Design Error Diagnosis in Digital Circuits with Stuck-at Fault Model

A. Jutman, R. Ubar

Tallinn Technical University, Computer Engineering Department
Raja 15, 12618, Tallinn, Estonia. Fax: (+372) 620 22 53
artur@pld.ttu.ee, raiub@pld.ttu.ee

Abstract. *In this paper we describe in detail a new method for the single gate-level design error diagnosis in combinational circuits. Distinctive features of the method are hierarchical approach (the localizing procedure starts at the macro level and finishes at the gate level), use of stuck-at fault model (it is mapped into design error domain only in the end), and design error diagnostic procedure that uses only test patterns generated by conventional gate-level stuck-at fault test pattern generators (ATPG). No special diagnostic tests are used because they are much more time consuming. Binary decision diagrams (BDD) are exploited for representing and localizing stuck-at faults on the higher signal path level. On the basis of detected faulty signal paths, suspected stuck-at faults at gate inputs are calculated, and then mapped into suspected design error(s). This method is enhanced compared to our previous work. It is applicable to redundant circuits and allows using incomplete tests for error diagnosis. Experimental data on ISCAS benchmark circuits shows the advantage of the proposed method compared to the known algorithms of design error diagnosis.*

1. INTRODUCTION

The VLSI design process is getting more and more complicated due to continuously growing sizes and complexity of the projects. Due to this fact the error emergence probability is big enough on different design stages. An error not detected at the earlier stages may cause a cost explosion of the project and significantly increase the time-to-market. Design verification and error localization, in their turn, become more and more time consuming tasks.

Most design error diagnosis approaches are either simulation-based [1, 3, 5, 7, 10, 11] or symbolic [2, 4, 8, 9, 12]. Simulation based approaches rely on a number of test vectors that differentiate the implementation and the specification. The potential error region is reduced according to simulation results of these vectors. The *symbolic* approaches do not rely on test vectors. They usually use Ordered Binary Decision Diagrams (OBDD) [14] to characterize the necessary and sufficient condition of a potential error source as a Boolean formula. The symbolic approaches are more accurate than simulation-based, however the explosion of the complexity for some classes of circuits puts practical limitations to the use of BDDs in locating design errors.

In [4, 6, 8, 15] structural approaches for design error rectification are proposed. Such approaches mostly apply verification techniques to narrow down the potential error region in the implementation. The techniques applied to search error(s) in this remaining area may be various. In [15] a heuristic called *back-substitution* is employed in hopes of fixing the error incrementally. In [4] a symbolic re-synthesis approach is used. Although these approaches are suitable for large circuits, their major drawback is that the success of the whole procedure is highly dependent on existence of structural similarity between two circuits.

In [16] and [17] general ideas and basic theoretical concepts of a new hierarchical design error diagnosis method are proposed, which is free from the restriction of the

structural approaches described above. The specification can be represented on any level of abstraction; it can be given as a truth table, as a BDD, as another gate-level circuit, or as a Boolean formula. The implementation is represented by structural BDDs (SSBDD) [22], which are generated directly from gate-level netlists. The success of the method does not depend on any structural similarity between the specification and the implementation. The idea of this new approach is refined and experimental results are presented in [18, 19].

This paper presents a further enhancement of the method that allows to extend it so that it is able to diagnose design errors using incomplete tests and that it is also applicable to redundant circuits.

The method is based on the stuck-at fault model, where all the analysis and reasoning are carried out in terms of stuck-at faults and only in the end the result of diagnosis will be mapped into the design error area. Such a treatment allows to exploit traditional ATPGs to serve the problem of design error diagnosis.

In contrast to BDD or OBDD circuit representations the complexity of SSBDD representation used in this approach grows not exponentially but only linearly [22]. In addition, SSBDD representation preserves structural information about a circuit allowing to develop fault diagnosis procedures that are more efficient for increasing the speed in error detection and localization than gate-level ones. On SSBDDs, a primary set of suspected faulty signal paths is calculated. Based on this set, a list of suspected erroneous gates is generated, which will be subsequently reduced to the minimum by using the information obtained from the test experiment.

Our approach combines both verification and error localization techniques. The information gathered on the *verification* stage is used during *localization* reducing the time needed for the whole process.

Due to the facts enumerated above, our method allows to increase the speed of design error diagnosis significantly in comparison to the known methods.

In [13] a widely adopted simple design error model is given. It includes two basic error categories: functional errors of gate elements and connection errors of signal lines. Gate errors, in their turn, are divided into gate substitution, extra/missing gate, and extra/missing inverter errors, and connection errors into extra and missing line errors. However in [7] it is shown that simple extra gate errors could be rectified using only gate substitution error model. Thus in this work, we consider only single extra/missing inverter and gate replacement error types. So, missing gate error and connection types of errors remain out of the scope of this paper. They, as well as multiple errors, are the subject of our future work.

Section 2 of the paper presents necessary definitions and terminology. The use of stuck-at faults and mapping of the diagnosis results into the design error domain are explained in Section 3. Representation of faults and signal paths on macro and gate levels is discussed in Section 4. Fault tables and vector representation of faults are described in Section 5. The algorithm for design error diagnosis is presented in Section 6 and an example of using is given in Section 7. Section 8 describes some limitations and possible improvements for the method. Experimental data is discussed in Section 9, and finally, Section 10 presents some conclusions.

2. DEFINITIONS AND TERMINOLOGY

Consider a circuit specification, and its implementation. The way of specification representation is not significant. Only a relationship between input patterns and output responses in the specification is important. However, without loss of generality, let the specification and the implementation be given at the Boolean level. The specification output is given by a set of variables $W = \{w_1 , w_2 , ... , w_m\}$, and the implementation output is given by a set of variables $Y = \{y_1 , y_2 , ... , y_m\}$, where m is the number of outputs. Let $X = \{x_1, x_2, ... , x_n\}$ be a set of input variables. The implementation is a gate network and Z is a set of internal variables used for connection of gates. Let S be the set of variables in the implementation $S = Y \cup Z \cup X$. The gates implement simple Boolean functions AND, NAND, OR, NOR, and NOT. An additional gate type FAN is added (one input, two or more outputs) to model fanout points (Fig. 1). It is not used in the diagnostic procedure but it is needed in several definitions below.

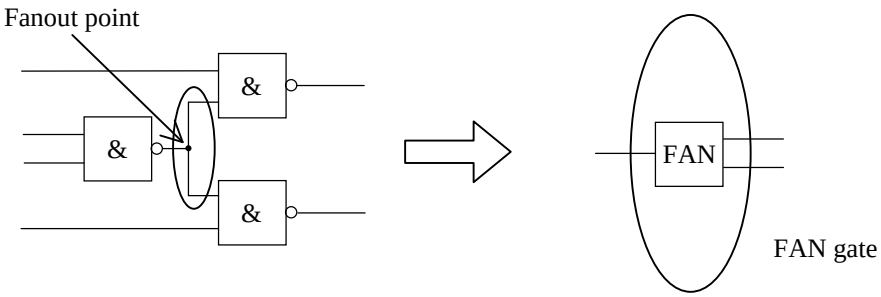


Fig. 1. ***A FAN gate***

We use two different levels for representing the implementation: gate and macro-level representations.

Let X^F and Z^F be subsets of inputs and internal variables that fanout (they are inputs to FAN gates). Let Z^{FG} be a subset of internal variables that are outputs of FAN gates. At the gate level, the network is described by a set $NG = \{g_k\}$ of gate functions $s_k = g_k(s_k^1, s_k^2, ... ,s_k^h)$ where $s_k \in Y \cup Z$, and $s_k^j \in (Z - Z^F) \cup (X - X^F)$. At the macro-level, the network is given by a set $NF=\{f_k\}$ of macro functions $s_k = f_k(s_k^1, s_k^2, ..., s_k^p)$ in an equivalent parenthesis form (EPF) [21], where $s_k \in Y \cup Z^F$, and $S_k = \{s_k^1, s_k^2, ..., s_k^p\} \subseteq Z^{FG} \cup (X - X^F)$ are its set of inputs. In other words, a macro is a tree-structured subcircuit with no fanout points inside. It has several inputs and only one output (Fig. 2).

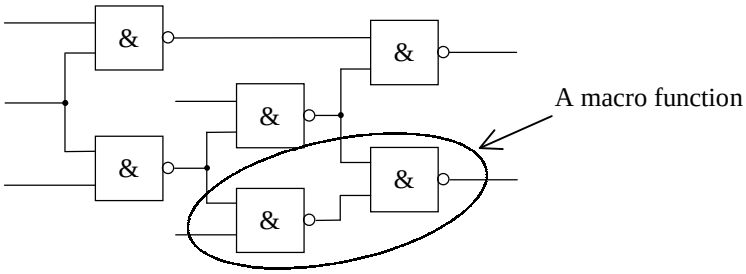


Fig. 2. ***An example of a macro function***

The following design error types are considered throughout the paper in relation to gates $g_k \in NG$.

Definition 1. *Gate replacement error*. It denotes a design error, which can be corrected by replacing the gate g_i in NG with another gate g_j, by $g_i \rightarrow g_j$. The "$\rightarrow$" sign refers to "is replaced by".

Definition 2. *Extra/missing inverter error*. It denotes a design error, which can be corrected by removing/inserting an inverter at some input $s \in X$, or at some fanout branch $s \in Z^{FG}$: $s \rightarrow NOT(s)$.

Definition 3. *Single simple error hypothesis*. Our design error diagnosis methodology is based on a single error hypothesis, which assumes that only one error of the following types can be present in a circuit: 1) an extra/missing inverter, 2) a random gate replacement between AND, OR, NAND, and NOR gates.

It should be clear from these definitions that only inverters that are occasionally inserted/omitted at a gate input or at a primary input of a circuit are treated as extra/missing inverters. Inverter inserted/omitted at a gate output is covered by gate replacement error type (Fig. 3).

Fig. 3. ***Extra inverter and gate replacement errors***

Definition 4. *Test patterns*. For a circuit with n inputs, a test pattern T'_i is an n-bit vector which may be binary BN^n or ternary TN^n, where BN = {0,1} - the Boolean domain, TN = {0,1,U} - the ternary domain, where U is a don't care. Let a *test* be a set of all test patterns applied to a circuit during one test experiment as $T = \{T_1, T_2, ... T_t\}$, where t is a number of test patterns.

Definition 5. *Stuck-at fault set*. Let F be the set of stuck-at faults s/1 and s/0, where $s \in Z \cup X$.

Definition 6. *Detectable stuck-at faults*. A test pattern T_i detects a stuck-at-e fault s/e, $e \in \{0,1\}$ at the output y_j, if applying the test pattern T_i to the implementation and the specification, the result $y_j(T_i) \neq w_j(T_i)$ is observed. Let us call s/e a detectable by a test pattern T_i at the y_j output stuck-at fault. We denote the detection information as $\varepsilon \in \{0,1\}$, where

$$\varepsilon(T_i, y_j) = \begin{cases} 0, \text{ if the test } T_i \text{ passed at } y_j \\ 1, \text{ if the test } T_i \text{ detected an error at } y_j \end{cases}$$

and as $\delta \in \{X,0,1,D\}$, where

$$\delta(T_i, y_j) = \begin{cases} 0, \text{stuck-at 0 (s/0) is } \textit{detectable} \text{ by } T_i \text{ at } y_j \\ 1, \text{stuck-at 1 (s/1) is } \textit{detectable} \text{ by } T_i \text{ at } y_j \\ X, \text{no fault is } \textit{detectable} \text{ by } T_i \text{ at } y_j \end{cases}$$

If both types of stuck-at faults are detectable somewhere by a set of test patterns, then δ=D in this place.

Denote a set of stuck-at faults detectable by a test pattern T_i at an output y_j as $F(T_i, y_j)$, then let

$$F(T_i) = \bigcup_{y_j \in Y} F(T_i, y_j)$$

be a set of faults detectable by a test pattern T_i, and

$$F(T) = \bigcup_{T_i \in T} F(T_i)$$

be a set of faults detectable by all the test patterns $T_i \in T$. A test T is complete iff F(T)=F. The method already proposed by us in [18, 19] was based on an assumption that a test is complete. This paper presents a modification of the previous method adapted to incomplete tests.

Definition 7. *Set of failing test patterns*. Let $E = \{ T_i \mid \exists j: T_i \rightarrow y_j(T_i) \neq w_j(T_i) \} \subseteq T$ be a set of failing test patterns.

3. Mapping Stuck-at Faults into Design Error Model

The stuck-at fault model does not have a physical meaning in this paper. It is only used to produce, as in the case of traditional testing, a diagnosis in terms of stuck-at faults, which is then mapped into design error domain. The method of mapping follows from the proof of the following theorem given in [16] and [17].

Theorem 1. To detect a design error in the implementation at an arbitrary gate g_k where $s_k = g_k(s_1, s_2, ..., s_h)$, it is sufficient to apply a pair of test patterns which detect the stuck-at faults s_i /1 and s_i /0 at one of the gate inputs s_i, i = 1,2, ... h.

From the proof the following set of corollaries was driven which describes the mapping from a stuck-at fault set to the design error model domain:

- localizing both the s/1 and s/0 faults on two or more gate inputs refers to the missing/extra inverter at the gate output, i.e. to the replacement errors: AND $\leftrightarrow$ NAND and OR $\leftrightarrow$ NOR;
- localizing s/1 faults at one or more gate inputs refers to the replacement errors: AND $\rightarrow$ OR, OR $\rightarrow$ NAND, NAND $\rightarrow$ NOR, and NOR $\rightarrow$ AND;
- localizing s/0 faults at one or more gate inputs refers to the replacement errors: AND $\rightarrow$ NOR, OR $\rightarrow$ AND, NAND $\rightarrow$ OR, and NOR $\rightarrow$ NAND;
- localizing both the s/1 and s/0 faults at one of the gate inputs s_i refers to the error $s_i \rightarrow$ NOT(s_i) at this input;
- localizing both the s-1 and s-0 faults at more than one branch of a primary input $s_i \in X^F$ refers to the error $s_i \rightarrow$ NOT(s_i) at this input.

The mapping is summarized in Table 1. Suspected erroneous gates in implementation are given in the first column. Next several columns refer to stuck-at faults detected at the gate inputs s_1, s_2, …, s_h. The last column represents the correction needed to rectify the circuit.

Gate	Stuck-at faults							Correction
	s_1		s_2		…	s_h		
AND	0	1	0	1	…	0	1	NAND
		1		1	…		1	OR
	0		0		…	0		NOR
	0	1			…			NOT(x_1)
			0	1	…			NOT(x_2)
					…	0	1	NOT(x_h)
OR	0	1	0	1	…	0	1	NOR
	0		0		…	0		AND
		1		1	…		1	NAND
	0	1			…			NOT(x_1)
			0	1	…			NOT(x_2)
					…	0	1	NOT(x_h)
NAND	0	1	0	1	…	0	1	AND
	0		0		…	0		OR
		1		1	…		1	NOR
	0	1			…			NOT(x_1)
			0	1	…			NOT(x_2)
					…	0	1	NOT(x_h)
NOR	0	1	0	1	…	0	1	OR
		1		1	…		1	AND
	0		0		…	0		NAND
	0	1			…			NOT(x_1)
			0	1	…			NOT(x_2)
					…	0	1	NOT(x_h)

TABLE 1. ***Mapping stuck-at faults into design error domain***

4. MACROS, PATHS, AND STUCK-AT FAULT MODELING

The following fault modeling method was developed for macro-level test generation based on usage of structurally synthesized BDDs (SSBDD) as the model for tree-like subcircuits or macros [20, 21, 22]. Every macro is represented by its own SSBDD.

Definition 8. *Signal paths*. We denote $L(s_k^j)$ as a set of variables on a path from the input of the macro $s_k^j \in S_k$ to its output s_k.

As macros are trees, there exists a one-to-one correspondence between inputs $s_k^j \in S_k$ and the gate-level signal paths $L(s_k^j)$ in the macro. The literal s_k^j in the EPF is an inverted (not inverted) variable if the number of inverters on the path from s_k^j to s_k is odd (even).

Definition 9. *SSBDD*. A SSBDD is a directed noncyclic graph G_k with a single root node and a set of nonterminal nodes M_k labeled by (inverted/not inverted) Boolean variables (arguments of the function or inputs of the tree-like subcircuit). Every node has exactly two successor-nodes, whereas terminal nodes are labeled by constants 0 or 1. The set M_k represents a macro f_k so that one-to-one correspondence exists between the nodes $m \in M_k$ and signal paths L(s) where $s \in S_k$. Let s(m) denote the literal at the node m in the graph G_k.

We also use another definition of signal paths throughout the paper: $L(m_k) = (g_k^1, g_k^2, g_k^3, \ldots, g_k^r)$ is a path corresponding to the node m_k where g_k^j is a gate on the path and there is a relation of order $R(g_k^a, g_k^b)$, $a,b \in [1,r]$ where from $a<b$ results that g_k^b stands closer to the output of the macro than g_k^a.

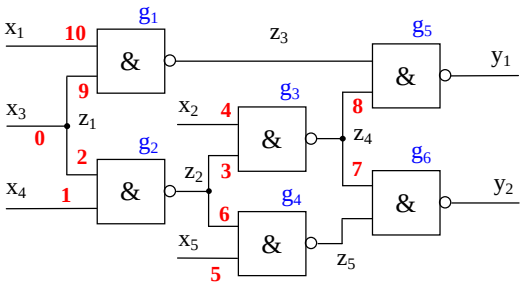


Fig. 4. ***Combinational circuit***

Example 1. Consider the combinational circuit from Fig. 4. It is a small circuit "c17" of ISCAS'85 benchmarks [23]. So, by previous definitions, $X=\{x_1, x_2, x_3, x_4, x_5\}$ is a set of input variables, $Y=\{y_1, y_2\}$ is a set of outputs, $Z=\{z_1, z_2, z_3, z_4, z_5\}$ represents internal variables, and $NG=\{g_1, g_2, g_3, g_4, g_5, g_6\}$ is a set of gates. The nodes of all SSBDDs are denoted in Fig. 4 by numbers 0 to 10 for nodes from m_0 to m_{10} correspondingly. The procedure of formal synthesis of SSBDDs from gate-level networks is based on a graph superposition and described in detail in [20, 22]. In Fig. 5, a short example of macro-level SSBDD construction procedure is given. The example refers to the macro consisting of g_4, and g_6 gates.

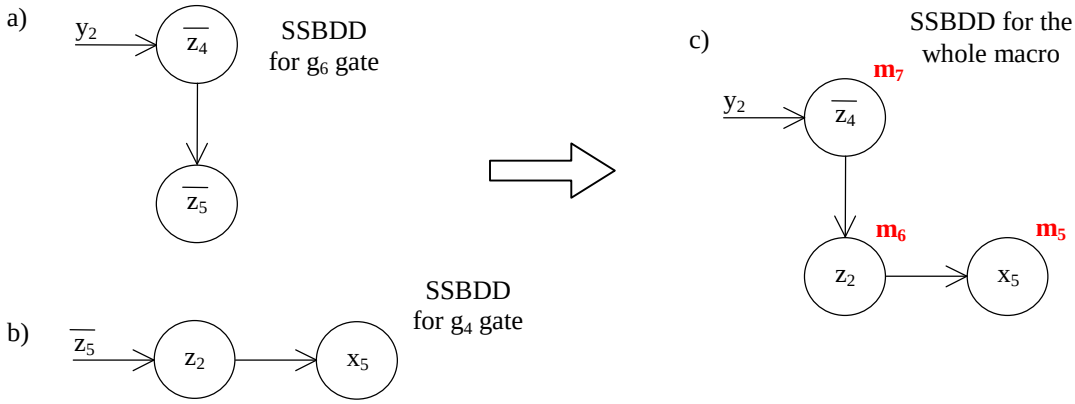


Fig. 5. ***Macro-level SSBDD construction***

Consider the SSBDD corresponding to internal variable $\bar{z}_5$ (Fig. 5(b)) where z_2 and x_5 are variables in nonterminal nodes, and $\bar{z}_5$ is the root node. Terminal nodes labeled by constants 0 and 1 are not shown. However, if one goes down from z_2 or x_5, one gets to the node labeled "0" that means $\bar{z}_5 = 0$. If one goes right from x_5, one gets to the node labeled "1" ($\bar{z}_5 = 1$). If the variable in a nonterminal node equals 0, one goes down. If it is 1, one goes to the right. For example, if z_2=1 and x_5=0, one gets to the terminal node labeled "0" ($\bar{z}_5 = 0$).

Such SSBDDs are constructed for each gate in the macro (Fig. 5). Then the superposition starts from the macro output and ends when the inputs of the macro are reached. In Fig. 5, it is shown that the node corresponding to internal variable $\bar{z}_5$ in Fig. 5(a) is substituted by the SSBDD corresponding to $\bar{z}_5$ (Fig. 5(b)) to form the SSBDD for the whole macro (Fig. 5(c)).

The following table shows how the nodes m_k in SSBDD macros are related to gate-level paths $L(m_k)$ in the circuit.

Macro	1	2		3		4			5		
m_k	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}
$L(m_k)$	$\varnothing$	g_2	g_2	g_3	g_3	g_4,g_6	g_4,g_6	g_6	g_5	g_1,g_5	g_1,g_5

TABLE 2. ***SSBDD nodes and corresponding gate paths***

Definition 10. *Faults on signal paths, fault class*. Let $F(s_k^j/e)$ where $e \in \{0,1\}$ be a set of all faults (a fault class) in the gate-level signal path from the input s_k^j of the macro S_k to its output s_k.

A fault s_k^j/e can be regarded as the representative of a fault class $F(s_k^j/e)$, since it is enough to test only the fault s_k^j/e to test all the faults $F(s_k^j/e)$.

If a fault $\delta(s(m_k))$ is detected in an SSBDD node m_k then a class of detected faults $F(\delta(s(m_k)))$ results at the gate level.

5. FAULT TABLES AND VECTORS

T_i	$s(m_k)$					E
	$s(m_1)$	$s(m_2)$	$s(m_3)$	...	$s(m_q)$	
T_1	δ_{11}	δ_{12}	δ_{13}	...	δ_{1q}	ε_1
T_2	δ_{21}	δ_{22}	δ_{23}	...	δ_{2q}	ε_2
T_3	δ_{31}	δ_{32}	δ_{33}	...	δ_{3q}	ε_3
⋮	⋮	⋮	⋮		⋮	⋮
T_t	δ_{t1}	δ_{t2}	δ_{t3}	...	δ_{tq}	ε_t

TABLE 3. ***A fault table***

Let us represent the set of detectable on the macro level faults F(T) as a *fault table* (Table 3) where row i corresponds to a test T_i and column k corresponds to a variable

$s(m_k)$ in the node m_k and $\delta_{ik} \in \{0,1,X\}$ shows what kind of fault is detectable at m_k by a test pattern T_i.

Each row of this table (δ_{i1}, δ_{i2}, δ_{i3}, … , δ_{iq}) represents $F(T_i)$ – the set of faults detectable by a test pattern T_i. The last column (E) represents a set of failing test patterns, i.e. shows the results of a test experiment, whether a test detected an error or not.

Since we also need to know sets of faults detectable by a particular test pattern at each primary output y_j, we have to construct such fault tables for each output separately.

In this case (Fig. 6), each row ($\delta(T_i, y_j, s(m_1))$, $\delta(T_i, y_j, s(m_2))$, $\delta(T_i, y_j, s(m_3))$, … $\delta(T_i, y_j, s(m_q))$) of a table j represents $F(T_i, y_j)$ – the set of faults detectable by a test pattern T_i at a primary output y_j.

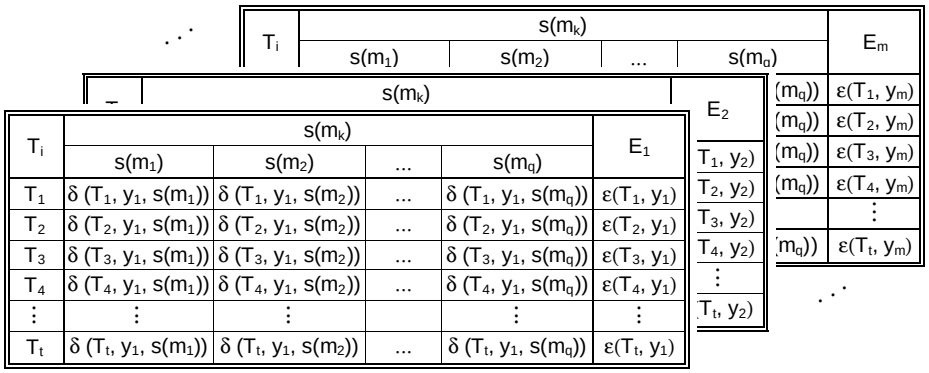


Fig. 6. ***Tables of faults detectable at each output***

The last column in each table shows whether a test pattern detected an error at a particular output, or not.

It is not hard to notice that each row of Table 3 as well as rows of tables in Fig. 6 can be represented in a vector form.

Definition 11. *Vector of detectable faults*. Let us denote a vector of faults detectable at the macro level by a test pattern T_i as $V_M^d(T_i) = (\delta_{i1}, \delta_{i2}, \delta_{i3}, \dots, \delta_{iq})$ and let $V_M^d(T_i, y_j) = (\delta(T_i, y_j, s(m_1)), \delta(T_i, y_j, s(m_2)), \delta(T_i, y_j, s(m_3)), \dots \delta(T_i, y_j, s(m_q)))$ be a vector of faults detectable at the macro level at a particular output y_j by a test pattern T_i.

Definition 12. At the macro level, let $V_M^c = (\delta(s(m_1)), \delta(s(m_2)), \delta(s(m_3)), \dots, \delta(s(m_q)))$ be a vector of faults guaranteed not to be present in a circuit on account of a set of failing test patterns E and a set of passed test patterns T-E. In other words, they are the faults that cannot be present in the circuit, all the other faults can be regarded to suspected or possible faults.

Let us define some operations with these vectors. We will need four types of operations: intersection ($\cap$), union ($\cup$), subtraction (–) and inversion ($\bar{\delta}$) (Fig. 7). All these operations are performed element-wise when applied to a pair of vectors.

Note that the definition of the inversion may seem a little bit confusing, but it becomes clear if we say that we use it propagating stuck-at faults along the signal path. If a gate on the path is inverting (NOT, NOR, NAND) then the detected stuck-at fault at the input of the gate is inverted at the output. If no faults are detected (X) at

the input, there is nothing to invert and no faults are detected at the gate output as well. Similarly, if both types of faults (D) are detected at the gate input, the separate inversions of each fault give us again both types of faults (D) detected at the output.

	$\delta_1 \cap \delta_2$				$\delta_1 \cup \delta_2$				$\delta_1 - \delta_2$				$\overline{\delta}$	
δ_1 \ δ_2	X	0	1	D	X	0	1	D	X	0	1	D	δ	$\overline{\delta}$
X	X	X	X	X	X	0	1	D	X	X	X	X	X	X
0	X	0	X	0	0	0	D	D	0	X	0	X	0	1
1	X	X	1	1	1	D	1	D	1	1	X	X	1	0
D	X	0	1	D	D	D	D	D	D	1	0	X	D	D

Fig. 7. ***Operations with fault vectors***

6. ALGORITHM DESCRIPTION

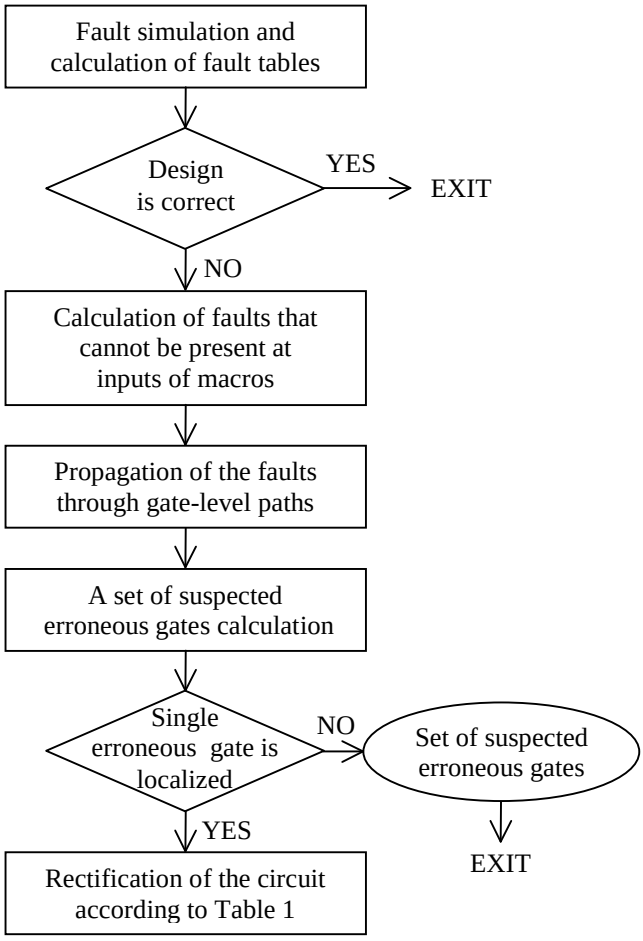


Fig. 8. ***Overall diagnostic algorithm***

The diagnostic procedure we described in [18, 19] is modified in this paper in order to adapt it to incomplete tests and to be applicable to circuits with redundancy. The

proposed design error diagnosis algorithm is implemented and works on two different levels of abstraction: macro (SSBDD) and gate level. The algorithm itself is very simple. This is due to underlying SSBDD circuit representation and techniques based on it. The stuck-at fault model is used during almost the whole diagnostic process instead of design error model. It helps to avoid complicated techniques for error localization. The overall algorithm is presented in Fig. 8.

Firstly, during the stage of verification through fault simulation, fault tables are constructed (Fig. 6) for each output. Then the macro-level diagnosis is done. It is based on the information which vector passed at which output, which not. The result represents a vector of faults guaranteed not to be present in a circuit. This vector is computed using the following formula:

$$V_M^c = \left(\bigcup_{T_i \in E} \left[\bigcup_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) - \bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) \right] \right) \bigcup \left(\bigcup_{T_i \in E} \left[\bigcup_{y_j : \varepsilon(T_i, y_j)=0} V_M^d(T_i, y_j) \right] \right) \bigcup \left(\bigcup_{T_i \in T-E} V_M^d(T_i) \right) \quad (1)$$

This formula represents a modification of Theorem 1 from [19]. In contrast to [19], we do not calculate a set of suspected faults first and then subtract a set of faults that guaranteed not be present in a circuit. We suspect everything first and calculate only a vector of faults guaranteed not to be present in the circuit. This allows to use incomplete tests for diagnosis and to include the faults not covered by the test into the suspected area.

Every test pattern can either detect an error or not, and if it does, it can detect the fault at one or more outputs. Due to this fact, the formula (1) consists of three main parts. The first part of the formula (1)

$$\bigcup_{T_i \in E} \left[\bigcup_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) - \bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j) \right]$$

refers to outputs where the error has been detected (faulty outputs) and calculates detectable by a faulty test vector $T_i \in E$ faults that cannot be present in a circuit, as they are not included in an intersection of detectable faults by faulty outputs:

$$\bigcap_{y_j : \varepsilon(T_i, y_j)=1} V_M^d(T_i, y_j)$$

This is due to the fact that a single design error can be only present in this intersection. The second part of (1)

$$\bigcup_{T_i \in E} \left[\bigcup_{y_j : \varepsilon(T_i, y_j)=0} V_M^d(T_i, y_j) \right]$$

addresses the faulty vectors but correct outputs. It is a union of faults that cannot be present in a circuit, as they are detectable by a faulty vector $T_i \in E$ at a primary output y_j while this output is correct. The last portion of (1)

$$\bigcup_{T_i \in T-E} V_M^d(T_i)$$

is a union of all faults detectable by all test patterns $T_i \in$ T-E that has not detected an error at all.

The total union of these three parts is a set of all faults that cannot be present in a circuit. It means that the real error (or a combination of stuck-at faults corresponding to the real error) is located somewhere in the area that is a complement of V_M^c to the whole stuck-at fault set F. The size of this remaining area depends on diagnostic properties of test patterns, fault coverage, and presence of redundancy in a circuit.

The macro-level diagnosis is complete at this step. The final result on this stage is a vector V_M^c where each element $\delta(s(m_k))$ shows what kind of fault cannot be present in an SSBDD node m_k.

From this step gate-level diagnosis for localization of erroneous gate(s) begins. Since every node at the macro level corresponds to a certain path (a set of gates) $L(m_k)=(g_k^1, g_k^2, g_k^3, ..., g_k^r)$ on the gate level (see Table 2), we have to propagate a fault in the node through all the gates in the path inverting it each time we meet an inverting gate (NOT, NOR, or NAND).

Algorithm 1.

Let $s(m_k)$ be a variable in a node m_k, $L(m_k) = (g_k^1, g_k^2, g_k^3, ..., g_k^r)$ be a path corresponding to the node m_k, $\delta(s(m_k))$ be a fault in the SSBDD node m_k, and $\delta(g_k^j)$ be a fault at the output of a gate g_k^j. We propagate a fault through the path as follows:

1. $$\delta(g_k^1) = \begin{cases} \delta(s(m_k)), \text{if the gate } g_k^1 \text{ is either OR or AND} \\ \overline{\delta}(s(m_k)), \text{if the gate } g_k^1 \text{ is either NOT, NOR or NAND} \end{cases}$$

2. $$\delta(g_k^j) = \begin{cases} \delta(g_k^{j-1}), \text{if the gate } g_k^j \text{ is either OR or AND} \\ \overline{\delta}(g_k^{j-1}), \text{if the gate } g_k^j \text{ is either NOT, NOR or NAND} \end{cases}$$

3. Repeat step 2 while $2 \le j \le r$ ■

Using this algorithm we construct vectors of faults $V_G^c(m_k) = (\delta(g_1), \delta(g_2), \delta(g_3), \dots, \delta(g_p))$ for each path $L(m_k)$, where $\delta(g_i) = \{X,0,1,D\}$ is a fault that is guaranteed not to be present at the output of a gate $g_i \in$ NG and p is a number of gates in the whole circuit. Every $\delta(g_i)$ from $L(m_k)$ is calculated by the following rule:

$$\delta(g_i) = \begin{cases} X, \text{if } g_i \notin L(m_k) \\ \text{calculate by Algorithm 1, if } g_i \in L(m_k) \end{cases}$$

A union of all these vectors $V_G^c(m_k)$ gives us a vector of all stuck-at faults that cannot be present in a circuit:

$$V_G^c = \bigcup_{m_k \in M_k} V_G^c(m_k)$$

It means that the combination of stuck-at faults caused by a real design error is located among all other stuck-at faults (not covered by V_G^c).

Let $V_G^s = (\delta(g_1), \delta(g_2), \delta(g_3), \dots, \delta(g_p))$ is a vector of *suspected* faults at gate outputs for every gate $g_i \in$ NG in a circuit, where $\delta(g_i) = \{X,0,1,D\}$ is the fault suspected at the

output of a gate g_i. This vector calculation will be the final step of the gate-level diagnosis before stuck-at-fault-to-design-error mapping.

First of all, we initialize the vector V_G^s by placing $\delta(g_i) = D$ in every position of V_G^s. It means that we initially suspect every gate to be erroneous and therefore any combination of stuck-at faults is suspected at the output of every gate in a circuit. After that we update V_G^s subtracting V_G^c from it.

Now we know all stuck-at faults suspected at gate outputs and at inputs of every macro. Thus, we know combinations of suspected stuck-at faults at gate inputs and we can refer to Table 1 to determine how to rectify the circuit. However, if the set of erroneous gates contains more than one gate (the error has not been located exactly), the mapping cannot be performed.

7. EXAMPLE

Consider a test with 5 patterns that is applied to the inputs of the circuit in Fig. 4. Suppose now that test patterns T_1 and T_5 fail at both outputs that results in $E=\{T_1,T_5\}$ and $\varepsilon(T_1, y_1)= \varepsilon(T_1, y_2)= \varepsilon(T_5, y_1)= \varepsilon(T_5, y_2)=1$ while other $\varepsilon(T_i, y_j)=0$. The vectors of detectable faults $V_M^d(T_i, y_j)$ and error detection information for such a case are presented in the following fault tables (Table 4,5) for outputs y_1 and y_2 correspondingly.

T_i	$s(m_k)$											E_1
	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	
T_1	0	0	0	1	X	X	X	X	0	X	1	1
T_2	1	X	X	X	1	X	X	X	0	1	X	0
T_3	0	X	X	X	X	X	X	X	X	0	0	0
T_4	1	X	1	0	0	X	X	X	1	X	X	0
T_5	X	X	X	0	0	X	X	X	1	X	X	1

TABLE 4. ***Fault table for y_1***

T_i	$s(m_k)$											E_2
	m_0	m_1	m_2	m_3	m_4	m_5	m_6	m_7	m_8	m_9	m_{10}	
T_1	0	0	0	1	X	X	1	0	X	X	X	1
T_2	X	X	X	X	1	1	X	0	X	X	X	0
T_3	X	1	X	X	X	0	0	X	X	X	X	0
T_4	1	X	1	X	X	X	X	X	X	X	X	0
T_5	X	X	X	0	0	X	X	1	X	X	X	1

TABLE 5. ***Fault table for y_2***

From that, according to (1), we create a vector of faults that cannot be present in the circuit on the macro level (macro-level diagnosis). First, compute the part of (1) concerning fault detection information by outputs for each faulty vector:

$$\bigcup_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1, y_j) = (0,0,0,1,X,X,1,0,0,X,1)$$

$$\bigcap_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1, y_j) = (0,0,0,1,X,X,X,X,X,X,X)$$

$$\bigcup_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1, y_j) - \bigcap_{y_j:\varepsilon(T_1,y_j)=1} V_M^d(T_1, y_j) = (X,X,X,X,X,X,1,0,0,X,1)$$

$$\bigcup_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5, y_j) = (X,X,X,0,0,X,X,1,1,X,X)$$

$$\bigcap_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5, y_j) = (X,X,X,0,0,X,X,X,X,X,X)$$

$$\bigcup_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5, y_j) - \bigcap_{y_j:\varepsilon(T_5,y_j)=1} V_M^d(T_5, y_j) = (X,X,X,X,X,X,X,1,1,X,X)$$

$$\bigcup_{T_i \in E} \left[\bigcup_{y_j:\varepsilon(T_i,y_j)=1} V_M^d(T_i, y_j) - \bigcap_{y_j:\varepsilon(T_i,y_j)=1} V_M^d(T_i, y_j) \right] = (X,X,X,X,X,X,1,D,D,X,1)$$

The second part of (1) cannot be calculated because T_1 and T_2 vectors failed at each output. However, the third part of (1) can be successfully calculated as a union of all faults detectable by vectors corresponding to test patterns that have not detected an error:

$$\bigcup_{T_i \in T-E} V_M^d(T_i) = (D,1,1,0,D,D,0,0,D,D,0)$$

The total union gives us $V_M^c = (D,1,1,0,D,D,D,D,D,D,D)$. It provides information that both stuck-at 0 and stuck-at 1 cannot be present in node m_0, stuck-at 1 cannot be present in m_1 but stuck-at 0 is possible, and so on.

We will turn to gate-level diagnosis now. The best is the case when an exactly located erroneous gate is the result of the diagnosis, and there exists a combination of suspected stuck-at faults at the gate inputs that clearly defines the type of detected design error (Table 1). However, sometimes the final result represents just a set of suspected erroneous gates with some suspected stuck-at fault combinations at their inputs. The explanation of possible reasons for that is given in the next chapters.

At the gate-level diagnosis stage we use Algoritm 1 and Table 2 to find a vector $V_G^c(m_k)$ for every path $L(m_k)$. The union of them is:

$$V_G^c = (D,0,D,D,D,D)$$

Subtracting it from initial $V_G^s = (D,D,D,D,D,D)$ we have

$$V_G^s = (X,1,X,X,X,X)$$

It means that no stuck-at fault is suspected at g_1, g_3, g_4, g_5, and g_6 outputs and these gates are correct, and stuck-at 1 is suspected at g_2 output. As stuck-at 0 faults are suspected at nodes m_1 and m_2 that are inputs of the gate g_2, then according to Table 1 (the case of NAND gate) it refers to the design error NAND $\rightarrow$ OR. To correct the design, the NAND gate g_2 should be replaced by an OR gate.

8. LIMITATIONS AND IMPROVEMENT OF THE METHOD

The first thing that can be noticeable is that XOR and XNOR gate substitutions are not included in the error model. It is because some of such cases may not be detected at all even if we have 100% fault coverage. Theorem 1 says that it is sufficient to apply a pair of test patterns detecting stuck-at faults s_i /1 and s_i /0 at one of the gate inputs to detect a design error in the implementation at an arbitrary gate. This statement is not applicable to XOR and XNOR gates. However, for some cases, we can extend the mapping table (Table 1) for XOR and XNOR gates, too.

Consider an OR gate g_k where s_1, s_2,...,s_h are its inputs and s_k is its output. Let $T_{OR,0}$ = $\{s_i=1, \forall j, j\neq i: s_j=0\}$ be a test pattern used to test s_i /0 fault and $T_{OR,1}$ = $\{\forall i, i=1,2,\ldots,h: s_i=0\}$ – to test s_i /1 fault. Suppose there should be a XNOR gate in implementation instead of the OR gate g_k. In such case, both suck-at faults will be detected at s_i and, similarly, at each gate input. Such situation is referred to OR→NOR substitution in Table 1. So, extending our gate substitution model into XNOR gate cases makes it impossible to detect the exact substitution type. However, it gives us an opportunity to try several possible substitutions to identify the correct one. The worse case is when we can't detect a substitution error at all.

Consider the same OR gate and the same test vectors $T_{OR,0}$ and $T_{OR,1}$. Suppose now the correct gate is XOR. These test vectors can't discover any difference between the two gates. This means that the error will not be detected at all or it will be detected occasionally by some other test pattern that is generated to detect some other stuck-at faults. This leads to a situation where wrong place will be suspected to be erroneous.

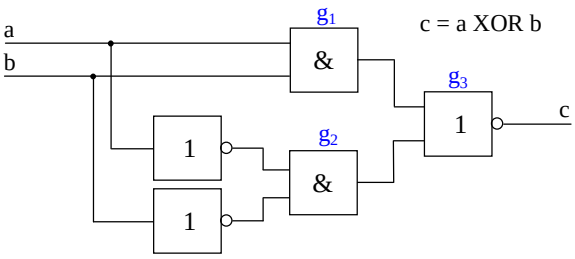
Consider an AND gate now. The test pattern pair for AND gate is $T_{AND,0}$ = $\{\forall i, i=1,2,\ldots,h: s_i=1\}$ to test s_i /0 fault and $T_{AND,1}$ = $\{ s_i=0, \forall j, j\neq i: s_j=1\}$ to test s_i /1 faults. Both vectors detect AND→XOR error if *h* is even or AND→XNOR if *h* is odd. However, AND→XOR error will not be detected at all if *h* is odd and AND→XNOR if *h* is even. Similar examples can be constructed for NOR and NAND gates too.

The situation described above restricts the substitution model of our method to four types of simple gates (AND, OR, NAND, NOR). However, it can be extended to some cases of XOR/XNOR gate substitution (Table 6).

Gate	Stuck-at faults							Correction
	s_1		s_2		…	s_h		
AND	0	1	0	1	…	0	1	even XOR, odd XNOR
OR	0	1	0	1	…	0	1	XNOR
NAND	0	1	0	1	…	0	1	odd XOR, even XNOR
NOR	0	1	0	1	…	0	1	XOR

TABLE 6. ***Mapping table extension***

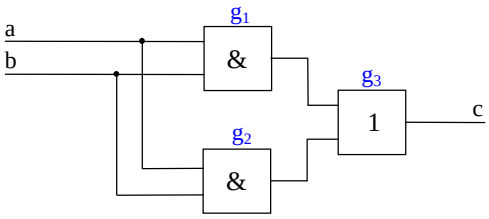
The cases opposite to discussed, namely when a gate is wrongly implemented as a XOR/XNOR, are more simple to detect and locate because every XOR/XNOR gate can be represented as a network of simpler gates and each simple gate can be tested separately. In [7], an example is given how a XOR gate replacement error can be rectified using only simple gate replacement model (Fig. 9). Thus, our model also covers a gate replacement by XOR/XNOR gates.



Replace g_3 by OR — to get c = a XNOR b
Replace g_1 by NOR — to get c = a OR b
Replace g_1 by NAND — to get c = a AND b
Replace g_2 by NOR — to get c = a NAND b
Replace g_2 by NAND — to get c = a NOR b

Fig. 9. ***Representation of a XOR gate [7]***

There is also a restriction on usage of our method if a design error causes redundancy. Consider a circuit in Fig. 10. Let the gate g_2 have NOR function but occasionally be substituted by an AND gate. The function of the whole circuit is AND now (c = a&b). The test for a/0 and b/0 is $T_{AND,0}$ = {1,1}, for a/1 is $T_{AND,a1}$ = {0,1}, and for b/1 is $T_{AND,b1}$ = {1,0}. Table in Fig.10 shows that this complete test is not capable to detect the given gate substitution error. Such design errors can only be detected occasionally by the test patterns that are intended to test other design errors, as it was described for OR→XOR case.



a	b	c=a&b	$c=(a\&b)\vee\overline{(a\vee b)}$
1	1	1	1
0	1	0	0
1	0	0	0

Fig. 10. ***Redundancy effect***

9. Results and Analysis of Experiments

The goals of the experiments described here were twofold:

- to compare the efficiency (the speed of fault localization) of the new diagnostic approach in comparison to previous results
- to evaluate the design error *diagnostic* properties of test patterns generated by traditional gate-level ATPGs for only stuck-at fault *detecting* purposes

Diagnostic experiments were carried out on ISCAS'85 benchmarks. Columns 2,3,4 in Table 7 give information about size of the benchmarks in terms of input, output and gate quantities. Note that the number of gates given in the table may be different from the real gate count because of XOR and XNOR gates representation by a number of simpler gates (Fig. 9).

Experiments were carried out in the following way. We took a netlist of an ISCAS'85 circuit and treated it as a wrong implementation. Then we generated SSBDD model

from it and created test patterns for detecting stuck-at faults. After that, using an error insertion tool, we created a "correct" specification and simulated it with the same test patterns to find the difference between output responses of the specification and the implementation. Using this information, the error analysis tool produced a diagnosis. Then, we took the initial "wrong" SSBDD again and created new "correct" specification with error insertion tool. After that the whole process was repeated. In other words, we had one "wrong" implementation and many "correct" specifications.

The fault coverage (column 6) of the test patterns created by the test generator described in [22] and the test generation time in seconds (column 11) are presented in Table 7. Experiments were carried out on the computer platform Sun SparcServer 20 (2 x Super Sparc II microprocessors, 75MHz) with Solaris 2.5.1 operating system.

The number of experiments carried out for each circuit are shown in column 5. For the cases marked by star (*), one random error for each gate per experiment was inserted. For the other cases, all possible single gate errors were simulated and analyzed – one error per experiment.

Circuit Name	Number of				Fault Cove-rage, %	Suspected Erroneous Gates				Time, s			Time for [7], s
						Number			Av. % of Total	Test Gene-ration	Fault Analysis (average)	Total	
	In-puts	Out-puts	Gates	Experi-ments		Min	Max	Av.					
1	2	3	4	5	6	7	8	9	10	11	12	13	14
c432	36	7	232	671	93,02	3	92	11,3	4,86	0,62	0,06	0,7	17,57
c499	41	32	618	1622	99,33	1	307	73,3	11,86	1,01	0,8	1,8	111,64
c880	60	26	357	1144	100	1	33	5,7	1,60	0,19	0,3	0,5	126,79
c1355	41	32	514	1830	99,51	1	248	55,2	10,74	1,35	0,9	2,3	241,79
c1908	33	25	718	1922	99,31	3	70	12,3	1,71	0,93	1,0	1,9	341,92
c2670	233	140	997	997*	94,97	69	218	91,1	9,14	3,55	8,5	12,1	661,91
c3540	50	22	1446	1446*	95,27	107	190	115,7	8,00	3,08	2,3	5,4	1513,82
c5315	178	123	1994	1994*	98,69	6	204	15,7	0,79	2,38	17,4	19,8	1814,04
c6288	32	32	2416	2416*	99,34	17	92	21,5	0,89	2,17	1,7	3,9	1895,90
c7552	207	108	2978	2978*	95,95	131	372	144,3	4,84	12,06	26,8	38,9	

TABLE 7. ***Diagnostic results for ISCAS '85 benchmark circuits***

The efficiency in the speed of diagnosis (columns 11, 12, 13) is compared to the results of [7] (column 14). However, it should be noted that the algorithm from [7] is implemented in Prolog language that is comparatively slow. The total time of diagnosis (column 13) in this work consists of two components: test generation time (column 11) and fault diagnosis (column 12).

The numbers in columns 7, 8, and 9 show, correspondingly, the minimal, maximal, and average diagnostic resolutions (numbers of suspected gates) reached by the tests. However, it should be noted that the diagnostic resolution of this method is not excellent for some circuits. The possible reasons are:

- the test patterns used for diagnosis were not originally generated for diagnostic purposes but just to *detect* an error
- redundancy, present in several circuits

To reach higher diagnostic resolution, additional test patterns should be generated. For this purpose, for example, the method of [7] can be used. Since the suspected area (column 10) for diagnostic search is reduced from 100% to from 0,79% (in the best

case) to 11,86% (in the worst case), the combination of the method proposed in the present work with some other method of additional diagnostic patterns generation, should reach significant improvements.

10. CONCLUSIONS

The method described in this paper has the following distinct features:

- The whole procedure takes place hierarchically at two different levels:
  - macro level, where the error detection and localization of the suspected erroneous SSBDD nodes are carried out
  - gate level for erroneous gate localization and exact specification of the design error

  Exploiting the hierarchy allows to combine the efficiency of working at the higher level (for error detecting) with the accuracy (needed for error diagnosis) at the lower level.
- Working with the stuck-at fault model allows to base on a single error hypothesis, which actually means working with several error hypothesis from design error model [13] in parallel.
- Compared to our previous work [18, 19] the method is extended in order to be capable to provide a solution even if the test is not complete and/or a portion of redundancy is present in a circuit.

The method has good chances for further improvement due to its high efficiency in speed and, therefore, the possibility of use in combination with some other diagnostic test generation and rectification techniques.

Experimental data is provided to demonstrate the efficiency of the method.

Our future research in this field is directed to improvement of the diagnostic resolution of the method. For this purpose additional test pattern generation heuristics can be worked out or several existing approaches can be used. It is also necessary to pay attention to the cases of multiple errors, complex gates, and line errors. The use of word level DDs seems to be very efficient in design error diagnosis at higher functional levels like RTL or behavioral ones.

Acknowledgements – This work has been supported by the COPERNICUS project No 977133 VILAB and by the Estonian Science Foundation grant G-1850. Authors appreciate the work of Jaan Raik for helping to carry out the experimental work.

REFERENCES

1. Veneris A, Venkataraman S, Hajj IN, Fuchs WK. Multiple Design Error Diagnosis and Correction in Digital VLSI Circuits. In: Proc. IEEE VLSI Test Symposium, April 1999, pp. 58B63.
2. Hoffmann DW, Kropf T. Using BDD-based Decomposition for Automatic Error Correction in Combinatorial Circuits. Technical Report, University of Karlsruhe, available at http://goethe.ira.uka.de/~hoff.

3. Huang SY, Cheng KT, Chen KC, Cheng DI. ErrorTracer: A Fault Simulation Based Approach to Design Error Diagnosis. In: Proc. International Test Conference, November 1997, pp. 974B981.
4. Huang SY, Chen KC, Cheng KT. Incremental Logic Rectification. In: Proc. VLSI Test Symposium, April 1997, pp. 134B139.
5. Wahba A, Borrione D. Connection errors location and correction in combinational circuits. In: Proc. European Design and Test Conference, ED&TC-97, Paris, France, March 1997.
6. Huang SY, Chen KC, Cheng KT. Error Correction Based on Verification Techniques. In: Proc. Design Automation Conference, June 1996, pp. 258B261.
7. Wahba A, Borrione D. A Method for Automatic Design Error Location and Correction in Combinational Logic Circuits. Journal of Electronic Testing: Theory and Applications 1996;8:113B127.
8. Lin CC, Chen KC, Chang SC, Marek-Sadowska M, Cheng KT. Logic Synthesis for Engineering Change. In: Proc. Design Automation Conference, June 1995, pp. 647B652.
9. Chung PY, Wang YM, Hajj IN. Logic design error diagnosis and correction. IEEE Transactions on VLSI Systems 1994;3:320B332.
10. Tomita M, Yamamoto T, Sumikawa F, Hirano K. Rectification of Multiple Logic Design Errors in Multiple Output Circuits. In: Proc. 31st Design Automation Conference, 1994, pp. 212B217.
11. Tamura KA. Locating Functional Errors in Logic Circuits. In: Proc. 26th Design Automation Conference, June 1989, pp. 185B191.
12. Madre JC, Coudert O, Billon JP. Automating the Diagnosis and the Rectification of Design Errors with PRIAM. In: Proc. ICCAD'89, 1989, pp. 30B33.
13. Abadir MS, Ferguson J, Kirkland TE. Logic Design Verification via Test Generation. IEEE Transactions on Computers, January 1988, pp. 138B148.
14. Bryant RE. Graph-based Algorithms for Boolean Function Manipulation. IEEE Transactions on Computers, 1986;C-35:667B691.
15. Brand D, Drumm A, Kundu S, Narrain P. Incremental Synthesis. In: Proc. International Conference on Computer Aided Design, 1994, pp. 14B18.
16. Ubar R, Borrione D. Localization of Single Gate Design Errors in Combinational Circuits by Diagnostic Information about Stuck-at Faults. In: Proc. 2nd Int. Workshop on Design and Diagnostics of Electronic Circuits and Systems, Szczyrk, Poland, September 1998, pp. 73B79.
17. Ubar R, Borrione D. Generation of Tests for Localization of Single Gate Design Errors in Combinational Circuits Using the Stuck-at Fault Model. In: Proc. 11th IEEE Brazilian Symposium on IC Design, Rio de Janeiro, Brazil, Sept. October 1998, pp. 51B54.
18. Ubar R, Jutman A. Hierarchical Design Error Diagnosis in Combinational Circuits by Stuck-at Fault Test Patterns. In: Proc. 6th International Conference Mixed Design of Integrated Circuits and Systems, Kraków, June 1999, pp. 437B442.

19. Jutman A, Ubar R. Design Error Localization in Digital Circuits by Stuck-At Fault Test Patterns. In: Proc. 22nd International Conference on Microelectronics, 19-22 Sept. 1999 (to appear).
20. Ubar R. Test Synthesis with Alternative Graphs. IEEE Design and Test of Computers, Spring, 1996, pp. 48B59.
21. Ubar R, Raik J. Multi-Valued Simulation with Binary Decision Diagrams. In: Proc. IEEE European Test Workshop, Cagliari, May 1997.
22. Raik J, Ubar R. Feasibility of Structurally Synthesized BDD Models for Test Generation. Proc. IEEE European Test Workshop, Barcelona, May 1998, pp.145B146.
23. Brglez F, Fujiwara H. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN. In: International Symposium on Circuits and Systems, Special Session on ATPG and Fault Simulation, 1985.