PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 41, NO. 4, PP. 251-257 (1997)

A COMPREHENSIVE METHOD FOR THE TEST CALCULATION OF COMPLEX DIGITAL CIRCUITS

József SZIRAY

Széchenyi College
Faculty of Informatics and Electronic Engineering
Hédervári út 3,
H-9026 Győr, Hungary
E-mail: sziray@rs1.szif.hu

Received: June 5, 1997

Abstract

The paper presents a general test calculation principle which serves for producing tests for a wide range of possible faults: stuck-at-constant logic level (single, multiple), bridging (single), as well as behavioural (functional, single) faults. The proposed method handles multi-valued logic, where the number of logic values is unlimited. The level of circuit modelling is also allowed to vary in a wide range: switch level, gate level, functional level, etc. are equally allowed. Both combinational and sequential circuits are considered. The principle is comparatively simple, and it yields an opportunity to be realized by an efficient computer program.

Keywords: test-pattern calculation, fault modelling, multi-valued logic, modelling of logic networks, hardware-description languages.

1. Introduction

For the purposes of **built-in self testing (BIST)**, an increasing demand is experienced for possessing efficient test sets that are capable of detecting a wide variety of faults in complex digital circuits. The fulfilment of this demand is especially important in the field of VLSI CMOS circuits which are rapidly spreading in the modern hardware construction.

This paper is intended to present a general test calculation principle that is suitable for producing tests for a wide range of possible faults: stuck-at-constant logic level (single, multiple), bridging (single), as well as behavioural (functional, single) faults. The proposed principle handles multi-valued logic, where the number of logic values is unlimited. All of those representations are accepted here which operate in the logic domain by applying multi-valued logic. For instance, if the circuit behaviour is described by the **VHDL language** [1], the logic values will correspond to the signal values in the actual description. Moreover, the level of circuit modelling is also allowed to vary in a wide range: switch level, gate level, functional level, etc. are equally allowed. (The comprehensive circuit representation based on alternative graphs in [2] can also be accepted here.) At the same time,

the proposed principle is comparatively simple, and it yields an opportunity to be realized by an efficient computer program, both for combinational and for sequential circuits.

2. The Test Calculation Principle

The test calculation principle to be presented enables the user to model the digital circuits at various levels, namely:

- **Switch level** for MOS circuits: the transistors are the building elements.
- **Gate level**: logic gates are applied exclusively.
- **Functional level**: the logic values of an element are calculated with knowledge of its external functional behaviour. For this modelling purpose, high level **hardware-description languages (HDL's)** can be applied, for instance, VHDL.

Let the vector of primary input and output variables for the complete network be $\bar{x} = (x_1, x_2, \ldots, x_n)$ and $\bar{z} = (z_1, z_2, \ldots, z_m)$, 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 d will be applied.

Suppose that a sequence of primary input vectors $X(t) = \overline{x}_1, \overline{x}_2, \ldots, \overline{x}_t$ detects $q \geq 1$ simultaneous faults at a primary output z_j. Now the task of calculating $X(t)$ can be stated in the following way. Find a sequence of input patterns which implies $z_j = \alpha$ in the fault-free network, and $z_j = \beta$ in the faulty network, where $\alpha \in V$, $\beta \in V$, and $\alpha \neq \beta$.

To reach this goal, we associate the logic values α and β with z_j, and attempt to find a sequence of input patterns which equally justifies

1) the normal value of z_j for the normal (fault-free) network, and
2) the faulty value of z_j for the faulty network.

In the first case, $X(t)$ justifies $z_j = \alpha$ through the values of all the necessary network lines in the usual manner. In the second case, however, since the faulty values are self-dependent, they need not be justified by $X(t)$. Thus, in the faulty network, $X(t)$ and the faulty values justify $z_j = \beta$ jointly.

The test sequence can be derived by applying the **line-value justification concept** . As known, line-value justification is a procedure with the aim of successively assigning input values to the logic elements in such a way that they are consistent with each previously assigned value. (This concept is an auxiliary calculation process for justifying an initial set of logic values in a network, first applied in the D-algorithm for two-valued logic [3], [4].)

In our approach, 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** [5], [6]. 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 i-th line in the network will be denoted by $v(i)$. Then, for example, a composite value of line i is

$$v(i) = v_i/v_j \ .$$

In the composite justification the computational costs can be greatly reduced by the following consideration [5]. The signal values in the normal and faulty networks cannot differ at the lines that do not carry any signals propagating from the sites of the faults. These are called **inactive lines**, for which $v_i \neq v_j$ would represent inconsistency if they had the composite value of v_i/v_j. It should be realized that for our purposes it is sufficient to determine which lines (called **potentially active lines**) carry signals from the faulty lines to z_j. This holds true, since all the other lines in the network are either inactive or are not involved in the justification process. The set of potentially active lines can be generated by intersecting the set of the lines that are reached from the faulty lines with the set of the lines from which z_j can be reached. The two sets are very easy to obtain by topologically tracing out the signal connections. This is done by starting from the faulty lines and proceeding forward, then starting from z_j, and proceeding backward. If a line is encountered in both the forward and backward tracing then it is potentially active.

The implementation of the above principle for a synchronous sequential network may require the justification process to be performed through different storage states of the network, which results in a test sequence $X(t)$ of length t. The detailed principle of doing it for stuck-at-0 / 1 faults is described in [5], [6].

In the following, we are going to present the necessary and sufficient conditions for performing the composite justification in case of different fault models.

3. Tests for Different Fault Models

3.1. Stuck-at-Constant Logic Level Faults

An arbitrary signal is erroneously preserving a certain logic value (for example, logic 0 or 1, high impedance value, etc.), independent of the actual control values which may influence the normal value of this signal. Any logic values are allowed within the range of possible values for that particu-

lar signal. It should be added that not only single faults, but also multiple faults are included, with no limits in their multiplicity.

Let the set of lines with stuck faults be denoted by 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:

$$z_j = \alpha/\beta$$

for a selected primary output, where

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

and

$$v(i) = d/s_i \quad \text{for each} \quad i \in S_L\,.$$

In the justification process, the value d need not be justified, whereas the stuck values s_i must not be justified.

3.2. Bridging (Coupling) Faults

They are defined for two signals. This type of fault occurs when a signal erroneously takes up the normal logic value of another signal. In this case the two signals are said to be bridged (coupled). Only single occurrence is stipulated here for the bridging faults.

Let the dominant logic value of two bridged lines be ω: It means that the value ω will appear on both lines, instead of the normal different values. If the bridged lines are b_1 and b_2 then the initial values for the composite justification will be as follows:

$$z_j = \alpha/\beta$$

for a selected primary output, where

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

and

$$v(b_1) = v(b_2) = d/\omega\,.$$

In the justification process, the following measures have to be taken for the bridged lines:

a) If either of them is reached by a determined logic value (i.e., a value other than d) in the normal domain, then the value of the other bridged line must be set to ω/ω.
b) If in a backtrack process, b_1 or b_2 is reached, then the initial value d/ω has to be assigned to both lines again, as the solution when cancelling the former decisions, i.e., when cancelling the former calculated logic values.

3.3. Behavioural (Functional) Faults

This type of fault is related to a logic module M_i for which the faulty behaviour is to be previously defined by the user. The definition can be made by means of a high level HDL (e.g., VHDL), where the faulty output responses are given explicitly, at the presence of the faultless input signals or the faultless input sequence obtained by M_i. In our case, only single-module faults will be handled at a time.

The initial values to be justified are:

$$z_j = \alpha/\beta$$

for a selected primary output, where

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

and the input/output signals of M_i in the normal domain, as well as the faulty output signals of M_i in the faulty domain.

3.4. Switch Level Faults

The fault model of MOS circuits includes stuck-at-1/0 logic faults on the connecting lines, and switch faults in the transistors: stuck open (the transistor is erroneously in an open circuit state), and stuck short (the transistor is erroneously in a short circuit state). The paper [7] describes a procedure to convert a transistor structure into an equivalent logic gate structure. In this way the transistor faults can be converted to stuck faults in the logic structure. All this makes it possible to use composite justification for single and multiple faults in MOS circuits, without any modifications.

On the other hand, it is also possible to apply the direct switch level (transistor) structure when performing composite justification. Here the initial settings are:

$$z_j = \alpha/\beta$$

for a selected primary output, where $\alpha \neq \beta$.

The possible values for α and β are logic 0 and 1, as well as high impedance state (Z). Furthermore, the faulty transistors are to be in their faulty state (open or short) in the faulty domain.

In this approach line justification involves the tracing back of the possible switching paths through the transistors in the circuit. The detailed description of the algorithm exceeds the frames of this paper.

4. Concluding Remarks

This paper has been meant for showing how the test calculation algorithm first published in [5] can be generalized for a wide range of fault models and circuit models. The flexibility of the 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 by justifying the initial logic conditions. The salient advantage of composite justification is the total absence of the **fault propagation** phase. This feature makes the approach extremely flexible in terms of circuit modelling and fault classes. The same applies to the use of an HDL [8], [9]. As known, the fault propagation phase, i.e., D-propagation is an inherent part of the wide-spread D-algorithm [3] [4], where this phase implies serious difficulties for functional level models and multiple faults.

The test generation algorithm called PODEM (Path-Oriented DEcision Making) [11], [12] was shown to be more efficient than the D-algorithm. The improvement is related to the decision tree of the line justification, in that the PODEM algorithm has indeed succeeded in reducing the occurrences of backtraces, in comparison with the D-algorithm. In the D-algorithm, since the assignment of values is allowed to internal lines, more than one choice is available at each internal line or gate, and backtracking could occur at each gate. In contrast, the PODEM algorithm allows assigning values only to primary inputs. The values assigned to primary inputs are then propagated toward internal lines by the logic implication. Thus, in this approach, backtracking can occur only at the primary inputs.

The results achieved in the PODEM algorithm can be further improved, since there still remained many possibilities of reducing the number of backtracks in the algorithm. Such an improvement has been the FAN algorithm [13], [14]. The fanout-oriented test-generation algorithm FAN traces out the structure of the network, in order to find the nonexistence of the solution as soon as possible. Also, logic implication is widely involved in the procedure.

On the other hand, the acceleration results in the PODEM and FAN algorithms are relied on the use of the gate-level network structure [12]. However, if functional level modelling 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 modelling. The same is applied to the network types, i.e., combinational or sequential, and also, the types and multiplicity of the faults.

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 the 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 specific state or an output pattern [8], [10]. For this purpose, high level hardware-description languages, such as VHDL, can also be applied [9].

The computerized implementation of the proposed principle will handle multi-valued logic, where the number of logic values corresponds to the number of the possible signal values in the HDL applied for modelling [8], [9].

The complexity of a logic module is not limited in principle. Only practical trade-offs, such as processing time, and user efforts in terms of modelling accuracy are to be considered.

Acknowledgement

The author wishes to thank Dr. András Pataricza for his valuable advice in preparing the present paper.

References

[1] NAVABI, Z.: VHDL: Analysis and Modeling of Digital Systems, McGraw-Hill, Inc., USA, 1993.
[2] UBAR, R.: Test Generation for Digital Systems Based on Alternative Graphs, Dependable Computing - EDCC-1 (Ed. by D. Echtle, D. Hammer and D. Powell), pp. 151-164, Springer-Verlag, Berlin, 1994.
[3] ROTH, J. P.: Diagnosis of Automata Failures: a Calculation and a Method, IBM Journal of Research and Development, Vol. 10, pp. 278–291, July 1966.
[4] BREUER, M. A. – FRIEDMAN, A. D.: Diagnosis and Reliable Design of Digital Systems, Computer Science Press, USA, 1976.
[5] SZIRAY, J.: Test Calculation for Logic Networks by Composite Justification, *Digital Processes*, Vol. 5, No. 1–2, pp. 3–15, 1979.
[6] SZIRAY, J.: 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.
[7] JAIN, S. K. – AGRAWAL, V. D.: Modeling and Test Generation Algorithms for MOS Circuits, *IEEE Trans. on Computers*, Vol. C–34, pp. 426–433, May 1985.
[8] SZIRAY, J. – NAGY, ZS.: OPART: A Hardware-Description Language for Test Generation, Microprocessing and Microprogramming, (Ed. by P. Nunez), pp. 525–530, North-Holland Publishing Company, Amsterdam, 1991.
[9] SALLAY, B. – TILLY, K. – PATARICZA, A. – CSERTÁN, G. – HEGEDÜS, Z. – PETRI, A. – SURJÁN, L. – SZIRAY, J.: A Proposed VHDL Subset for Test Generation Purposes, Technical Report FUTEG-1/1994, PECO 9624 Project, Budapest, 1994.
[10] BREUER, M. A. – FRIEDMAN, A. D.: Functional Level Primitives in Test Generation, *IEEE Trans. on Computers*, Vol. C–29, pp. 223–235, March 1980.
[11] GOEL, P.: An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits, *IEEE Trans. on Computers*, Vol. C-30, pp. 215–222, March 1981.
[12] ABRAMOVICI, M. – BREUER, M. A. – FRIEDMAN, A. D.: Digital Systems Testing and Testable Design, Computer Science Press, USA, 1990.
[13] FUJIWARA, H. – SHIMONO T.: On the Acceleration of Test Generation Algorithms, *IEEE Trans. on Computers*, Vol. C–32, pp. 1137–1144, December 1983.
[14] FUJIWARA, H.: Logic Testing and Design for Testability, The MIT Press, USA, 1985.