3 we derive the necessary and sufficient conditions for fault effect propagation. Advantages and potential drawbacks of the proposed propagation method are discussed in Section 4. Experimental results using the ISCAS '85 benchmark circuits [13] are provided in Section 5 and conclusions are presented in Section 6.

2 Fault Effect Propagation

2.1 Previous Approaches

The D–Algorithm [6] was one of the first approaches to automatic test pattern generation for stuck–at faults. This approach uses single and multiple path propagation. Therefore, the problem of concurrent justification and propagation is reduced to the task of justification. However, this approach suffers from the enormous amount of possible paths for propagation.

Goel [7] proposes to assign optional values only to primary inputs and to derive the information whether the fault effect is propagated after the implication. In this way, the search space could be reduced. In spite of some heuristics to ease the propagation, a directed propagation was not achieved. Especially, the detection of redundant faults remains a difficult task. This approach has been improved by various procedures. Fujiwara [8] proposed to add an X-Path-Check and unique sensitization to Goel's algorithm to improve especially the redundancy identification. Schulz et al. [9] further extended Fujiwara's method by introducing an improved implication and unique sensitization procedure. Further improvements were suggested, e.g., [11, 14, 15], however, a directed propagation has not been achieved.

Larrabee [10] and Chakradar et al. [16] proposed to duplicate the part of the circuit that is in the transitive fanout of the fault location. Analogous with [11, 14, 15], these approaches do not support a directed propagation.

Muth [17] proposed a nine–valued model for combinational and sequential test generation. Cheng [18] introduced the split circuit model. They emphasized that single path sensitization may be incomplete using the five valued logic for multiple fault effect propagation. However, the convenient operation with its related necessary and sufficient conditions and the relevance for practical problems were not shown.

Stanion and Bhattacharya [12] were the first who tried to take the propagation path into account. At best, they achieved a single path propagation. At worst, they have to duplicate the complete circuit.

2.2 The New Approach

Two facts form the motivation to develop a new propagation approach. First, most redundant faults can be proven redundant during propagation *considering only few gates near the fault location*. Second, for almost all testable faults, single path oriented propagation is better than multiple path oriented propagation. Therefore, we propose an approach for propagation that

- is based on a single path oriented method. We explicitly sensitize a single path p from the fault site x to a primary output o. Exploiting our new method we implicitly keep track of all multiple paths from x to the primary output o that contain p as a single path. This is an advantage compared to single path sensitization without considering multiple paths. We call this propagation method Single Path Oriented Propagation ($SPOP$).
- assures the success of propagation without verification by a backtrace and implication phase. Hence, we completely separate propagation and justification.
- is orthogonal to D–frontier based methods. Experimental results of Section 5 show this situation.
- is applicable for fault effect propagation in sequential circuits. Using SPOP, it is possible to examine all paths from the fault location to the primary outputs completely *before* considering the paths to the flip flop inputs and *before* considering the next time frame.
- uses the best suited algebra for propagation. This algebra is derived from the necessary and sufficient conditions for SPOP.
- is complete, i.e., given enough time a propagation path will be found if a single or multiple propagation path exists (and vice versa).

2.3 Basic Procedure

Figure 1 illustrates the basic steps of the proposed procedure. We start by selecting a target fault. Beginning from the fault location a path to a primary output is selected and this path is sensitized path–segment by path–segment. A path–segment is a subpath which starts at a fanout branch (or a primary input) and ends at the next fanout stem (or primary output). For each selected path–segment the conditions for sensitizing the gates have to be determined. Before choosing the next path–segment, all possible implications are performed to detect locally induced conflicts as soon as possible. We select the next path–segment with a minimum distance to any primary output. This alternating selection of path–segments as well as the sensitizing and implication process is continued until an output is reached. Then, the unjustified values are justified. This procedure implies an orientation to a single path which is constructed step by step by selecting path–segments. It is worth mentioning that, if the implication procedure detects a locally induced conflict, the current path–segment is rejected and an alternative path–segment is chosen for inspection.

In the following section we derive conditions that assure that only Single Path Oriented Propagation has to be considered.

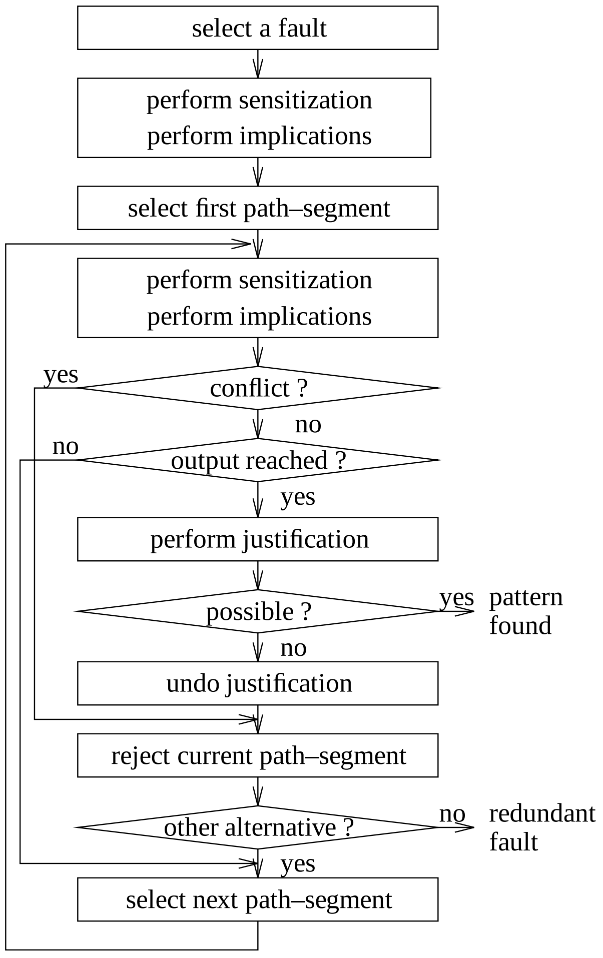

Figure 1: Basic propagation procedure

2.4 Basic Notation

Let us assume an arbitrary circuit with the fault location at signal x and a signal y in the transitive fanout of x, as shown in Figure 2. Signal y is driven by a two–input gate G with the inputs a and b. The symbol $\circ$ stands for any logic function. Therefore, we can formulate the gate operation for the fault–free circuit by $y = a \circ b$ and for the faulty circuit by $y^f = a^f \circ b^f$ $(x, x^f, a, a^f, b, b^f, y, y^f \in \{0,1\})$. In Figure 2, the subpath from x to y via a is drawn in bold. The Boolean difference y_x of signal y with respect to signal x at the fault location is y_x. The general condition for fault effect propagation from x through gate G is

$$\begin{aligned} y_x &= y^f \oplus y \\ &= (a^f \circ b^f) \oplus (a \circ b) && (1) \\ &= [\underbrace{(a \oplus a_x)}_{a^f} \circ \underbrace{(b \oplus b_x)}_{b^f}] \oplus (a \circ b). && (2) \end{aligned}$$

The condition for Single Path Oriented Propagation from x

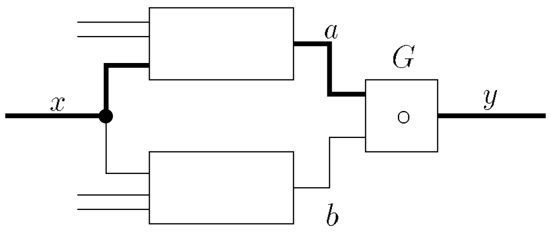

Figure 2: SPOP in a combinational circuit

via a to y is given with (2) by

$$a_x \cdot y_x(a_x = 1) \quad = \quad a_x \cdot ((\overline{a} \circ b^f) \oplus (a \circ b)). \qquad (3)$$

Analogous, the condition for SPOP from x via b to y results in

$$b_x \cdot y_x(b_x = 1) \quad = \quad b_x \cdot ((a^f \circ \overline{b}) \oplus (a \circ b)). \qquad (4)$$

Both conditions are also valid in the case of an n-input gate. The SPOP conditions are necessary and sufficient due to the following relation

$$y_x \quad = \quad a_x \cdot y_x(a_x = 1) + b_x \cdot y_x(b_x = 1) \qquad (5)$$

which can be easily extended to an n-input gate. However, only the simplified equations (3) and (4) have to be exploited for propagation.
That equation (5) is valid, can be shown by Shannon expansion of $y_x(a_x, b_x)$ using cofactors with respect to a_x and b_x and the well–known relation $y_x(a_x = 0, b_x = 0) = 0$ which can be proved easily with equation (2).

3 Necessary and Sufficient Conditions for Fault Effect Propagation

The necessary and sufficient conditions for fault effect propagation can be directly found from equation (3) or (4).

3.1 Conditions for Gates

As described in Section 2.3, the procedure starts at the fault location x and ensures the Single Path Oriented Propagation of the fault effect along the selected path. Hence, the Boolean difference with respect to x at the current end of the path evaluates to 1. E.g., if the path ends at signal a the Boolean difference a_x (x is the location of the fault) equals 1. Since the values of the signals at the fault location for the faulty and the fault–free circuit are known, the value of a is known, too. For example, if y is the output signal of an AND–gate, the term $a_x \cdot (\overline{a} \cdot b^f \oplus a \cdot b)$ has to equal 1 in order to propagate the fault effect to the output of the gate. Inserting the known value of a, the equation can be simplified. For example, $a = 0$ and $a_x = 1$ results in $y_x = b^f$. Therefore, the signal b has to equal 1 in the faulty circuit. There are no requirements in the fault–free circuit. The result of the calculations assuming $a_x = 1$ for an AND-gate and an OR-gate are summarized in the left part of Table 1. The first column contains the type of the gate to be sensitized. The following two columns show the resulting conditions, for $a = 1$ and $a = 0$. In

	$a_x = 1$	
gate	$a = 1$	$a = 0$
AND	$b = 1$	$b^f = 1$
OR	$b^f = 0$	$b = 0$

	$a_x = 1$	
gate	D	$\overline{D}$
AND	$G1$	$F1$
OR	$F0$	$G0$

Table 1: Propagation Values and Symbolic Representation

[17] a symbolic representation for the values of Table 1 has been introduced. The difference between the faulty and the fault–free value of signal a is expressed by D and $\overline{D}$. The sensitization conditions for b are $G1$, $G0$, $F1$, and $F0$, i.e., either the faulty or the fault–free value of b has to be 0 or 1.

3.2 Conditions for High–Level Primitives

In the following we assume a full adder. Inputs are a and b, the carry input is c, and the outputs are n and s. Therefore, the equations for the carry n and the sum s are

$$n = a \cdot b + a \cdot c + b \cdot c$$
$$s = \overline{a} \cdot \overline{b} \cdot c + \overline{a} \cdot b \cdot \overline{c} + a \cdot \overline{b} \cdot \overline{c} + a \cdot b \cdot c.$$

Exploiting (1), s_x and n_x result in

$$\begin{aligned} s_x &= (a \oplus a^f) \oplus (b \oplus b^f) \oplus (c \oplus c^f) \\ n_x &= ((a \cdot b) \oplus (a^f \cdot b^f)) \oplus ((a \cdot c) \oplus (a^f \cdot c^f)) \oplus \\ &\quad ((b \cdot c) \oplus (b^f \cdot c^f)). \end{aligned}$$

The terms $a_x \cdot s_x(a_x = 1)$ and $a_x \cdot n_x(a_x = 1)$ can be calculated and the sensitization conditions can be derived. This step results in Table 2. E indicates that the fault effect is not visible and F means that the fault effect must be observed at the corresponding signal. Similar tables can be obtained for $b_x = 1$ and $c_x = 1$.

output	a=D	a=$\overline{D}$
s	b=E; c=E b=F; c=F	b=E; c=E b=F; c=F
n	b=1; c=0 b=0; c=1 b=D c=D	b=1; c=0 b=0; c=1 b=$\overline{D}$ c=$\overline{D}$

Table 2: Propagation Values and Symbolic Representation

In [19] a procedure, `determine_best_logic`, has been introduced in order to create complete logic systems. Applying `determine_best_logic` to our problem, the 15-valued logic, as proposed in [20] is obtained.

4 Discussion of the new approach

4.1 Advantages of the Proposed Method

The proposed approach has several advantages over the previously published methods:

- Multiple path propagation is implicitly included in **Single Path Oriented Propagation**. Hence, during propagation, all single/multiple paths are considered.

- The success of fault propagation is assured without a simulation based verification. Therefore, propagation is separated from justification.
- Especially for **sequential circuits**, the proposed propagation method is convenient. It is possible to check all propagation paths to primary outputs, i.e., all propagation possibilities within the current time frame, before the next time frame has to be examined.
- Experimental results show that a **fast identification of redundant faults** without justification is achieved. If the fault effect cannot be propagated along the selected path, for almost all faults a conflict occurs during the first implications. Hence, only very few propagation paths have to be examined and this results in a significant reduction of the search space. This statement has been verified for all benchmarks circuits [13, 21]. Typical results can be seen in Table 3. Columns 2 and 3 show the percentage of all faults that need unnecessary justifications to prove the fault as redundant or to generate a test pattern. The columns demonstrate that only a negligible number of faults cannot be classified by the minimal number of justifications, i.e., a sensitized path can almost always be justified.

circuit	justification	
	redundant	testable
c1355	0 %	0 %
c1908	0 %	0 %
c2670	0.3 %	0.2 %
c3540	0.05 %	0.1 %
c6288	0 %	12.7 %
c7552	0.06 %	0.9 %
s1423	0 %	0 %
s5378	0 %	0 %
s9234	0 %	0.07 %
s15850	0 %	0 %
s38417	0 %	0 %
s38584	0 %	0 %

Table 3: Percentage of faults with unnecessary justifications

- The propagation algorithm **avoids any unnecessary effort** for easy testable faults because only necessary assignments are performed.
- **Easy identification** of redundant faults is automatically included.
- The **management** and the **updating** of lists (e.g., the D–frontier) **is not necessary** during the propagation.
- It is **easy** to combine the propagation algorithm with **other ATPG techniques**.
- The proposed propagation technique performs an **efficient pruning** of the search space, since the decisions are performed at the fanout stems in the transitive fanout of the fault location and not at the primary inputs.
- **Many speed–up techniques** are implicitly considered (e.g., dominators, dynamic dominators, X-Path-check, unique sensitization)
- Single Path Oriented Propagation can be applied to other **fault models**, e.g., the gate delay fault model.

4.2 Possible Problems

While developing the new fault propagation method we checked for potential drawbacks.

- Are there many possible paths to sensitize?
  In the worst case, all single paths, starting at the fault location, have to be treated. To examine this point, we have performed stuck–at ATPG for all faults for all circuits in column 1 of Table 4. Columns 2 through 7 show the maximum number of paths sensitized to treat 50 %, 99 %, and 100 % of all faults, i.e., to prove the faults either as redundant (columns 2,4,6) or to generate a test pattern for them (columns 3,5,7). Assume that we want to treat 99 % of all faults (columns 4 and 5). Table 4 shows that we have to sensitize not more than 52 (34) paths per redundant (testable) fault. Moreover, all faults can be classified testable or redundant by sensitizing less than 60 paths. Hence, only few paths have to be sensitized.

circuit	50 %		99 %		100 %	
	red	test	red	test	red	test
c1355	15	1	15	2	15	4
c1908	0	1	7	1	7	2
c2670	0	1	12	3	15	9
c3540	0	1	15	3	15	9
c6288	0	1	0	34	0	48
c7552	4	1	12	12	12	32
s1423	0	1	2	3	3	3
s5378	0	1	9	3	26	33
s9234	0	1	23	9	52	17
s15850	0	1	21	3	49	24
s38417	0	1	14	3	14	19
s38584	0	1	52	3	52	43

Table 4: Number of path to sensitize for a redundant(red)/testable(test) fault

- How to select the propagation path at a branch?
  At a fanout, we first sensitize the branch whose distance to any output is minimal. The results of Table 4 and 5 show that this greedy heuristic is highly efficient.

5 Results

We have implemented the propagation algorithm in a C language program, TIP, and used it for stuck–at test pattern generation. Therefore, a good comparison to previous approaches can be made and the effectiveness of our algorithm is shown impressively.

We have applied the test pattern generator to the ten well-known combinational benchmark circuits [13]. We ran the experiments on a DECstation 5000/200. The results are summarized in Tables 5, 6, and 7.

In an initial experiment, we demonstrate the quality of our approach. Table 5 shows the results of the combinational benchmark circuits for test generation without exploiting a preceding random test generation phase. Column 1 shows the circuit name. Columns 2, 3, and 4 give the number of tested faults, redundant faults, and aborted faults. The test pattern generator is combined with a fast fault simulator. Fault simulation is performed for every generated test pattern. The CPU time for test pattern generation and fault simulation is stated in column 5. The resulting number of test patterns after reverse order fault simulation is shown in column 6. For all circuits, well compacted test sets

circuit	tested	red.	aborted	CPU [s]	patterns
c432	520	4	0	0.6	58
c499	750	8	0	0.7	57
c880	942	0	0	0.4	57
c1355	1566	8	0	1.6	90
c1908	1870	9	0	4.5	127
c2670	2630	117	0	6.6	118
c3540	3291	137	0	12.4	156
c5315	5291	59	0	4.2	123
c6288	7710	34	0	3.5	25
c7552	7419	131	0	21.7	222

Table 5: Results without using RTPG (DEC 5000/200)

have been generated in less than one CPU–minute. Moreover, all circuits feature no aborted faults. This statement is valid for all ISCAS '85 [13] and the combinational part of all ISCAS '89 [21] circuits, too. This is remarkable since (besides the learning procedure of [9]) no procedures and heuristics as, e.g., dominators or unique sensitization, are exploited explicitly. The excellent result of c6288 emphasizes that a large number of possible propagation paths (c6288 has about 10^{20} paths) does not result in a large CPU-time.

Next, we prove that our new approach is not a further improvement of D–frontier based algorithms, but can be seen as an independent method. Therefore, we consider faults that are hard to propagate. We choose Socrates [9], a typical representative of D–frontier propagation, as a reference. We determine two sets of faults for each of the ISCAS '85 benchmark circuits. The first one is the set of faults where Socrates has to perform at least one backtrack. We assume these faults as hard–to–propagate for a D–frontier based algorithm. This set is denoted as HP_D. Analogously, we get the second set HP_O, the hard–to–propagate faults for our approach. We introduce the following metric d.

$$d = \frac{|HP_D \cap HP_O|}{|HP_D \cup HP_O|} \cdot 100\%$$

This metric equals 100% if both approaches consider the same faults as hard–to–propagate; it is 0% if both approaches have to backtrack for disjoint faults, i.e., both approaches are orthogonal. The result of the experiment is

circuit	$\|HP_D\|$	$\|HP_O\|$	$\|HP_D \cap HP_O\|$	d [%]
c432	2	2	2	100.0
c499	17	9	0	0.0
c880	0	0	0	—
c1355	76	22	4	4.3
c1908	21	16	1	2.8
c2670	18	46	8	14.3
c3540	9	14	1	4.5
c5315	18	17	5	16.7
c6288	29	40	3	4.5
c7552	118	92	63	42.9

Table 6: Comparison for hard–to–propagate faults

shown in Table 6. The first column denotes the circuit name. The next three columns represent the number of hard–to–propagate faults $|HP_D|$ and $|HP_O|$ for each approach and the number of hard–to–propagate faults common for both approaches $|HP_D \cap HP_O|$. In the last column, we have the described metric d. With the exception of circuit c432, a small circuit with only two faults to consider, the results demonstrate that both approaches consider almost disjoint fault sets. The metric d is lower than 20 % for seven of the ten circuits. This clearly shows that the new approach is orthogonal to D–frontier based methods.

In a further experiment, we compare the CPU-time for test generation of our approach to those of Socrates [9], Nemesis [10], TRAN [16], CONTEST [14] and the approach of Kundu et al. [22]. Table 7 shows the result of this comparison. All these approaches use a random test pattern

circuit	[9]	[10]	[16]	[14]	[22]	TIP
c432	4.3	7.7	0.8	3	0.3[1]	0.6
c499	4.9	2.9	1.8	7	0.4	0.2
c880	5.2	36.2	3.0	1	0.3	0.2
c1355	13.9	18.8	6.7	20	1.7	0.5
c1908	33.8	66.0	13.0	17	4.8	1.2
c2670	57.5	335.0	95.2	91	6.7[1]	6.6
c3540	56.6	255.0	24.9	32	9.0[1]	3.5
c5315	31.3	65.7	32.1	16	3.2	1.9
c6288	87.0	105.5	44.3	89	20.2	3.4
c7552	247.6	738.0	308.0	284	19.3	19.5
$\emptyset$	25.4	58.1	14.4	19.7	2.7	1.45
CPU	2.0	11.2	21.8	4.5	15.8	19.1
$\emptyset_w$	50.8	650.7	313.9	88.7	42.7	27.7
Speed	1.8	23.5	11.3	3.2	1.6[1]	1.0

Table 7: Comparison of CPU-times [s] for ATPG with RTPG

generation phase to reduce the number of faults that have to be treated explicitly. To ease the comparison, we first perform a random test pattern generation, too. Since the results of the different approaches have been performed on different machines, we compensated the machine speed. The last four lines of Table 7 show the geometric mean of the CPU-times, an approximation of the machine speed (SPECint89), the geometric mean of CPU–times weighted with the corresponding machine speed, and the speed of the test generation normalized to the approach proposed in this paper. All approaches are slower than the proposed approach, ranging from a factor of 1.6 to a factor of 23.5. Please note that [22] has aborted the generation for some redundant faults. Therefore, the approach would be slower if an aborted free result would have been achieved. Socrates is about 1.8 times slower than the proposed approach. However, since only about 5 % of the faults are treated by deterministic test generation, the speed–up for deterministic test generation without random test generation is about 2.3. CONTEST is about three times slower than our approach though it uses a specialized random test pattern generation phase. The remaining two ATPG–tools need more than one order of magnitude more CPU–time than our method.

The proposed approach performs steady excellent for all

[1] aborted faults are left

benchmark circuits, i.e., for easy testable circuits as well as for circuits with hard redundant faults. The approach performs better than state–of–the–art tools that use a D–frontier to propagate the fault effect. Though SPOP performs better than D–frontier based approaches, it is worth mentioning that a combination of both, SPOP and a D–frontier based method, may lead to an interesting algorithmic improvement of propagation and test pattern generation considering the orthogonal behavior demonstrated above.

6 Conclusion

In this paper, we have derived necessary and sufficient conditions for a Single Path Oriented Propagation. These conditions show that single path propagation is sufficient for propagation, i.e., multiple path propagation is implicitly considered. From the sufficient and necessary conditions, best suited logic systems for CAD–tasks can be deduced. Experimental results indicate the high efficiency of the approach. On the one hand, aborted–free ATPG has been presented that is about a factor of two faster than state–of–the–art ATPG tools. On the other hand, we have shown that the proposed propagation approach is orthogonal to the D–frontier propagation.

References

[1] Luis Entrena and Kwang-Ting Cheng. Sequential Logic Optimization By Redundancy Addition And Removal. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 310–315, 1993.

[2] Bernhard Rohfleisch and Franc Brglez. Introduction of Permissible Bridges with Application to Logic Optimization after Technology Mapping. In *Proceedings of the European Conference on Design Automation*, pages 87–93, 1994.

[3] Shih-Chieh Chang and Malgorzata Marek-Sadowska. Perturb and Simplify: Multi-level Boolean Network Optimizer. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, 1994.

[4] Daniel Brand. Verification of Large Synthesized Designs. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 534–537, 1993.

[5] Wolfgang Kunz. HANNIBAL: An Efficent Tool for Logic Verification Based on Recursive Learning. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 538–543, 1993.

[6] J. Paul Roth. Diagnosis of Automata Failures: A Calculus and a Method. *IBM Journal of Research and Development*, pages 278–281, July 1966.

[7] Prabhakar Goel. An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits. *IEEE Transactions on Computers*, pages 215–222, March 1981.

[8] Hideo Fujiwara. FAN: A Fanout-Oriented Test Pattern Generation Algorithm. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 671–674, June 1985.

[9] Michael H. Schulz and Elisabeth Auth. Improved Deterministic Test Pattern Generation with Applications to Redundancy Identification. *IEEE Transactions on Computer–Aided Design*, pages 811–816, July 1989.

[10] T. Larrabee. Test Pattern Generation Using Boolean Satisfiability. *IEEE Transactions on Computer–Aided Design*, pages 4–15, January 1992.

[11] John Giraldi and Michael L. Bushnell. Search State Equivalence for Redundancy Identification and Test Generation. In *Proceedings IEEE International Test Conference*, pages 184–193, 1991.

[12] Ted Stanion and Dehashis Bhattacharya. TSUNAMI: A Path Oriented Scheme for Algebraic Test Generation. In *Proceedings IEEE International Symposium on Fault–Tolerant Computing*, pages 36–43, 1991.

[13] Franc Brglez and Hideo Fujiwara. A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran. In *IEEE International Symposium on Circuits and Systems; Special Session S6AB on ATPG and Fault Simulation*, pages 663–698, June 1985.

[14] Udo Mahlstedt, Thorsten Grüning, Cengiz Özcan, and Wilfried Daehn. CONTEST: A Fast ATPG Tool for Very Large Combinational Circuits. In *Proceedings IEEE/ACM International Conference on Computer–Aided Design*, pages 222–225, 1990.

[15] Wolfgang Kunz and Dhiraj Pradhan. Accelerated Dynamic Learning for Test Pattern Generation in Combinational Circuits. *IEEE Transactions on Computer–Aided Design*, pages 684–694, No. 5, 1993.

[16] Srimat T. Chakradar, Vishvani D. Agrawal, and Steven G. Rothweiler. A Transitive Closure Algorithm for Test Generation. *IEEE Transactions on Computer–Aided Design of Integrated Circuits*, pages 1015–1028, 1993.

[17] P. Muth. A Nine-Valued Logic Model for Test Generation. *IEEE Transactions on Computers*, pages 630–636, C–25, 1976.

[18] Wu-Tung Cheng. Split Circuit Model for Test Generation. In *Proceedings IEEE/ACM Design Automation Conference*, pages 96–101, June 1988.

[19] Karl Fuchs, Hannes C. Wittmann, and Kurt J. Antreich. Fast Test Pattern Generation for all Path Delay Faults Considering Various Test Classes. In *Proceedings European Test Conference*, pages 89–98, April 1993.

[20] Henry Cox and Janusz Rajski. On Necessary and Nonconflicting Assignments in Algorithmic Test Pattern Generation. *IEEE Transactions on Computer–Aided Design*, pages 515–530, April 1994.

[21] Franc Brglez, David Bryan, and Krzysztof Koźmiński. Combinational Profiles of Sequential Benchmark Circuits. In *Proceedings IEEE International Symposium on Circuits and Systems*, pages 1929–1934, May 1989.

[22] Sandip Kundu, Leendert M. Huisman, Indira Nair, and Vijay Iyengar. A Small Test Generator for Large Designs. In *Proceedings IEEE International Test Conference*, pages 30–40, September 1992.