SINGLE GATE DESIGN ERROR DIAGNOSIS
IN COMBINATIONAL CIRCUITS

Raimund UBAR* and Dominique BORRIONE

Université Joseph Fourier/TIMA, 120 Rue de la Piscine, BP 53, 38041 Grenoble Cedex 9, France; raiub@pld.ttu.ee, Dominique.Borrione@imag.fr

Received 28 September 1998

Abstract. We propose a new approach to generate diagnostic tests and localize single gate design errors in combinational circuits. The method is based on using the stuck-at fault model with subsequent translation of the diagnosis into the design error area. This allows to exploit standard gate-level Automated Test Pattern Generators for verification and design error diagnosis purposes. A powerful hierarchical approach is proposed for generating test patterns which, at first, localize the faulty macro (tree-like subcircuit), and then localize the erroneous gate in the faulty macro. Experimental data show the efficiency of the macro-level test generation and fault simulation compared to the plain gate-level approaches.

Key words: design errors, stuck-at faults, test generation, error diagnosis, decision diagrams.

1. INTRODUCTION

As digital systems are becoming increasingly complex, design verification and design error localization are becoming more and more time consuming in the case of designs containing hundreds of thousands of gates as random logic. Verification and error localization are traditionally handled separately: for verification the methods of simulation and tautology checking can be used, whereas for error localization, after an error is detected, other dedicated methods are introduced [1,2].

While a lot of work has been done in the field of test synthesis and fault diagnosis in relation to fabrication faults, very little has been done in the field of design error diagnosis [1-5]. In [6], a new Binary Decision Diagrams (BDD) technique has been proposed. However, the explosion of the complexity for some classes of circuits puts practical limitations to the use of BDD's in locating

* On the leave from Tallinn Technical University, Raja 15, 12618 Tallinn, Estonia.

https://doi.org/10.3176/eng.1999.1.01
design errors. A brief overview of currently available solutions to the diagnosis problem has been given in [2].

The technique proposed in [1] assumes the existence of a single gate error in the combinational circuit. Simple gate errors are considered, and three error hypotheses have been introduced. The diagnoser works successively under one of these hypotheses. The reasoning is carried out at the plain gate level. A set of rules has been developed for all procedures with gates concerning the diagnostic reasoning as well as creation of activated paths through gates.

In the present paper, the same problem is formulated as in [1], i.e., a single design error case in combinational circuits is being attacked. Differently from [1] where the whole analysis is carried out at the plain gate level, in the present paper, a hierarchical approach is exploited which increases the speed of error detection and localization. Also differently from [1] where only the diagnosis problem is formulated and solved, in the present paper, the error detection and error diagnosis tasks are solved jointly which allows to increase the efficiency of error localization.

The originality of this paper lies in using structurally synthesized BDD's (SSBDD) [7,8] which allowed to develop efficient higher level path activation and fault reasoning procedures for increasing the speed in test generation and fault diagnosis. The method is based on the stuck-at fault model, where all the analysis and reasoning is carried out in terms of stuck-at faults and only in the end the result of diagnosis is mapped into the design error area. Such a treatment allows to exploit traditional Automated Test Pattern Generators (ATPG's) to serve the problem of design error diagnosis.

The paper is organized in the following manner. Section 2 presents the necessary definitions and terminology. The use of stuck-at faults and mapping the diagnosis results into the design error area is explained in Section 3. The model of SSBDD's is described in Section 4. The test generation technique for detecting design errors is presented in Section 5, and the error localization ideas are given in Section 6. Efficiency of the approach is considered in Section 7, and Section 8 presents our conclusions.

2. DEFINITIONS AND TERMINOLOGY

Consider a circuit specification, and its implementation, both 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 the set of input variables. The implementation is a gate network and \( Z \) is the set of internal variables used for the connection of gates. The gates are implementing simple Boolean functions AND, OR, NAND, NOR, and NOT. An additional gate type FAN is added (one input, two or more outputs) to model fanout points.
We use two different levels for representing the network: the gate and macro-level representations. Let $S$ be the set of variables in the implementation $S = Y \cup Z \cup X$. Let $X_F$ and $Z_F$ be the subsets of inputs and internal variables that fanout (they are input to a FAN gate). Let $Z_{FG}$ be the subset of internal variables that are output of a FAN gate. Then at the gate level, the network can be 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 $s_k^j \in Z \cup (X - X_F)$. Let us introduce macro functions for representing tree-like subcircuits of the network. Then, at the macro-level, the network is given by a set $NF = \{f_k\}$ of macro functions $s_k = f_k(s_k^1, s_k^2, ..., s_k^p)$ where $s_k \in Y \cup Z_F$, and $s_k^j \in Z_{FG} \cup (X - X_F)$.

**Definition 2.1. Test patterns.** For a circuit with $n$ inputs, a test pattern is a $n$-bit vector which may be binary $B^n$ or ternary $T^n$, where $B = \{0,1\}$ is the Boolean domain, $T = \{0,1,U\}$ is the ternary domain, and $U$ is a don't care.

**Definition 2.2. Stuck-at fault set.** Let $F$ be the set of stuck-at-1 faults $s/1$ and stuck-at-0 faults $s/0$, where $s \in Z \cup X$. Detection of faults in $F$ is sufficient for stating that the circuit is stuck-at fault free.

**Definition 2.3. Detecting 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 when 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. Mathematically, a stuck-at-$e$ is detected on $s$ if: $T_i \Rightarrow (\partial y_j/\partial s = 1)$ & $(s = -e)$ where $s \in Z \cup X$, and $y \in Y$.

**Definition 2.4. Stuck-at fault cover.** The circuit is tested completely by a test $T = \{T_1, T_2, ..., T_t\}$ for stuck-at faults, if $T$ detects all the faults in $F$. The gate $g_k$ which implements the function $s_k = g_k(s_k^1, s_k^2, ..., s_k^h)$ is tested by $T$ for stuck-at faults, if $T$ detects both stuck-at-1 and stuck-at-0 faults at all the gate inputs $s_k^j$.

The stuck-at fault model does not have a physical meaning in this paper. In reality, a design error is detected at $y_j$ when under the application of a test $T_k$, the result $y_j(T_k) \neq w_j(T_k)$ is observed. Using the stuck-at fault model, we only imitate the traditional testing by comparing the behaviour of the implementation and the specification as a "golden device". From tests that have shown an error, we produce, as in the case of traditional testing, a diagnosis in terms of stuck-at faults, which are then mapped into design errors. The following design error types are considered throughout the paper in relation to gates $g_k \in NG$.

**Definition 2.5. 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 $g_i \rightarrow g_j$.

**Definition 2.6. Extra/missing invertor error.** It denotes a design error which can be corrected by removing/inserting an invertor at some input $s \in X$, or at some fanout branch $s \in Z_{FG}$: $s \rightarrow \overline{NOT(s)}$. 


Definition 2.7. Single error hypothesis. Our design error diagnosis methodology is based on a single error hypothesis where it is assumed that in the circuit a single error from the following error types can exist: 1) an extra/missing invertor, 2) an arbitrary gate replacement between AND, OR, NAND, and NOR gates.

3. MAPPING DETECTED STUCK-AT FAULTS INTO DESIGN ERRORS

Theorem 3.1. To detect a design error in the implementation at an arbitrary gate \( g_k \) where \( s_k = g_k(s_1, s_2, \ldots, s_h) \), 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 \( s_i \), \( i = 1, 2, \ldots, h \).

Proof. 1. Consider first the detection of AND ↔ OR errors. A necessary condition is

\[
(s_1 \land s_2 \land \ldots \land s_h) \oplus (s_1 \lor s_2 \lor \ldots \lor s_h) = 1.
\]

The possible solutions of this equation are

\[
s_i \land \neg s_j = 1, \text{ where } i, j = 1, 2, \ldots, h, \text{ and } i \neq j.
\]

Thus, if we set at least two inputs of a gate to complementary values, then the errors of types AND → OR and OR → AND will be detected at the output of the gate.

2. Consider the case of design errors related to the AND gate. Let us choose a test pattern

\[
T_{\text{AND},1} = \{s_i = 0, \forall j, j \neq i: s_j = 1\},
\]

which is one of solutions (2) of Eq. (1), and which detect also the stuck-at fault \( s_i/1 \) at the AND input. It is easy to see that the pattern \( T_{\text{AND},1} \) detects not only the error AND → OR, but also the error AND → NAND, and the errors of missing/extra invertors at the input \( s_i \).

Consider now the error AND → NOR. The necessary condition for detecting the error is

\[
(s_1 \land s_2 \land \ldots \land s_h) \oplus \neg (s_1 \lor s_2 \lor \ldots \lor s_h) = 1,
\]

which has two solutions:

\[
(s_1 \land s_2 \land \ldots \land s_h) = 1,
\]

\[
(\neg s_1 \land \neg s_2 \land \ldots \neg s_h) = 1.
\]

The solution (4) gives a test pattern

\[
T_{\text{AND},2} = \{\forall i, i = 1, 2, \ldots, h: s_i = 1\},
\]
which detects not only the design error AND → NOR, but also stuck-at faults $s_j/0$ at all of the AND inputs. It is easy to see that the pattern $T_{\text{AND},2}$ detects also all the errors of missing/extra invertor at the other AND gate inputs $s_j$ ($j \neq i$) which were not detected by the pattern $T_{\text{AND},1}$.

Hence, we have shown that the test patterns $T_{\text{AND},1}$ and $T_{\text{AND},2}$ which detect, correspondingly, a stuck-at fault $s_j/1$ and a stuck-at fault $s_j/0$ at least at one input $s_i$ of the gate, are sufficient for detecting all the design errors related to the replacement of AND by another gate and the invertor errors at all the inputs of the AND gate.

3. Consider now the case of design errors related to the OR gate. Let us choose a test pattern

$$T_{\text{OR},1} = \{ s_i = 1, \forall j, j \neq i: s_j = 0 \},$$

which is one of solutions (2) of Eq. (1), and which detects the stuck-at fault $s_j/0$ at the OR input. It is easy to see that the pattern $T_{\text{OR},1}$ detects not only the error OR → AND, but also the error OR → NOR and the errors of missing/extra invertors at the input $s_j$.

Consider now the error OR → NAND. The necessary condition for detecting this error is

$$(s_1 \lor s_2 \lor ... \lor s_h) \oplus \neg (s_1 \land s_2 \land ... \land s_h) = 1.$$  \hfill (6)

There are two solutions for this equation, which can be formulated as (4) and (5). The solution (5) gives a test pattern

$$T_{\text{OR},2} = \{ \forall i, i = 1, 2, ... , h: s_i = 0 \},$$

which detects not only the design error OR → NAND, but also stuck-at faults $s_j/1$ at all of the OR inputs. It is easy to see that the pattern $T_{\text{OR},2}$ detects also all the errors of missing/extra invertor at the other OR gate inputs $s_j$ ($j \neq i$) which were not detected by the pattern $T_{\text{OR},1}$.

Hence, we have shown that the test patterns $T_{\text{OR},1}$ and $T_{\text{OR},2}$ which detect, correspondingly, a stuck-at fault $s_j/0$ and a stuck-at fault $s_j/1$ at least at one input of the gate, are sufficient for detecting all the design errors related to the replacement of OR by another gate and all the invertor errors at the inputs of the OR gate.

4. In similar way as in points 1 and 2, we can show that the test patterns $T_{\text{AND},1}$ and $T_{\text{AND},2}$ which detect a stuck-at fault $s_j/1$ and a stuck-at fault $s_j/0$ at least at one input of the NAND gate, are sufficient for detecting all the replacements of a NAND by another gate, and the invertor errors at all the inputs of the NAND gate.

5. In the same way as in points 1 and 3, we can show that the test patterns $T_{\text{OR},1}$ and $T_{\text{OR},2}$, which detect a stuck-at fault $s_j/0$ and a stuck-at fault $s_j/1$ at least at one input of the NOR gate, are sufficient for detecting all the replacements of a NOR by another gate, and the invertor errors at all the inputs of the NOR gate.
From the proof of the Theorem 1, the following set of corollaries follows which describe the mapping from a stuck-at fault diagnosis to a design error diagnosis.

**Corollary 3.1.** Localizing both the stuck-at-1 and stuck-at-0 faults on two or more gate inputs refers to the missing/extra inverter at the output of the gate, i.e., to the replacement errors: \( \text{AND} \leftrightarrow \text{NAND} \) and \( \text{OR} \leftrightarrow \text{NOR} \).

**Corollary 3.2.** Localizing stuck-at-1 faults at one or more gate inputs refers to the replacement errors: \( \text{AND} \rightarrow \text{OR} \), \( \text{OR} \rightarrow \text{NAND} \), \( \text{NAND} \rightarrow \text{OR} \), and \( \text{NOR} \rightarrow \text{AND} \).

**Corollary 3.3.** Localizing stuck-at-0 faults at one or more gate inputs refers to the replacement errors: \( \text{AND} \rightarrow \text{NOR} \), \( \text{OR} \rightarrow \text{AND} \), \( \text{NAND} \rightarrow \text{OR} \), and \( \text{NOR} \rightarrow \text{NAND} \).

**Corollary 3.4.** Localizing both the stuck-at-1 and stuck-at-0 faults at one of the gate inputs \( s_i \) refers to the error \( s_i \rightarrow \text{NOT}(s_i) \) at this input.

**Corollary 3.5.** Localizing both the stuck-at-1 and stuck-at-0 faults at more than one branch of a primary input \( s_i \in X^p \) refers to the error \( s_i \rightarrow \text{NOT}(s_i) \) at this input.

**Example 3.1.** As a direct illustration of Theorem 1 and Corollaries 3.1–3.4, the mapping between localized stuck-at faults and design errors for 2-input gates is shown in Table 1.

<table>
<thead>
<tr>
<th>Gate</th>
<th>Stuck-at faults</th>
<th>Design error</th>
<th>Gate</th>
<th>Stuck-at faults</th>
<th>Design error</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_1 )</td>
<td>( s_2 )</td>
<td>( s_1 )</td>
</tr>
<tr>
<td>AND</td>
<td>0 0 1 1</td>
<td>NAND OR NOT(x₁) NOT(x₂)</td>
<td>0 1 0 1</td>
<td>AND OR NOT(x₁) NOT(x₂)</td>
<td>0 1 0 1</td>
</tr>
<tr>
<td>NOR</td>
<td>0 1 0 1</td>
<td>NOR AND NOT(x₁) NOT(x₂)</td>
<td>0 1 0 1</td>
<td>OR AND NOT(x₁) NOT(x₂)</td>
<td>0 1 0 1</td>
</tr>
</tbody>
</table>
4. REPRESENTING CIRCUITS BY MACROS AND SSBDD's

We now consider a method which was developed for macro-level test generation based on using SSBDD as the model for macros [7]. Test patterns are generated at the macro-level, however, the fault (and error) diagnosis is made at the gate-level. Therefore, a correspondence should be established to map the macro-level results back to the gate-level.

Consider a given implementation as a network of macros \( NF = \{ f_k \} \) where each macro is a tree-like subnetwork whose inputs \( s \in S_k \) are either primary inputs which are not fanouts, \( s \in X - X^F \), or branches of the fanout nodes of the network, \( s \in Z^{FG} \). The set of inputs is: \( S_k = \{ s_k^1, s_k^2, \ldots, s_k^P \} \subseteq (X - X^F) \cup Z^{FG} \). Each macro \( f_k \in NF \) implements a function \( s_k = f_k(s_k^1, s_k^2, \ldots, s_k^P) \), given in an equivalent parenthesis form (EPF) [8], where the arguments \( s_k^i \in S_k \) in EPF are considered as literals.

**Definition 4.1. Signal paths.** Let \( s_k = f_k(s_k^1, s_k^2, \ldots, s_k^P) \) be a macro implemented at the gate level, and \( S_k = \{ s_k^1, s_k^2, \ldots, s_k^P \} \) be its set of inputs. We denote \( L(s_k) \) the set of variables on a path from the input of the macro \( s_k^i \in S_k \) to its output \( s_k \). As macros are trees, there exists a one-to-one correspondence between inputs \( s_k^i \in S_k \) and the gate-level signal paths \( L(s_k) \) in the macro. The literal \( s_k^i \) in the EPF is an inverted (not inverted) variable if the number of invertors on the path from \( s_k^i \) to \( s_k \) is odd (even).

**Definition 4.2.** An SSBDD is a graph \( G_k = (M_k, \Gamma_k, S_k) \) with a set of nodes \( M_k \), which represents a macro \( f_k \) so that an one-to-one correspondence exists between the nodes \( m \in M_k \) and signal paths \( L(s) \) where \( s \in S_k \). The set of nodes \( M_k \) is partitioned into nonterminal nodes \( M_k^N \) and terminal nodes \( M_k^T \), \( M_k = M_k^N \cup M_k^T \). There is one initial node \( m_0 \in M_k^N \) and only two terminal nodes: \( M_k^T = \{ m_0^T, m_1^T \} \). The terminal nodes are labelled by constants 0 and 1, whereas the nodes \( m \in M_k^N \) are labelled by literals \( s \in S_k \). There is a mapping from the set of nodes of the SSBDD to the set of literals of the EPF: let \( s(m) \) denote the literal at the node \( m \). The mapping \( \Gamma_k(m, e) \) defines the successor of \( m \) for the value of \( s(m) = e, e \in \{0, 1\} \). Denote \( \Gamma_k(m, e) = m^e \). A test pattern \( T_i \) which assigns values to \( S_k \) defines a set of activated edges in \( G_k \). The edge between \( m \) and \( m^e \) is activated when \( s(m) = e \) for the pattern \( T_i \). Activated edges which connect nodes \( m_i \) and \( m_j \) make up an activated path in the graph (an ordered subset of nodes) \( l(m_i, m_j) \subseteq M_k \). A path \( l(m_0, m^T_e) \) is called fully activated path. An SSBDD \( G_k = (M_k, \Gamma_k, S_k) \) represents a gate-level network which implements the function \( s_k = f_k(s_k^1, s_k^2, \ldots, s_k^P) \) iff for each pattern \( T_i \), a full path \( l(m_0, m^T_e) \) in \( G_k \) is activated such that \( s_k = e \) [7,8].

The procedure of formal synthesis of SSBDD's from gate-level networks based on a graph superposition procedure is considered in [7,8].
Example 4.1. Consider a combinational circuit in Fig. 1. The circuit is partitioned into 6 macros $g_{20} = f_{20}(x_1, x_2, x_3, x_4, x_5)$, $g_{22} = f_{22}(x_5, x_6, x_7)$, $g_{25} = f_{25}(x_6, x_8, x_{10})$, $g_{33} = f_{33}(x_1, x_{13}, x_{20}, x_{25})$, $g_{34} = f_{34}(x_{11}, x_{22}, x_{25})$, and $g_{35} = f_{35}(x_7, x_9, x_{12}, x_{14}, x_{22})$ where each macro is a tree-like subcircuit.

The macros are represented by an SSBDD in Fig. 2, and the one-to-one correspondence between paths $L$ in the circuit and nodes $m$ in SSBDD's is given in Table 2.

![Fig. 1. Combinational circuit.](image1)

![Fig. 2. Decision diagrams for the circuit in Fig. 1.](image2)
5. Generating Test Pairs for Detecting Design Errors

Theorem 5.1. A node \( m \in M_k^N \) in the SSBDD \( G_k \) is tested for a fault \( s(m)/e \), \( e \in \{0,1\} \), by a test pattern \( T_i \) iff it activates the following three paths in the graph: \( l_1 = l(m_0, m) \), \( l_2 = l(m^0, m^T_0) \), \( l_3 = l(m^1, m^T_1) \), and \( s(m) = -e \).

The proof is given in [7].

If \( g_k \not\in Y \), then a path should be activated from \( s_k \) through other macros to some of the primary outputs of the network, using similar technique on SSBDD’s as for generating a test for a node \( m \in M_k^N \) in \( G_k \).

Theorem 5.2. If a test pair \( T_i = \{ T_{i,1}, T_{i,2} \} \) which detects both stuck-at faults \( s(m)/1 \) and \( s(m)/0 \) at the node \( m \in M_k^N \) in the SSBDD \( G_k \), does not show an error, then all the gates along the path \( L(s(m)) \) in the gate-level implementation are free from design errors.

Proof. From the definition of SSBDD’s, it follows that a node \( m \) in \( G_k \), labelled by a variable \( s(m) \), represents the signal path \( L(s(m)) \) in the circuit. In [7] it was shown that testing the faults \( s(m)/1 \) and \( s(m)/0 \) in \( G_k \) is equivalent to testing all the stuck-at faults along the path \( L(s(m)) \). In other words, if a test pair \( (T_1, T_2) \) which detects the stuck-at faults \( s(m)/1 \) and \( s(m)/0 \) at the node \( m \) in SSBDD \( G_k \) shows no error on the implementation outputs, it means that no stuck-at faults on the path \( L(s(m)) \) in the gate-level implementation can be present. In accordance with Theorem 3.1 and Corollaries 3.1–3.5, it also means that no design errors on the path \( L(s(m)) \) can be present.

Definition 5.1. Test pairs and symbolic test patterns. A pair of test patterns \( \{T_{i,1}, T_{i,2}\} \) which detects both stuck-at faults \( s(m)/1 \) and \( s(m)/0 \) at the node \( m \in M_k^N \) in the SSBDD \( G_k \), and which differs only in the value of the variable
\(s(m)\), is represented by a single symbolic test pattern \(T_i\) where to \(s(m)\) is assigned symbolic value \(D\). By assigning \(D = 0\), we obtain from \(T_i\) a pattern which tests \(s(m)/1\), and by assigning \(D = 1\), we obtain from \(T_i\) a pattern which tests \(s(m)/0\). The assignment \(s(m) = D\) in a symbolic test pattern \(T_i\) refers directly to the set of gates along the path \(L(s(m))\) which are under test. If \(T_i\) does not show an error, all the gates on this path are error free.

The symbolic test pattern \(T_i\), according to Theorem 5.1, has to activate the paths \(l_1\), \(l_2\), and \(l_3\) in such a way that the solution is independent of the value of \(s(m)\). Assigning now to \(s(m)\) in \(T_i\) a symbolic value \(D \in \{0,1\}\), the pattern \(T_i\) takes a symbolic form which in fact represents a pair of patterns \(T_i = \{T_{i,1}, T_{i,2}\}\).

Sometimes it is possible to merge in the same symbolic test pattern more than one test pair for testing simultaneously more than one path through different outputs in the circuit. In this case, we have to assign to all simultaneously tested variables \(s(m_j)\), which represent nonoverlapping paths \(L(s(m_j))\), different symbols \(D_j \in \{0,1\}\). Suppose, we have created a symbolic test pattern \(T_i\) for testing a path \(L(s(m_j))\) with \(s(m_j) = D_1\) and with several variables assigned by \(U\) (don't care). If there is a variable \(s(m_p) = U\) in \(T_i\), and it is possible to update \(T_i\) by assigning \(s(m_{n1}) = U\), \(s(m_{n2}) = D_2\), and by fixing other \(U\)'s with 0 or 1 (if needed), so that the modified \(T_i\) forms a new symbolic test (pair of patterns) for testing the variable \(s(m_p)\), then we can merge the initial and updated test pairs in the same symbolic pattern. In its final form this pattern has two symbolic assignments \(s(m_{j1}) = D_1\) and \(s(m_{j2}) = D_2\). The substitution of \(D_1\) and \(D_2\) by 0 and 1 can be made independently because, when considering the pair for \(s(m_{j1})\), we have \(s(m_{p2}) = U\), and vice versa. In this way, a single symbolic test pattern can still be implemented as a single pair of patterns, which, in fact, is acting as two test pairs with different testing targets in parallel. In the described way it may be possible to match in a single symbolic test pattern even more than two test pairs. Each of these pairs will test a signal path in the circuit, which is not overlapping with other tested paths. The number of paths which can be tested in parallel cannot exceed the number of the outputs of the circuit.

**Example 5.1.** Let us create a test pair for testing both stuck-at faults \(s(m_4)/0\) and \(s(m_4)/1\) at the node \(m_4\) in the graph \(G_{20}\) in Fig. 2. This corresponds to testing all the stuck-at faults along the path \(L(s(m_4))\) from the input \(s_4\) to the output \(s_{20}\) of the macro \(f_{20}\) in the circuit. To test both faults \(s(m_4)/0\) and \(s(m_4)/1\) in \(G_{20}\), by a symbolic pattern, we find the following assignments according to Theorem 5.1:

\[
\begin{align*}
l_1 &= (m_0, m_2, m_3, m_4) \rightarrow \{s_1 = 1, s_2 = 0, s_3 = 0\}, \\
l_2 &= (m^{T_0}, m^{T_0}) = \emptyset \text{ (no activation needed, the terminal node } m^{T_0} \text{ is already reached)}, \\
l_3 &= (m_5, m^{T_1}) \rightarrow \{s_5 = 1\}.
\end{align*}
\]
This gives us a test pair \( T = \{ s_1 = 1, s_2 = 0, s_3 = 0, s_4 = D, s_5 = 1 \} \) where all other input variables of the circuit have the value \( U \) - unassigned.

Since \( s_{20} \) is not an output variable, we activate a path from \( s_{20} \) to \( s_{33} \) through the macro \( f_{33} \) which is represented in Fig. 2 by \( G_{33} \). For that, we create a test pair for testing in \( G_{33} \) the node \( m_1 \), labelled by \( \neg s_{20,1} \), by activating in \( G_{33} \) the paths: \( l_1 = (m_0, m_1) \rightarrow \{ s_1 = 1, s_{20,1} = D \} \), \( l_2 = (m_2, m_1^T, 0) \rightarrow \{ s_{13} = 0 \} \), \( l_3 = \emptyset \). The updated test pattern for testing the faults at nodes \( m_4 \) in \( G_{20} \) and \( m_1 \) in \( G_{33} \) is \( T = \{ s_1 = 1, s_2 = 0, s_3 = 0, s_4 = D, s_5 = 1, s_{13} = 0 \} \). The activated paths and the tested nodes in \( G_{20} \) and \( G_{33} \) are shown in bold. The test pair tests all the faults along the paths \( L(s(m_4)) = (s_{15}, s_{17}, s_{20}) \) for \( G_{20} \) and \( L(s(m_1)) = (s_{24}, s_{28}, s_{33}) \) for \( G_{33} \) in the circuit. According to Theorem 5.2, if the test pair \( T \) will not show an error, the gates \( g_{15}, g_{17}, g_{20}, g_{24}, g_{28}, \) and \( g_{33} \) are error free. The tested path in the circuit in Fig. 1 is highlighted in bold.

**Definition 5.2.** Design error cover. The subset of gates for which at least one stuck-at-1 fault and one stuck-at-0 fault are detected by a test \( T = \{ T_1, T_2, \ldots, T_l \} \), is called the design error cover \( C(T) \). The subset of gates, for which at least one of stuck-at faults (either stuck-at-1 or stuck-at-0) is detected by a test \( T \), is denoted by \( NG(T) \). In general, \( C(T) \subseteq NG(T) \).

**Corollary 5.1.** The design error cover \( C(T_i) \) for a symbolic test pattern \( T_i \) created for testing both stuck-at faults of \( s(m) \) is equal to the subset of gates traversed by the path \( L(s(m)) \).

**Theorem 5.3.** If a test \( T \) for a given combinational circuit has the cover \( C(T) \) which includes all the gates of the circuit, \( C(T) = NG \), and the test \( T \) shows no error, then no single gate design errors are present in the circuit.

Theorem 5.3 is a direct consequence of Definition 5.1 and Theorem 5.2.

**Example 5.2.** In Table 3, two symbolic test patterns are depicted. The union of their test covers includes all the gates of the circuit. Since the circuit has more than one output, the activation of more than one path \( L(s(m)) \) in the circuit is possible. To differentiate simultaneously and independently activated paths, we use independent symbolic values \( D_1, D_2, D_3 \in \{ 0, 1 \} \). The diagnostic information about the test consisting in total of four patterns \( T = \{ T_{1,1}, T_{1,2}, T_{2,1}, T_{2,2} \} \) is shown in Table 4. The symbolic values \( D_i \) show which node in the SSBDD model and which gates in the circuit will be tested by both of the symbolic patterns, and at which outputs the responses are to be observed. For example, the first test pattern \( T_1 \) detects, by substituting \( D_1 \) with 0 and 1, the path \( L(s_{4}) = \{ g_{15}, g_{17}, g_{20} \} \) in the macro \( f_{30} \) and the path \( L(s_{20,1}) = \{ g_{24}, g_{28}, g_{33} \} \) in the macro \( f_{33} \), both through the output \( s_{33} \). The same pattern detects also, by substituting \( D_2 \) with 0 and 1, the path \( L(s_8) = \{ g_{21}, g_{25} \} \) in the macro \( f_{25} \) and the path \( L(s_{25,2}) = \{ g_{30}, g_{34} \} \) in the macro \( f_{34} \) through the output \( s_{34} \), and by substituting \( D_3 \) with 0 and 1, the path \( L(s_9) = \{ g_{23}, g_{27}, g_{32}, g_{35} \} \) in the macro \( f_{35} \) through the output \( s_{35} \).
The paths in the BDD model in Fig. 2, activated by the symbolic test pattern $T_1$, are highlighted by bold arrows. The tested nodes are bold and shaded for the pattern $T_1$, and shaded for the pattern $T_2$. The paths $L(s) = \{g_{15}, g_{17}, g_{20}\}$ in the macro $f_{20}$, and the path $L(s_{20}) = \{g_{24}, g_{28}, g_{33}\}$ in the macro $f_{33}$, tested by the pattern $T_1$ through the output $s_{33}$ (using the symbol $D_1$), are shown in the circuit in Fig. 1 in bold.

### 6. LOCATING DESIGN ERRORS BY TESTING STUCK-AT FAULTS

The main idea of the diagnosis procedure lies in the hierarchical approach: at the first stage, the localization of a faulty macro is carried out; second, in the faulty macro, the faulty node is determined, which is then mapped into a design error. For localizing the erroneous macro, we can use the diagnostic information from different primary outputs on which the erroneous macro has an influence. For localizing the erroneous gate in the faulty macro, we activate different paths through the macro which include or do not include the suspected faulty gate, and perform a reasoning.

**Definition 6.1. Activated macros.** Let us call

$$AM(T_i, y) = \{f_k \mid k: T_i \rightarrow \partial y/\partial s_k = 1, y \in Y\}$$

a set of macros activated by the test pattern $T_i$, so that there exists an activated path from the output $s_k$ of the macro $f_k$ up to the primary output $y \in Y$. 

---

**Table 3. Test patterns for detecting design errors**

<table>
<thead>
<tr>
<th>Test</th>
<th>Inputs</th>
<th>Macro outputs/Outputs</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_1$</td>
<td>1 2 3 4 5 6 7 8 9 10 11 12 13 14</td>
<td>20 22 25 33 34 35</td>
</tr>
<tr>
<td>$T_1$</td>
<td>1 0 0 D_1 1 1 1 D_2 D_3 0 0 0 0</td>
<td>D_1 1 D_2 -D_1 D_2 D_3</td>
</tr>
<tr>
<td>$T_2$</td>
<td>0 D_1 1 U 0 0 D_2 U U 1 1 U 1 U</td>
<td>D_1 -D_2 1 D_1 -D_2 U</td>
</tr>
</tbody>
</table>

---

**Table 4. Fault detecting information for the test $T_1$ in Table 3**

<table>
<thead>
<tr>
<th>Macro</th>
<th>$f_{20}$</th>
<th>$f_{22}$</th>
<th>$f_{25}$</th>
<th>$f_{33}$</th>
<th>$f_{34}$</th>
<th>$f_{35}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>BDD nodes tested</td>
<td>$m_4$</td>
<td>$m_1$</td>
<td>$m_2$</td>
<td>$m_1$</td>
<td>$m_3$</td>
<td>$m_0$</td>
</tr>
<tr>
<td>Variables tested</td>
<td>$s_4$</td>
<td>$s_{21}$</td>
<td>$-s_{71}$</td>
<td>$s_8$</td>
<td>$-s_{201}$</td>
<td>$s_{202}$</td>
</tr>
<tr>
<td>Gates tested, $L(s)$</td>
<td>15, 17, 20</td>
<td>16, 20</td>
<td>18, 19, 22</td>
<td>21, 25</td>
<td>24, 28, 33</td>
<td>29, 33</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$T_1$</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_1$</th>
<th>$D_2$</th>
<th>$D_3$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$T_2$</td>
<td>$D_1$</td>
<td>$D_2$</td>
<td>$D_1$</td>
<td>$D_1$</td>
<td>$D_2$</td>
</tr>
</tbody>
</table>
Definition 6.2. Suspected faulty macros. Let us denote $SM$ the set of macros which are suspected to be faulty, and $SM(T_i)$ the set of macros which are suspected to be faulty on the basis of the test $T_i$ which has shown an error.

Let $E(T_i)$ be the subset of primary outputs where an error has been detected by applying the test pattern $T_i$

$$E(T_i) = \{ y_j \in Y \mid y_j(T_i) \neq w_j(T_i) \} \subseteq Y.$$ 

Theorem 6.1. If a test pattern $T_i$ shows an error, the following set of suspected faulty macros results

$$SM(T_i) = \bigcap_{y \in E(T_i)} AM(T_i, y).$$

Proof. The proof results from the single error hypothesis. If an error has been detected at more than one output $y \in E(T_i) \subseteq Y$, then the single erroneous macro can belong only to the intersection of the sets of suspected faulty macros $AM(T_i, y)$ at erroneous outputs.

Based on Theorem 6.1, each failed test pattern $T_i$ in the diagnosis procedure will iteratively update the current set of suspected faulty macros:

$$SM = SM \cap SM(T_i).$$

When a result $|SM| = 1$ has been reached, a faulty macro $f_k \in SM$ is localized. Then, the procedure of localizing the faulty gate in $f_k$ can be started.

From Theorem 5.2, the following statement results.

Corollary 6.1. If a test pattern $T_i$, which shows an error, detects a fault at $s(m)$ in $f_k$, then the subset of gates $L(s(m))$ is suspected to contain the faulty gate.

Definition 6.3. Suspected faulty gates. Let $SG$ denote the set of gates which are suspected to be faulty, and $SG_k(T_i, s_k = e)$ the set of gates which are suspected to be faulty in the macro $s_k$ when the test $T_i$ has shown an error and the value of $s_k$ was $e \in \{0, 1\}$.

Theorem 6.2. Suppose, we have established a set of suspected faulty nodes $SG_k(s_k = e)$ in the macro $f_k \in SM$. If a test pattern $T_j$ with $SG_k(T_j, s_k = e)$ shows an error then $SG_k(s_k = e)$ is updated as

$$SG_k(s_k = e) := SG_k(s_k = e) \cap SG_k(T_j, s_k = e),$$

otherwise, if $T_j$ does not show an error, then

$$SG_k(s_k = e) := SG_k(s_k = e) - SG_k(T_j, s_k = e).$$

Proof. The proof of these statements follows from the single fault assumption. If there are two subsets of nodes under error suspicion then only the intersection of these subsets can include the error source, and the first statement follows.
Otherwise, if we know that one subset does not include the error source, and another non-disjoint subset includes the error, their common part should be taken as error free, from which the second statement follows.

**Diagnostic procedure.** The diagnostic procedure consists of three parts: error detection, erroneous macro localization, and the design error localization.

1. The error detection part of the procedure starts with the successive application of test pairs, generated so that each test pair covers as big an untested part of the circuit as possible. If no test pattern shows an error, the circuit, according to Theorem 5.2, is error free. After detecting an error by a test pattern $T_i$ we create, according to Theorem 6.1, a set of suspected faulty macros $SM(T_i)$.

2. If $|SM(T_i)| = 1$, then the design error is localized in the macro $f_k \in SM(T_i)$. Otherwise, if $|SM(T_i)| > 1$, we have to proceed with the macro-level fault diagnosis according to Theorem 6.1.

3. After localizing the faulty macro $f_k$, we start the design error diagnosis at the gate level according to Theorem 6.2. Different strategies can be used, based on the idea of getting as much information as possible from the additional tests being applied. A reasonable approach is to divide $SG_k(s_k = e)$ into two equal subsets, one being tested again, and the other one not. For this purpose, we need to find a node $m$ in the graph $G_k$, for which the intersection $SG_k(s_k = e) \cap L(s(m))$ is half of $SG_k(s_k = e)$. For that node a test pattern should be created with a restriction $s_k = e$. If $|SG_k(s_k = e)| = 1$ is reached, or if no node $m$ is found where $SG_k(s_k = e) \neq L(s(m))$, the procedure is terminated and the final reasoning about the design error is made on the basis of $SG_k(s_k = e)$. The final diagnosis about the design error is based on Theorem 3.1 and the Corollaries 3.1–3.5.

**Example 6.1.** Suppose an error was detected at the output $s_{34}$ when applying the test pattern $T_1$ (with $D_2 = 1$) from Example 5.2. The signal path in the circuit tested by $T_1$ through the output $s_{34}$ is highlighted in Fig. 3 by bold lines. The macro-level fault diagnosis procedure is illustrated on graphs $G_{34}$ and $G_{25}$ in Fig. 4. Since the error was detected at $s_{34}$, at first, the macro $f_{34}$ is suspected to be faulty. The activated path in $G_{34}$ traverses the nodes $m_0$ and $m_2$. It is easy to see that an erroneous change of the value of $\neg s_{11,2}$ at the node $m_2$ does not change the value of $s_{34}$. Hence, the variable $\neg s_{11,2}$ cannot be the reason of the erroneous output value, and the error source should be the node $m_0$ which represents a path through gates $g_{30}$ and $g_{34}$. On the other hand, this path has the origin at the output $s_{25,2}$ of the macro $f_{25}$. Simulating now the activated path in the graph $G_{25}$, we notice that the nodes $m_1$ and $m_2$ can also be the causes of the detected error. Both represent the path which traverses through the gates $g_{21}$ and $g_{25}$. Hence, the macro-level reasoning gives a subset $SM(T_{1,2}) = \{f_{25}, f_{34}\}$ of suspected faulty macros, and two subsets of suspected faulty gates in these macros: $SG_{25}(T_{1,2}, s_{25} = 1) = \{g_{21}, g_{25}\}$, $SG_{34}(T_{1,2}, s_{34} = 1) = \{g_{30}, g_{34}\}$. 

16
To localize the faulty macro at the macro level (according to Theorem 6.1), we construct a pattern which tests different subsets of currently suspected macros at different outputs. Let us test the suspected node \( m_1 \) in the macro \( f_{25} \) at the two primary outputs \( s_{33} \) and \( s_{34} \). To do so, we create a test pattern for testing \( m_4 \) (labelled by \( s_{25,1} \)) in \( G_{33} \) and \( m_0 \) (labelled by \( s_{25,2} \)) in \( G_{34} \). The activated paths in \( G_{33} \) and in \( G_{34} \) in Fig. 5a are shown by bold arrows. This gives us the test pattern \( T_3 = \{ s_1 = 0, s_{13} = 1, s_{20} = 1, s_{11} = 0, s_{25} = 1 \} \). For holding the suspected macro \( f_{25} \) at the same conditions as when the error was detected, we keep in \( T_3 \) the previous values \( s_{10} = 0, s_8 = 1 (D_2 = 1) \), and \( s_6 = 1 \) as in \( T_1 \). For \( T_3 \) we have \( AM(T_3, s_{33}) = \{ f_{25}, f_{33} \} \), \( AM(T_3, s_{34}) = \{ f_{25}, f_{34} \} \).

Fig. 3. Localizing the erroneous gate in the circuit.

Fig. 4. Macro level fault diagnosis in the case of detecting an error.
Suppose, the new test pattern shows an error at both outputs $s_{33}$ and $s_{34}$. From this, on the basis of Theorem 6.1, we find

$$SM(T_3) = AM(T_3, s_{33}) \cap AM(T_3, s_{34}) = \{f_{25}, f_{33}\} \cap \{f_{25}, f_{34}\} = \{f_{25}\},$$

which shows that the erroneous gate should be looked for in the macro $f_{25}$.

For localizing the gate error in $f_{25}$ we look for a node $m$ in $G_{25}$ so that $L(s(m)) \neq SG_{25}(T_{1,2}, s_{25} = 1) = \{g_{21}, g_{25}\}$. Such a node (see Table 1) is $m_0 : L(s(m_0)) = \{g_{23}\}$. To test the node $s(m_0)$ at the condition $s_{25} = 1$, we create the test pattern $T_4 = \{s_{10} = 1, s_8 = 0\}$ (see activated paths on $G_{25}$ in Fig. 5b). This pattern should be updated to observe the variable $s_{25}$ at a circuit output. To observe $s_{25}$ at $s_{33}$, we activate a path through $f_{33}$ by testing the node $m_4$ in $G_{33}$. This needs the following additional assignments: $s_1 = 0, s_{13} = 1, s_{20} = 1$. Suppose, the created test pattern does not show an error. Then on the basis of Theorem 6.2, we have

$$SG_{25}(s_{25} = 1) = SG_{25}(s_{25} = 1) - SG_{25}(T_4, s_{25} = e) = \{g_{21}, g_{25}\} - \{s_{25}\} = \{g_{21}\}.$$

Since the fault which was detected at the gate $g_{21}$ was stuck-at-0 (the fault $s_{9}/0$), then on the basis of the Corollary 3.3 the design error is AND$_{21}$ $\rightarrow$ NOR. The located erroneous gate is shaded in Fig. 3.

7. CONSIDERATIONS ON THE EFFICIENCY OF THE APPROACH AND EXPERIMENTAL DATA

The proposed diagnostic procedure consists of two parts – error detection and error diagnosis. Both parts are based on using the stuck-at-fault model and the hierarchical representation of random logic by SSBDD’s. The complexity of test generation (the number of needed tests) for error detection is determined by the possibility of covering all the gates of the circuit by as few paths as possible.

The upper bound of the number of tests $NT$ needed for fault detection in circuits with only one output (the most difficult situation) is
where \( NG \) is the number of gates. Two tests are needed for the case of a chain of gates, and the upper bound \( NG \) is needed for the two-level circuit. For the case of a tree of 2-input gates, the upper bound of the number of tests needed is \( \log_2 NG + 1 \).

In the general case of multi-output circuits the upper bound for the number of tests \( NT \) reduces drastically compared to the upper bound of \( NG \) in (7), because it is possible to test different faults in parallel over different outputs. Examples of the upper bounds of \( NT \) for ISCAS'85 circuits is given in Table 5. The efficiency of test generation when using SSBDD's is illustrated by the short time needed for test generation. The last column illustrates the efficiency of using SSBDD's when determining the faults detected by test patterns as the basic operation in fault diagnosis. A ratio of simulation speeds between using the gate level and macro level is given.

The method proposed in the paper has the following advantages compared to the previous work [1].

1. The design error detection and localization are combined and based on the same technique; this facilitates the use of the information about error free nodes, already obtained during the error detection procedure, for error localization.

2. The whole procedure takes place hierarchically at three different levels: macro level (for error detection and for localization of the erroneous macro), gate level (for localization of the node related to the site of the design error), and “stuck-at fault to design error mapping” level for exact specification of the design error. Exploiting the hierarchy allows a combination of the efficiency of working at the higher level (for error detection) with the accuracy (needed for error diagnosis) at the lower level.

### Table 5. Experimental data on test generation and fault simulation for ISCAS'85 benchmarks

<table>
<thead>
<tr>
<th>ISCAS circuit name</th>
<th>Number of faults</th>
<th>Fault cover, %</th>
<th>Number of test patterns</th>
<th>Number of compacted patterns</th>
<th>ATPG time, s</th>
<th>Fault simulation gate/macro speed ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>c432</td>
<td>616</td>
<td>97.33</td>
<td>89</td>
<td>55</td>
<td>0.10</td>
<td>3.86</td>
</tr>
<tr>
<td>c880</td>
<td>902</td>
<td>100.00</td>
<td>140</td>
<td>100</td>
<td>0.05</td>
<td>5.15</td>
</tr>
<tr>
<td>c1355</td>
<td>1552</td>
<td>99.64</td>
<td>70</td>
<td>52</td>
<td>0.24</td>
<td>3.08</td>
</tr>
<tr>
<td>c1908</td>
<td>1990</td>
<td>99.75</td>
<td>144</td>
<td>122</td>
<td>0.22</td>
<td>6.46</td>
</tr>
<tr>
<td>c2670</td>
<td>2692</td>
<td>96.67</td>
<td>160</td>
<td>119</td>
<td>0.55</td>
<td>6.47</td>
</tr>
<tr>
<td>c3540</td>
<td>3650</td>
<td>95.58</td>
<td>201</td>
<td>145</td>
<td>0.77</td>
<td>8.81</td>
</tr>
<tr>
<td>c5315</td>
<td>5770</td>
<td>99.78</td>
<td>178</td>
<td>108</td>
<td>0.57</td>
<td>8.37</td>
</tr>
<tr>
<td>c6288</td>
<td>7680</td>
<td>99.80</td>
<td>41</td>
<td>33</td>
<td>0.60</td>
<td>2.55</td>
</tr>
<tr>
<td>c7552</td>
<td>7924</td>
<td>99.46</td>
<td>276</td>
<td>198</td>
<td>2.71</td>
<td>9.04</td>
</tr>
</tbody>
</table>
3. Working with the stuck-at fault model on a single error hypothesis corresponds to working with all three hypothesis from [1] in parallel.

8. CONCLUSIONS

In this paper, a new approach has been presented to automatically diagnose single design errors in combinational circuits. The main original features of the method are: the hierarchical approach, based on using SSBDD’s, the use of very powerful error detection and fault localization procedures based on SSBDD’s and the idea of mapping stuck-at fault diagnosis into the final localization of the design error. The latter allows us to use the test patterns generated for stuck-at faults to produce design error diagnosis. Experimental data are provided for showing the efficiency of the error detection phase of the method. The efficiency of the second phase, error site localization, results from the drastically reduced area where the search for the faulty gate should be continued after error detection. The future research in this field is directed to the case of multiple design errors and to the case of complex gates. The use of word level decision diagrams seems to be very efficient in design error diagnosis at higher functional levels like register transfer levels or behavioural ones.

ACKNOWLEDGEMENTS

The authors appreciate the work of Jaan Raik, Eero Ivask, and Antti Markus from the Tallinn Technical University for carrying out the experiments.

REFERENCES

ÜKSIKVENTIILIDE DISAINIVIGADE DIAGNOOS KOMBINATSIOONSKEEMIDES

Raimund UBAR ja Dominique BORRIONE