Complexity Comparisons in Logic Testing

J. Sziray

Department of Informatics, Széchenyi University
Egyetem tér 1, H-9026 Győr, Hungary
E-mail: sziray@sze.hu

Abstract: The paper is concerned with analyzing and comparing two exact algorithms from the viewpoint of computational complexity. Both serve for calculating fault-detection tests of digital circuits. The first one is the so-called composite justification, and the second is the D-algorithm. The analysis will be performed on combinational logic networks at the gate level. Here single and multiple stuck-at logic faults will be considered. As a result, it is pointed out that the composite justification requires significantly less computational step than the D-algorithm and its modifications. From this fact it has been conjectured that possibly no other algorithm is available in this field with fewer computational steps. If the claim holds, then it follows directly that the test-calculation problem is of exponential time, and so are all the other NP-complete problems. It may also be expected that the minimal complexity of composite justification applies to any modeling level (either low or high) of digital circuits, just like the exponential-time solution.

Keywords: Computational complexity, test-pattern calculation, logic networks, multi-valued logic, NP-complete problems.

1. Introduction

As known, all computational problems can be classified into two categories: those that can be solved by algorithms, and those that cannot. The first category incorporates problems for which there exists a Turing machine that is guaranteed to halt on all possible inputs, and all such machines will be called algorithms. This principle is known as the Church-Turing thesis [1]. The second category includes problems known to be undecidable, i.e., for which there exists no Turing machine that is guaranteed to halt ever on all possible inputs.

With the enormous advances in computer technology of the past decades (see, e.g., the Moore's law), one may expect that each problem in the first category can today be solved in a satisfactory way. Unfortunately however, computing practice reveals that many problems, although in principle solvable, cannot be solved in any practical sense by computers, due to excessive time requirements [1]–[3].

There are a great number of such problems, and most of them have real practical importance. The common feature of them is that they all are NP-complete. (NP: nondeterministic polynomial.) As known, these problems have a computational complexity for which probably there exists no upper bound by a finite-degree polynomial of the problem size. It means actually that the number of computational steps is finite, but absolutely unpredictable [1]–[3].

Let the problem size be denoted by an integer ***n***. Now we can establish a polynomial of the finite-degree integer ***k***, as follows:

$$P_k(n) = a_k \cdot n^k + a_{k-1} \cdot n^{k-1} + ... + a_1 \cdot n + a_0 ,$$

where $k \geq 1$, and $a_i \geq 0$ is an integer for $i = 0, 1, 2, ... , k–1$, whereas $a_k \geq 1$ is also an integer value.

The computation of solutions for a problem is said to be polynomially bounded if and only if there exists an algorithm which has no more steps than $P_k(n)$, for each possible instance of the problem. In terms of time spent by a computer, this notion can also be expressed as polynomial-time algorithm. Unfortunately, no one has found such an algorithm for any NP-complete task so far, and it is widely believed that there is no algorithm of this category. Instead, it is expected that any algorithm solving the problem has a computational complexity of an exponential nature.

As a matter of fact, all the exact solutions found so far for NP-complete problems are really of exponential complexity. It means that the problem size ***n*** determines the number of algorithmic steps ***S(n)*** in the following way:

$$S(n) = a \cdot b^n,$$

where $a \geq 1$, $b \geq 1$ are constant integers.

At present, the question whether there exists a polynomial solution to NP-complete problems is undecided. The proven answer in this case is one of the most intriguing demands in the field of computation theory. This is all the more important, since if it were a polynomial algorithm to any individual NP-complete problem, then it could be transformed to all other NP-complete problems, as a solution, while preserving its polynomial property [1]–[3].

One of these important problems is the generation of fault-detection tests for digital circuits [4]–[7]. Here the problem size is an integer number, and can be expressed, among others, by the number of gates or transistors in a circuit. In our approach, the number of gates will be used, and denoted by ***n***.

This paper is meant for analyzing and comparing two concrete algorithms from the viewpoint of computational complexity. Both are for calculating fault-detection tests for digital circuits. The first one is the so-called ***composite justification*** [8]–[10], and the second is the ***D-algorithm*** [6], [7] developed by Roth. The analysis will be performed on combinational logic networks at the gate level. Here single and multiple stuck-at logic faults will be considered at the primary inputs and the gate outputs.

2. Treatment of single faults

Let the vector of primary input and output variables for a general logic network be $\overline{x} = (x_1, x_2, \ldots, x_n)$ and $\overline{z} = (z_1, z_2, \ldots, z_m)$, respectively. Let the set of possible logic values in the network be $V = \{v_1, v_2, \ldots, v_s\}$. In addition to the elements of V, the indifferent or don't care value $\boldsymbol{d}$ will be applied.

In case of **composite justification** the computations are carried out simultaneously in the normal and faulty network, i.e., in the normal and the faulty domain. Logic values simultaneously representing signal values in both the normal and the faulty networks are called **composite values.** Line justification performed in terms of composite values is referred to as **composite justification** [8], [9]. The two components of a composite value will be separated by a slash, with the normal component preceding the faulty one. The actual logic value of the $\boldsymbol{i}$-th line in the network will be denoted by $\boldsymbol{v(i)}$. Then, for example, a composite value of line $\boldsymbol{i}$ is

$$v(i) = v_a / v_b$$

Let the set of lines with stuck faults be denoted by $\boldsymbol{SL}$, where the number of lines belonging to SL is q. If the stuck value at line i of SL is $s(i)$, then the initial set of the logic values that are to be justified will be as follows:

$$z_j = \alpha / \beta \text{ for a selected primary output, where}$$

$$\alpha \in V, \quad \beta \in V, \text{ and } \alpha \neq \beta, \text{ and}$$

$$v(i) = d / s(i) \quad \text{for each} \quad i \in SL.$$

In the justification process, the value $\boldsymbol{d}$ need not be justified, whereas the stuck values $s(i)$ must not be justified. The reason for d is obvious, while the stuck values are self-consistent.

Below we introduce four more notations:

Let the **stuck-at-α** fault of the **i-th line** be denoted by **i(α)**. The apostrophe sign will stand for logic inversion. The network path containing the series of lines i, j, …, p will be denoted by

$$\mathbf{P(i - j - \ldots - p)}.$$

Network lines that carry signals from the faulty lines to z_j. are referred to as **potentially active lines** or **PAL's**. Here **PAL(i – j)** denotes the set of potentially active lines between lines $\boldsymbol{i}$ and $\boldsymbol{j}$.

As an example, consider the combinational network shown in Figure 1, where a test is to be calculated for fault **7(1)**. In this case

$$\text{PAL}(7 - 16) = \{7, 9, 10, 11, 12, 14, 16\}.$$

The potentially active lines are marked heavy in the schematic. The figure indicates the logic values assigned to the lines in the network upon completion of the calculation process.

When starting, at first z_j is set to 0 / 1, and whenever a contradiction occurs at the fault site, we interchange the components of each composite value, and proceed with the calculation in the same way as before [8].

The obtained test consists of the 0 or 1 components of the final composite values at the primary input lines:

$$\overline{x}_t = (d, 0, 0, d, d, 0).$$

As it can be seen, the test simultaneously sensitizes paths P(7-10-11-14-16) and P(7-10-12-16), while it leaves path P(7-9-14-16) necessarily unsensitized. The sensitized paths propagate the faulty signals towards the primary output z_1.

At the end of the calculations, those lines which actually carry the fault signal to z_j will be the real **active lines**. These lines have the so-called **active composite values** which are equivalent with the fault signals [8].

In the following an evaluation of the composite justification algorithm is given, in comparison with the D-algorithm. From a computational point of view, it is sufficient to take into account only one primary output of fault detection, say z_j.

In the event of a single fault $i(\alpha)$ the D-algorithm attempts to simultaneously sensitize a combination of paths leading from i to z_j. If successful, line justification is performed for the initially established set of sensitizing logic values. In the worst case each possible combination of paths must be processed in the same way until a consistent a test pattern is found.

If the number of different single paths leading from $i(\alpha)$ to z_j is $\boldsymbol{p}$, then the D-algorithm is to be performed $2^p - 1$ times in worst case. This number is $2^3 - 1 = 7$ for the network in Figure 1. As opposed to this, composite justification has only **one single** computational process, independently of the number of paths.

It has been pointed out by Muth in [11] that the use of Roth's five-valued logic (i. e., 0, 1, D, D', and d) in the D-propagation phase means overspecification which may lead to a large amount of wasteful computation. The proposed modification in [11] was to use nine-valued logic in the D-algorithm, instead of the original five-valued. This logic is based upon composite values. In worst case the modified D-algorithm attempts to propagate the faulty signal via each possible single path from i to z_j, and performs line-value justification in each case. Thus, the amount of computation in both the original and the modified D-algorithm is greatly influenced by the number of paths which may occur due to reconvergence. In our example in Figure 1, this number is **three**.

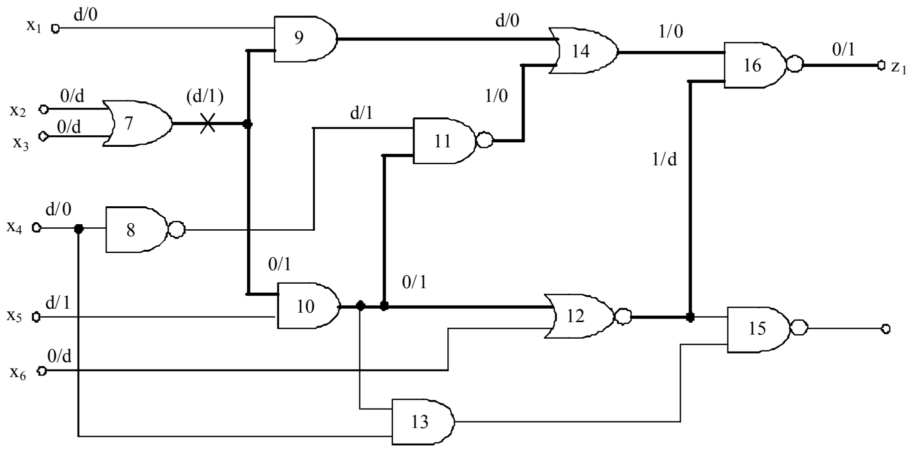

Figure 1. Composite justification for a single stuck-at fault

In contrast with these solutions, the composite justification algorithm leaves out the path sensitization phase along with the successive attempts to justify the different fault-propagation possibilities. Instead, it performs only **one justification process**, thus finding a test as a direct result. Moreover, this solution is affected by the presence of reconvergent paths to a much lesser extent than even the modified D-algorithm. The number of paths between i and z_j affects the amount of potentially active composite values used during justification. These are the values on the potentially active lines. For lines that do not belong to these paths, the determined logic values (i.e., 0 and 1) must not differ in the two domains. They are called **passive lines** in [8]. This fact results in a considerable reduction in the amount of possible decisions and backtracks.

3. Treatment of multiple faults

In the next example a test is to be found for the multiple fault 5(1), 6(0), and 9(1) as shown in Figure 2. The PAL's are also marked heavy in the schematic. When applying composite justification, the final result shows that the test vector $\overline{x}_t$ = (0, 1, 1, 0) propagates fault 6(0) along P(6-8-9-10) and fault 9(1) along P(9-10) simultaneously, whereas fault 5(1) remains blocked from propagation.

Figure 3 presents another solution for this same task. Here the obtained test vector $\overline{x}_t$ = (1, 0, d, d) propagates only fault 5(1) along P(5-7-10) to the primary output. The other two faults will not be propagated by the test.

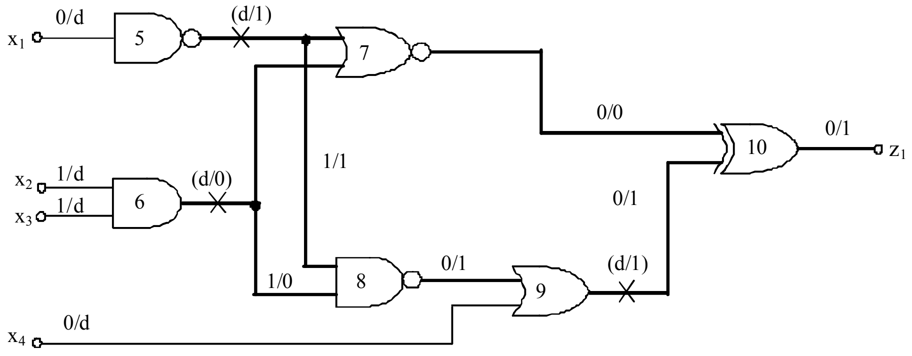

Figure 2. Multiple-fault test calculation. Result 1.

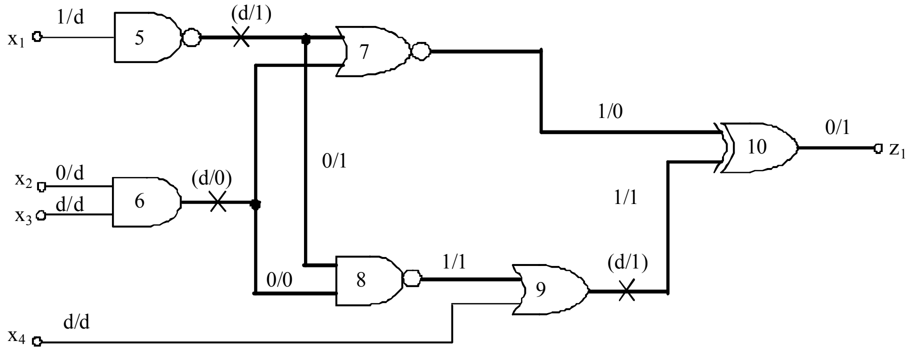

Figure 3. Multiple-fault test calculation. Result 2.

In order to find a test for a multiple fault, attempts are made to propagate different combinations of the individual faults to z_j when using the D-algorithm [6], [7]. If the number of faults is $\boldsymbol{q}$, a maximum of $2^q - 1$ attempts are required in worst case. In our example, this number is $2^3 - 1 = 7$. As a matter of fact, it should be remembered that within a single attempt, the D-algorithm has to be repeated $2^p - 1$ times, even for a single fault, where $\boldsymbol{p}$ is the number of the different paths from the fault site towards z_j. In our example this number for 5(1) is 3, for 6(0) is 3, and for 9(1) is 1, which results in the sum of 7. Thus, just the separate single-path propagations would require a maximum of $7 \times 7 = 49$ attempts in worst case. And so far we have not counted the possible multiple-fault propagations.

On the other hand, the D-algorithm becomes more complicated if more than one fault effect must be propagated simultaneously. In this case, the individual path combinations for the participant faults will be summed up, thus yielding the possible maximum number of attempts. Anyway, in general, the overall sum of computational steps may be a significant multiple of $2^q - 1$ in worst case.

For the modified D-algorithm it is sufficient to attempt to propagate the individual faults separately. Hence, this approach implies a maximum of ***q*** fault-propagation attempts [11]. In our example it is **three.**

The composite justification for multiple faults needs to be performed at first for $z_j = 0 / 1$, and if it fails then repetition for $z_j = 1 / 0$ is required. It means that the number of attempts in worst case is only **two,** independent of the multiplicity of faults. Nevertheless, this algorithm is only slightly affected by the number of faults. In addition, we do not have to iterate through all the possible fault propagation combinations. Instead, a combination necessarily results from the justification process itself, as illustrated in Figure 2 and 3.

4. The thesis of minimal complexity

The efficiency of composite justification is based on the fact that it establishes **the minimal necessary and sufficient set of logic values** which yield the test conditions for the faults. As seen, the tests are obtained only by justifying this initially set logic conditions. This fact implies **the total absence of the fault propagation phase,** which is a salient advantage of composite justification

There exist other test generation algorithms like PODEM [13] and FAN [14] that have been proved to be more efficient than the D-algorithm for single faults. As a matter of fact, the acceleration results in the PODEM and FAN are relied on some clever improvements on the original D-approach, by using structural and logic analyses, while the overall fault propagation and justification principle remained unchanged.

When multiple faults are considered, the complete procedure of the D-algorithm has to be repeated $2^q - 1$ times in worst case, for one primary output, where q is the multiplicity of faults. In this case, attempts are made to propagate different combinations of the individual faults. The modified D-algorithm has ***q*** attempts in worst case. On the other hand, composite justification requires only **two iterations** in worst case, also for one primary output.

From the above results the following **thesis** can be conjectured:

The composite justification algorithm has an **absolute minimal** computational complexity in generating a test for any single or multiple stuck-at faults in logic networks at the gate level. In other words, there exists no other exact algorithm which would need fewer computational steps.

Under the assumption that this thesis holds, the following consequences will be implied:

As known, the justification process itself is nothing else than traversing downward and upward along a given decision tree, until a solution is found. All this means that the process has an exponential complexity, depending on the number of nodes within the tree. On the other hand, since all NP-complete problems have the same property in terms of complexity, each of them can only be solved by some exponential-time algorithm in general. If so, it would mean that no polynomial-time solution is available for any of the NP-complete problems in general.

As far as the computer implementation is concerned, only line justification is to be accomplished in the presented principle, which is also an advantage. In order to perform line-value justification, the **inverse models** of the building elements in the network are required. An inverse model defines the set of possible input patterns which result in a given output pattern.

After all, the process may also be called **inverse simulation**, as opposed to the normal one, where the outputs are calculated with knowledge of the inputs. Thus, the two simulation approaches have opposite directions in treating the logic elements within a circuit.

5. Application of different modeling levels

The fact that composite justification is in essence an inverse simulation algorithm which avoids fault propagation, implies the following consequences:

1) The algorithm enables the user to model the digital circuits at various levels, namely:

- **Switch level for CMOS circuits:** the building elements are transistors.
- **Gate level:** logic gates are applied exclusively.
- **Functional level:** the logic values of an element are calculated with knowledge of its external functional behavior. For this modeling purpose, high level **hardware-description languages (HDL's)** can be applied, for instance, VHDL.
- **Register-transfer level (RTL):** the behavior of the digital system is described by means of bit vectors that are processed and transferred among various building blocks.

2) The feature that composite justification avoids direct fault propagation makes the approach extremely flexible in terms of circuit modeling and fault classes [9], [10], [12]. As known, D-propagation is an inherent part of the D-algorithm, where this phase implies serious difficulties for functional level models and multiple faults [7]. Functional algorithms for constructing computational tools of complex logic modules have been presented in [15]. This paper clearly illustrates the problems encountered in this topic.

As a matter of fact, the acceleration results in the PODEM and FAN are relied on the use of gate-level network structure. However, if functional level modeling is considered then the structural and logic analyses performed in both algorithms become extremely cumbersome and hardly feasible. In contrast, the approaches based on composite justification are not really limited by the way of network modeling. The same applies to the network types, i.e., combinational or sequential, and also, the types and multiplicity of the faults [9], [12].

At this point, it should be mentioned that in case of sequential circuits, multiple-fault calculation is necessary even for a single fault which may propagate to various storage elements, and there manifests itself in a virtual appearance.

For a sequential logic element, an inverse model defines the set of possible input patterns which result in a specific **state** or an **output** pattern [15], [16]. For this purpose, high level hardware-description languages, such as VHDL or any RTL language can also be applied.

3) It is known that generating fault-propagation tests is equivalent with generating tests for delay faults [17], [18]. It means that all the advantages related to composite justification are equally valid for delay testing as well [12].

4) As for transistor-level models, [19] has shown how the D-algorithm can be extended to CMOS circuits, while [20] has demonstrated the same with composite justification.

5) Finally, as a result of the above considerations, it may also be expected that the **thesis of minimal complexity** conjectured in Section 4 applies in general to any categories of test calculation problems for logic networks, independent of the way of modeling and fault type.

6. Conclusions

This paper has been meant for showing how the test calculation algorithm first published in [8] can be compared with the D-algorithm and its modifications. The comparison results demonstrate that the former one requires significantly less computational step than the D-algorithm and its improvements. From this fact it has been conjectured the thesis that possibly there is no other algorithm for the given problem with fewer computational steps. The implication of the thesis is that test calculation is definitely of exponential time, and so are all the other NP-complete problems.

As a result of further analyses in the paper, it may also be expected that the thesis of minimal complexity applies in general to any categories of test-generation problems for logic circuits, independent of the modeling level and fault types. From this it follows that the solutions to be obtained are also of exponential time.

The importance of exact algorithms manifests itself in the necessity of solving a practical problem at the cost of an enormous but still acceptable computation time. Such a requirement is to reach a high-degree fault coverage of a test set for digital circuits. Usually, it is far above 90%.

As known, there are random and heuristic methods that perform well in producing test sets. In general, these approaches are necessarily combined with fault simulation. Recently, there have been published new heuristic approaches which proved to be very efficient in treating VLSI (Very Large-Scale Integration) circuits, and a wide range of fault models [21], [22]. The paper [21] provides a comprehensive framework for treating various defects in a logic circuit, by applying an approach based on Cause-

Effect analysis [7]. As far as the paper [22] is concerned, it presents two new methods for the generation of test sets with a small number of specified bits.

However, if the fault coverage achieved is not satisfactory, as is often the case with heuristic methods, an additional test-generation attempt is required for the undetected faults. In this situation, only an exact algorithm can be taken into account.

Acknowledgements

This work was supported in part by the project TAMOP 4.2.1.B at the Széchenyi University of Győr.

References

[1] H. R. Lewis, Ch. H. Papadimitriou: *Elements of the Theory of Computation,* Prentice-Hall, Inc., USA, 1998.

[2] J. E. Hopcroft, R. Motwani, J. D. Ullman: *Introduction to Automata Theory, Languages, and Computation,* Second Edition, Addison-Wesley Publishing Company, USA, 2001.

[3] T. H. Cormen, Ch. E. Leiserson, R. L. Rivest, C. Stein: *Introduction to Algorithms,* McGraw-Hill Publishing Company, USA, 2001.

[4] O. H. Ibarra, S. K. Sahni: *Polynomially complete fault detection problems,* IEEE Trans. On Computers, Vol. C-24, pp. 242–249, March 1975.

[5] H. Fujiwara: *Logic Testing and Design for Testability,* MIT Press, USA, 1985.

[6] J. P. Roth: *Diagnosis of automata failures: A calculus and a method,* IBM Journal of Research and Development, Vol. 10, pp. 278–291, July 1966.

[7] M. Abramovici, M. A. Breuer, A. D. Friedman: *Digital Systems Testing and Testable Design,* Computer Science Press, USA, 1990.

[8] J. Sziray: *Test calculation for logic networks by composite justification,* Digital Processes, Vol. 5, No. 1–2, pp. 3–15, 1979.

[9] J. Sziray: *A comprehensive method for the test calculation of complex digital circuits,* Periodica Polytechnica, Budapest Techn. Univ., Series of Electronic Eng., Vol. 41, No. 4, pp. 251–257, 1998.

[10] J. Sziray: *Functional level test calculation and fault simulation for logic networks,* Discrete Simulation and Related Fields (Ed. by A. Jávor), pp. 223–234, North-Holland Publishing Company, Amsterdam, 1982.

[11] P. Muth: *A nine-valued circuit model for test generation,* IEEE Trans. on Computers, Vol. C-25, pp. 630–636, June 1976.

[12] J. Sziray: *Test Calculation for Logic and Delay Faults in Digital Circuits,* IEEE Microprocessor Test and Verification Workshop, (MTV-06), Proceedings, pp. 20–29, Austin, Texas, USA, December, 2006.

[13] P. Goel: *An implicit enumeration algorithm to generate tests for combinational logic circuits,* IEEE Trans. on Computers, Vol. C-30, pp. 215–222, March 1981.

[14] H. Fujiwara, T. Shimono: *On the acceleration of test generation algorithms,* IEEE Trans. on Computers, Vol. C-32, pp. 1137–1144, December 1983.

[15] M. A. Breuer, A. D. Friedman: *Functional level primitives in test generation,* IEEE Trans. on Computers, Vol. C-29, pp. 223–235, March 1980.

[16] J. Sziray, Zs. Nagy: *OPART: A hardware-description language for test generation,* Microprocessing and Microprogramming, (Ed. by P. Nunez), pp. 525–530, North-Holland Publishing Company, Amsterdam, 1991.

[17] C. J. Lin, S. M. Reddy: *On delay fault testing in logic circuits,* IEEE Trans. on Computer-Aided Design, Vol. CAD-6, pp. 694–703, September 1987.

[18] I. Pomeranz, S. M. Reddy: *A generalized test generation procedure for path delay faults,* FTCS-28, International Symposium on Fault-Tolerant Computing, Proceedings, pp. 274–283, Munich, June 1998.

[19] S. K. Jain, V. D. Agrawal: *Modeling and test generation algorithms for MOS circuits,* IEEE Trans. on Computers, Vol. C-34, pp. 426-433, May 1985.

[20] J. Sziray: *Switch-Level Test Calculation for CMOS Circuits,* IEEE 10th International Workshop on Microprocessor Test and Verification, (MTV-09), Proceedings, pp. 58–65, Austin, Texas, USA, December 7–9, 2009.

[21] A. Bosio, P. Girard, S. Pravossoudovitch, A. Virazel: *A Comprehensive Framework for Logic Diagnosis of Arbitrary Defects,* IEEE Trans. on Computers, Vol. C-59, pp. 289–300, March 2010.

[22] S. N. Neophytou, M. K. Michael: *Test Set Generation with a Large Number of Unspecified Bits Using Static and Dynamic Techniques,* IEEE Trans. on Computers, Vol. C-59, pp. 301–316, March 2010.