Connection Error Location and Correction in Combinational Circuits

Ayman M. Wahba Dominique Borrione

Modélisation et Vérification des Systèmes Digitaux, TIMA Laboratory,
BP 53, 38041 Grenoble Cedex 9, France

Abstract

We present new diagnostic algorithms for localizing connection errors in combinational circuits. Three types of errors are considered: extra, missing, and bad connection errors. Special test patterns are generated to rapidly locate the error. The algorithms are integrated within the Prevail^{TM} system. Results on benchmarks show that the error is always located, within a time proportional to the product of the circuit size, and the number of used patterns.

1 Introduction

Design fault diagnosis plays an essential role in providing correct VLSI products. Despite the use of automated synthesis tools to produce *correct by construction* circuits, experience shows that a phase of design correction is necessary to obtain correct designs [1]. A design methodology often used is *construct by correction*. Initial designs made by automated tools are manually modified to improve some design aspects such as the performance or area requirements, or to carry on small specification changes. During this manual phase, design errors are very likely to be inserted.

Existing verification tools can discover the existence of design errors, and provide counter examples in the form of input patterns that witness a difference between the behaviors of the implementation and of the specification [2]. The task of finding and correcting the error is then left to the designer who makes use of the counter examples to simulate the design and reason about the nature and the location of the error. After correction the result is verified again, and the same diagnosis-correction-verification cycle is repeated until a correct design is obtained.

This lengthy manual diagnosis process may exceed the design time itself. Its automation drew the attention of researchers in the last few years, but till now little work is done in this domain. Abadir [3] classified simple design errors into: i) *gate errors* (missing/extra inverter, gate replaced by another function), and ii) *connection errors* (missing/extra or wrong connection). The diagnostic systems presented in [4, 5, 6, 7] deal only with gate errors. The systems presented in [8, 9, 10] deal also with connection errors. Some of these methods suffer from a lack of precision in locating the error, others use a large number of test patterns before localizing the error. Methods based on the use of BDD's failed on some benchmarks due to the exponential explosion of the BDD size.

We have implemented a prototype diagnostic system capable of identifying a great variety of single functional and connection errors, with a 100% hit ratio. In order to reduce combinational complexity, the system makes one error hypothesis at a time, and returns, for each hypothesis, either the empty set (no change according to this hypothesis can correct the implementation), or a set of proposed changes (figure 1). Our methods for *gate* error diagnosis in combinational and sequential circuits are published in [11, 12]. This paper focuses on the principles for the automatic identification of *connection errors*.

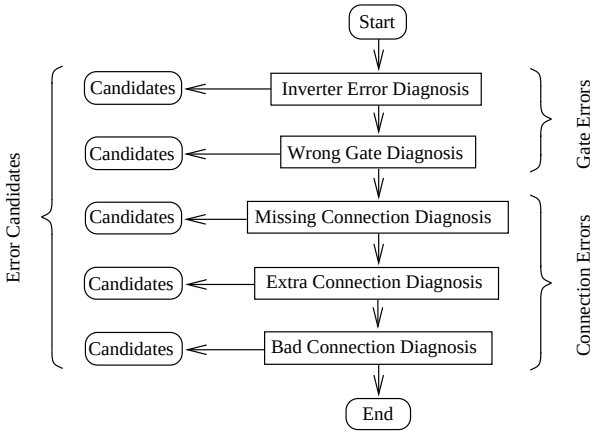

Figure 1: Diagnosis by error hypothesis

We improve upon other connection error diagnostic methods in three ways: i) we consider a larger class of connection errors that includes extra, missing, and bad connections; ii) our algorithms identify not only the gate at which the erroneous connection exists, but also which connection is the erroneous one; iii) using

ED&TC '97 on CD-ROM

diagnosis-oriented test patterns, the error localization is made using a small number of patterns (typically 10-20 patterns for a circuit of about 3500 gates).

The diagnostic system presented here is based on the close cooperation of three basic modules (figure 2): a test pattern generator, a simulator and a diagnoser. The pattern generator generates, for a given suspected gate, special input vectors capable of detecting the error if it exists at this gate. The simulator simulates the implementation and the specification under the application of these patterns. The simulation result is used by the diagnoser to limit the suspected area of the circuit. This information is passed to the pattern generator which selects another gate from the new suspected area, and the same operation is repeated until all suspected gates are processed or the suspected area is emptied.

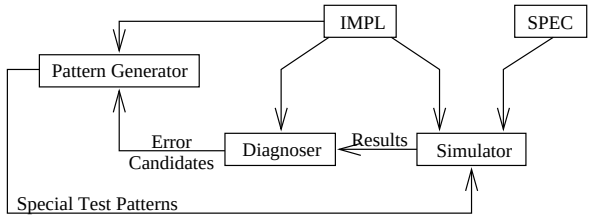

Figure 2: The overall diagnosis system

2 Basic definitions and terminology

Throughout this paper we consider a circuit specification *SPEC*, and its implementation *IMPL*, both at logic level. The specification output is denoted $W = \{w_1, w_2, ...w_m\}$, and the implementation output is denoted $Y = \{y_1, y_2, ...y_m\}$, where m is the number of outputs. The implementation is described as a gate network, while the description style for the specification is not restricted. A connection from the output of gate G_1 to the input of gate G_2 is denoted $C(G_1, G_2)$.

Definition 2.1 : The *search space* is a subset of all the circuit gates, and it contains any gate G_e at which extra/missing or bad connections may exist. $\diamond$

Initially, the search space contains all the implementation gates, and then it is reduced gradually by the diagnosis algorithm as test patterns are applied.

Definition 2.2 : For a circuit with n inputs, a *test pattern* is a n-bit vector which may be binary $\mathcal{B}^n$ or Ternary $\mathcal{T}^n$, where

$\mathcal{B}$: $\{0, 1\}$ - The Boolean Domain.

$\mathcal{T}$: $\{0, 1, X\}$ - The Ternary Domain.

X is a don't care value: the input pins with value X should have no influence on the output values.

Test patterns are generated so that they produce a binary value, (0 or 1) on one or more outputs of the *implementation* when they are applied to its inputs. Both the specification and the implementation are simulated under the application of the test patterns. We then distinguish between the patterns that detect the error and the ones that don't detect it.

A test pattern TP is called an Error Detecting Pattern *(EDP)*, or Non-Detecting Pattern *(NDP)*, with respect to a certain output y_i, according to the following rule:

$$TP \; is \begin{cases} EDP & if \;\; y_i \neq w_i \\ NDP & if \;\; y_i = w_i \end{cases}$$

In the following, whenever we refer to a gate, we always talk about a gate in the implementation (since the specification may be described functionally). The gate types considered are AND, OR, NAND, NOR and NOT. Other gate types (XOR, XNOR, complex gates) are represented as networks of the previous ones.

Error model:

Our model for connection errors is based on the study presented in [3] about simple design errors, and the problems submitted to us by the design engineers at Thomson-TCS. The model covers three types of connection errors (figure 3):

1. Missing connection at a gate input: A gate with $n-1$ inputs is used instead of a n inputs gate. All of the $n-1$ inputs are correctly connected.
2. Extra connection at a gate input: A gate with $n+1$ inputs is used instead of a n inputs gate. All of the n inputs are correctly connected, and the extra input is connected to an arbitrary node.
3. Bad connection at a gate input: one gate input is replaced by another input.

A basic assumption is that the circuit is combinational and that the error does not introduce loops in it (a separate pre-processing tool should check for cycles in the signal flowgraph).

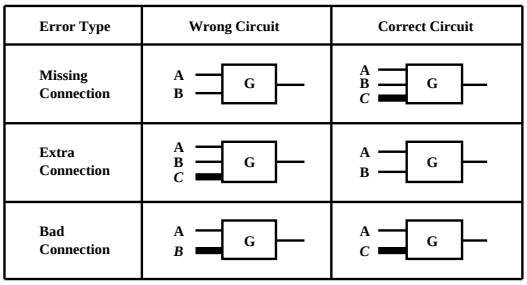

Figure 3: Connection errors

Definition 2.3 : $CV(G,TP)$ is the *current value at the output of a gate* G, when the test pattern TP is applied at the implementation primary inputs. $\diamond$

Definition 2.4 : Let $Y_m(G,V,TP)$ denote the value obtained at the output of the implementation, under the application of an input pattern TP, if the current value at the output of a gate G, $CV(G,TP)$, is replaced by another value V.
The *required value at the output of a gate* G, $RV(G,TP)$, under the application of an error detecting pattern TP, is the value which satisfies $Y_m(G,RV(G,TP),TP) = W(TP)$. $\diamond$

We introduce a new Boolean operator (*) to facilitate the description of the diagnosis algorithm. Its function is given by the table 1.

$b \backslash a$	1	0	x
1	x	0	0
0	1	x	1
x	1	0	x

Table 1: The (*) operator. a * b

Definition 2.5 : We define the *suspectability function* for a gate G, of type Type(G), when a test pattern TP is applied to the implementation inputs, as follows:

$$\boldsymbol{sus(G,TP)} = \begin{cases} RV(G,TP) * CV(G,TP) & if\ Type(G) = AND/NOR \\ not(RV(G,TP) * CV(G,TP)) & if\ Type(G) = OR/NAND \\ 0 & if\ Type(G) = NOT \end{cases}$$

For each gate type, there exists a boolean value of its output which is *forced* if any of the gate inputs is set to a given boolean value, regardless of the possible ternary values of the other inputs. This output value is called *forced value* and the corresponding input value is called *forcing value*. We denote these values *Forced(G)* and *Forcing(G)*. The following table shows these values for the different types of gates.

Gate type	*Forcing(G)*	*Forced(G)*
AND	0	0
OR	1	1
NAND	0	1
NOR	1	0
NOT	$v \in \{1,0\}$	$\overline{v}$

Table 2: *Forcing* and *Forced* values

Definition 2.6 : Let V_G be the value of the output of a gate G, $V_G \in \mathcal{T}$. A value $V_{comp}(G,V_G) \in \mathcal{T}$ is called *compatible* with V_G if it is necessary to set one or more inputs of G to the value $V_{comp}(G,V_G)$ to generate V_G at its output. For all gate types, we have:
$V_{comp}(G,V_G) = Forcing(G) \oplus Forced(G) \oplus V_G$
The set of inputs of G that have a value compatible with V_G is denoted $comp(G,V_G)$. $\diamond$

Definition 2.7 : A *changeable input* of a gate, under the application of an input pattern TP, is an input which, if its value is complemented, forces the output of G to take the required value $RV(G,P)$. $\diamond$

Definition 2.8 : A *fixed input* of a gate, under the application of an input pattern TP, is an input which, if its value is complemented, forces the output of G to take a new value V, $V \neq CV(G,P)$. $\diamond$

Definition 2.9 *A gate G_2 is in the* successor set *of another gate G_1,* successor(G_1), *if there is a path from the output of G_1 to one (or more) input of G_2.* $\diamond$

3 Diagnosis of missing connections

Proposition 3.1 *: Under the application of an error detecting pattern TP, a gate G in the search space of IMPL is suspected for having a missing connection at its inputs if and only if sus(G,TP) = 0.* $\diamond$

Proof: The proof of this proposition and all other popositions throughout the paper are omitted for brevity. Interested readers can refer to [13] for the complete proofs. $\diamond$

If a gate G in *IMPL* is suspected, under the application of an error detecting pattern TP, then we associate with it the set of nodes $P_m(G,TP)$, from which the missing connection may originate. By a node we mean any fanout stem.

Proposition 3.2 *: If a gate G in IMPL, under the application of an error detecting pattern TP, is suspected for having a missing connection, then:*

$$P_m(G,TP) = \{i \mid (i \in nodes\ of\ IMPL) \wedge value(i) = V_{comp}(G, RV(G,TP))\}$$

3.1 Analysis with non-detecting patterns

In many cases the applied test patterns don't detect the error, or detect the error on some primary outputs while the other outputs still have the correct values. Circuit analysis starting from those outputs having correct values can also help to exclude some suspected missing connections.

Proposition 3.3 *: If under the application of a Non-Detecting Pattern TP, the current value at the output of gate G, CV(G,TP), has to be fixed to keep*

the correct primary outputs unchanged, then the set of nodes $I_m(G,TP)$ that cannot be missing inputs for G is given by:

$$I_m(G,TP) = \{i \mid CV(G,TP) * (value(i) \oplus Forcing(G) \oplus Forced(G)) = \overline{Forced(G)}\}$$

3.2 The diagnosis algorithm

Based on the above propositions we present here a diagnostic algorithm for locating missing-connections errors. Given a test pattern TP, the algorithm scans the circuit from the primary outputs backward to the primary inputs. This is made by the recursive function *analyze-m-cnct* described below. Initially the *Search Space* contains all the gates of the circuit except for the inverters which cannot have missing connections. The algorithm *diagnose-m-cnct* is then executed repeatedly with different test patterns until the error is found or until no more test patterns can be generated. Pattern generation is presented in the next section.

```
algorithm diagnose-m-cnct(TP);
    apply TP to the inputs of the implementation IMPL;
    simulate IMPL;
    for every gate G_{y_i} driving a primary output y_i ≠ X do
        New Search Space = φ;
        if y_i ≠ w_i then Pat-Type := EDP;
        else Pat-Type := NDP; endif;
        analyze-m-cnct(Pat-Type,G_{y_i});
        if Pat-Type = EDP then
            Search Space = New Search Space; endif;
    endfor;
end algorithm.

procedure analyze-m-cnct(Pat-Type,G);
    if G is a primary input then exit; endif;
    if Pat-Type = EDP then
        if (G ∈ Search Space) ∧ (sus(G,TP) = 0) then
            % this if statement follows from proposition 3.1
            New Search Space = New Search Space ∪ {G};
            Nodes = P_m(G,TP);
            update-m-cnct(Pat-Type,G,Nodes);
        endif;
        for every input i ∈ Changeable-Inputs(G) do
            analyze-m-cnct(Pat-Type,i); endfor
    else
        Nodes = I_m(G,TP);
        update-m-cnct(Pat-Type,G,Nodes);
        for every input i ∈ Fixed-Inputs(G) do
            analyze-m-cnct(Pat-Type,i); endfor;
    endif;
end procedure.
```

If a gate G was already treated with other patterns, then it will have associated with it a set of possible missing connections $P_m(G)$, or a set of impossible missing connections $I_m(G)$. If the same gate is treated again under the application of a new test pattern TP, then it will have associated with it a new set $P_m(G,TP)$ or $I_m(G,TP)$. These new sets are used to update $P_m(G)$ and $I_m(G)$ by the procedure *update-m-cnct* described in the following.

```
procedure update-m-cnct(Pat-Type,G,Nodes);
begin
    if Pat-Type = EDP then
        % Nodes is a set of possible missing connection
        if G is already treated with an EDP then
            P_m(G) = P_m(G) ∩ Nodes;
        elseif G is already treated with an NDP then
            P_m(G) = Nodes - I_m(G);
        else    % The first time to treat G
            P_m(G) = Nodes;
        endif
    else        % Nodes is a set of impossible missing connection
        if G is already treated with an EDP then
            P_m(G) = P_m(G) - Nodes;
        elseif  G is already treated with an NDP then
            I_m(G) = I_m(G) + Nodes;
        else    % The first time to treat G
            I_m(G) = Nodes;
        endif
    endif;
end.
```

3.3 Test pattern generation

The diagnosis algorithm presented in the previous section can analyze the circuit under the application of any test pattern. However, to accelerate the diagnosis process, it is better to use error detecting patterns for the following reasons: each error detecting pattern TP_i specifies a set of gates SG_i at which extra connections may exist. If the error is detected under the application of n detecting patterns $TP_1, TP_2 \cdots TP_n$, then the error must exist in the intersection of $SG_1, SG_2, \cdots SG_n$. This intersection operation decreases rapidly the number of suspected locations, especially when the error is detected on different outputs in a multi-output circuit. On the other hand, non-error detecting patterns, can just exclude some suspected connection which may have already been considered suspected.

Error detecting patterns can be generated in many ways. The most direct way computes the function $YW(X) = Y(X) \oplus W(X)$ and finds the values of the input vector X that make $YW(X) = 1$. This computation can involve symbolic manipulation methods such as BDD's. The problem with BDD's is that their size may explode exponentially with some functions (e.g. integer multiplication functions), or when the input variables ordering is not well chosen. In our work we use the PODEM algorithm [14] to generate test patterns. Using this algorithm, we can generate a certain value at any node in the implementation, and propagate its effect to one or more primary output.

Proposition 3.4 *A test pattern TP detects whether a connection from a node i to the input of a gate G is*

missing, if when TP is applied to the inputs of IMPL, the value of i is set to Forcing(G), and all inputs of G are set to $\overline{Forcing(G)}$, and a path is sensitized from the output of G to at least one primary output. $\diamond$

In the prototype that we implemented, we start by applying a counter example which is an error detecting pattern supplied by the verifier or by any other means. This will limit the number of suspected gates and their possible missing connections. Then for each remaining suspected gate G with possible missing connections $n_1, n_2, \cdots n_m$, test patterns capable of detecting whether n_i ($i \in [1, m]$) is a missing connection or not, are generated and the diagnosis algorithm is applied to furtherly reduce the search space. The algorithm stops when the number of suspected gates is reduced to zero, which means that this error hypothesis is not correct, or when patterns are generated for all the suspected gates. The complete diagnostic algorithm is given here:

algorithm diagnose-m-cnct;
  Search Space = All the circuit gates - $\{g \mid Type(g) = \text{NOT}\}$;
  Tested $\leftarrow \phi$;
  while ($\mid$ *Search Space* $\mid > 1$) **and** (*Search Space* $\not\subseteq$ *Tested*) **do**
    Let G be a gate in *Search Space* nearest to primary inputs, and $G \notin Tested$
    Tested $\leftarrow$ *Tested* $\cup \{G\}$;
    for every suspected line i of G do
      TP = Pattern for detecting whether i is missing at G;
      % TP is generated following proposition 3.4
      diagnose-m-cnct(TP); **endfor**
  enddo
  for every $G \in$ *Search Space* **do**
    if $P_m(G) \neq \phi$ **then** write(G,$P_m(G)$); **endif**
  endfor
end algorithm.

4 Diagnosis of bad connection errors

Finding bad connections is more complex. Assume that an implementation contains n gates, and the average number of fan-ins of each gate is m. In the case of a missing connection, any one of the n gates may have a missing connection from any one of the other gates, so the size of the search space is $O(n^2)$. In the case of a bad connection error, any input of any one of the n gates may be replaced by a connection from any other gate. The size of the search space is thus $O(m \times n^2)$.

Proposition 4.1 *: Under the application of an error detecting pattern TP, a gate G in the Search Space of IMPL is suspected for having a bad connection if and only if one of the following three conditions holds:*

1. *sus(G,TP) = 0.*
2. *sus(G,TP) = 1 and RV(G,TP)* $\neq X$ *and*
   $\exists i, i \in inputs(G)$,
   $value(i) = V_{comp}(G, CV(G,TP)) \wedge$
   $\forall j \neq i, j \in inputs(G), value(j) = \overline{Forcing(G)}$
3. *sus(G,TP) = 1 and RV(G,TP) = X and*
   $\exists i, i \in inputs(G)$,
   $value(i) = V_{comp}(G, CV(G,TP)) \wedge$
   $\forall j \neq i, j \in inputs(G), value(j) \neq Forcing(G)$

In addition, if $Type(G)$ is NOT, then any $RV(G,TP)$ can be obtained by replacing the input of G by a connection from any node having the value $\overline{RV(G,TP)}$. So G is always suspected. This case is covered by condition 1 of the proposition. $\diamond$

If a gate G in *IMPL* is suspected under the application of an error detecting pattern TP, then we associate with it two sets of nodes:

1. $P_{bad}(G,TP)$: a subset of the inputs of G such that any one of them may be a bad connection, and must be replaced to correct the circuit output.
2. $P_{good}(G,TP)$: a subset of the nodes of *IMPL* such that any one of them may replace the bad connection to correct the implementation output.

Proposition 4.2 *: If a gate G in IMPL is suspected for having a bad connection, under the application of an error detecting pattern TP, then:*

$$P_{bad}(G,TP) = \begin{cases} \{i \mid i \in inputs(G)\} & if\ \ sus(G,TP) = 0 \\ \{i \mid i \in inputs(G) \wedge \\ \quad value(i) = V_{comp}(G, CV(G,TP))\} & Otherwise. \end{cases}$$

Proposition 4.3 *: Let G be a gate in IMPL, and let* $Nodes = \{i \mid i \in IMPL \wedge i \notin successor(G)\}$. *If under the application of an error detecting pattern TP, G is suspected for having a bad connection, then:*

if $RV(G,TP) \neq X$ ***then***
  $P_{good} = \{j \mid j \in Nodes \wedge value(j) = V_{comp}(G, RV(G,TP))\}$
elseif $\exists\ i \in$ inputs(G), value(i) $= X$ ***then***
  $P_{good} = \{j \mid j \in Nodes \wedge value(j) \neq Forcing(G)\}$
else $P_{good} = \{j \mid j \in Nodes \wedge value(j) = X\}$ $\diamond$

If the gate G is analyzed under the application of several error detecting patterns $TP_1, TP_2, \cdots TP_n$ then:

$$P_{good}(G) = P_{good}(G,TP_1) \cap P_{good}(G,TP_2) \cap \cdots P_{good}(G,TP_n)$$

$$P_{bad}(G) = P_{bad}(G,TP_1) \cap P_{bad}(G,TP_2) \cap \cdots P_{bad}(G,TP_n)$$

4.1 Analysis with non-detecting patterns

Proposition 4.4 *: If under the application of a test pattern TP the current value at the output of gate G, $CV(G,TP)$, has to be fixed to keep the value of a correct primary output unchanged, then the set $I_{good}(G,TP)$ of impossible good connections to G is computed as follows:*

$$
\begin{aligned}
&\textbf{if } CV(G,TP) = \overline{Forced(G)} \textbf{ then}\\
&\quad I_{good}(G,TP) = \{j \mid j \in IMPL \wedge \\
&\qquad\qquad value(j) \neq \overline{Forcing(G)}\}\\
&\textbf{elseif } CV(G,TP) = Forced(G) \textbf{ then}\\
&\quad \textbf{if } (\mid P_{bad}(G) \mid = 1) \wedge (\{i \mid i \in inputs(G) \wedge \\
&\qquad value(i) = Forcing(G)\} = P_{bad}(G)) \textbf{ then}\\
&\qquad I_{good}(G,TP) = \{j \mid j \in IMPL \wedge \\
&\qquad\qquad value(j) \neq Forcing(G)\}\\
&\quad \textbf{else } I_{good}(G,TP) = \phi; \textbf{ endif}\\
&\textbf{else } I_{good}(G,TP) = \phi; \textbf{ endif} \diamond
\end{aligned}
$$

Based on the above propositions, a diagnosis algorithm similar to the one presented in section 3.2 is implemented. We don't write it explicitly for brevity.

4.2 Test pattern generation

Proposition 4.5 *: A test pattern TP detects whether a node i is wrongly connected to the input of a gate G instead of another node j if when TP is applied to the input of IMPL, value(i) is equal to $\overline{value(j)}$, and any other input $k \in inputs(G), k \neq i$ has a value equal to $\overline{Forcing(G)}$, and a path is sensitized from the output of G to at least one of the primary outputs.* $\diamond$

4.3 Extra connection errors

An extra connection error can be corrected by connecting the extra connection at the input of a gate G to any node in the circuit having the value $\overline{Forcing(G)}$ (this may be either V_{cc} or V_{dd}).

5 Experimental results

To validate the above algorithms, a prototype diagnosis system is implemented in PROLOG, including the test pattern generator and the logic simulator. The software is tested on the ISCAS'85 [15] benchmarks. The characteristics of these circuits are shown in Table 3. Thousands of diagnosis experiments were made. In each one of them an error of random type was inserted at a random location in the circuit and the diagnosis was performed. The results obtained on a SPARC-10 workstation with 10 Megabytes of memory are shown in Table 4. The column titled "#exp" gives the number of experiments made on each circuit. The other columns give, for each error hypothesis, the average number of test patterns used before returning a set of error candidates "#pat", the average CPU time in seconds "CPU", and the number of error cadidates proposed by the system "#cand".

Name	Number of			Name	Number of		
	In	Out	Gates		In	Out	Gates
c17	5	2	6	c2670	233	140	1193
c432	36	7	160	c3540	50	22	1669
c499	41	32	202	c5315	178	123	2307
c1355	41	32	546	c6288	32	32	2416
c1908	33	25	880	c7552	207	108	3512

Table 3: Characteristics of the ISCAS'85 benchmarks

In all the cases the error was found with high precision. The number of proposed candidates is exactly one in almost all of the cases. The CPU time grows almost linearly with the product of the number of gates and the number test patterns used before a diagnostic report is made, which itself depends highly on the topology of the circuit under test. We found that for the circuits with large number of inputs and outputs, (c2670, c5315, and c7552), the average number of applied test patterns is relatively small with respect to other circuits. This is due to the fact that if the error is discovered on a large number of outputs, then the erroneous gate must reside within the common cone of influence of all these outputs. This reduces rapidly the search space of the circuit.

When we modified and verified the implementation using the proposed candidates, we found that in almost all cases the different candidates, if there are several, are all valid. This may arise for instance when a connection is missing at the input of an AND gate in a succession of n AND gates. A correction can be made by connecting the missing node to the input of any one of the n gates. Thus the diagnoser will propose n candidates. In other cases we find only one suspected gate, and several connections are proposed as missing or bad connections to it. This happens when several nodes in the circuit represent the same function (e.g. the input and the output of a buffer).

The obtained results are extremely valuable, and justify the development of a production quality version of the software, written in a more efficient programming language, which will reduce greatly the CPU times shown here (until now we didn't devote much effort to program optimization, as our main interest was to show the applicability of our approach). Our prototype diagnosis system is now a part of the PREVAIL^{TM} environment [16] and is being tested on industrial applications.

6 Conclusion

New algorithms for diagnosing missing, extra and bad connection errors in combinational circuits have been presented. This error model represents a real in-

Name	#exp	Missing Connection			Extra Connection			Bad Connection		
		#pat	CPU	#cand	#pat	CPU	#cand	#pat	CPU	#cand
c17	17	3.50	0.08	1.00	3.20	0.11	1.00	5.00	0.15	1.00
c432	326	13.28	72.43	1.63	10.38	34.34	1.11	27.29	110.12	1.01
c499	473	28.00	195.18	1.25	8.90	83.27	1.23	26.64	176.69	1.01
c1355	1306	24.00	633.99	1.19	10.83	118.74	1.18	46.14	653.35	1.08
c1908	1233	24.29	890.34	1.12	14.35	233.67	1.16	31.29	1345.77	1.76
c2670	985	12.76	409.05	1.54	7.23	171.58	1.01	19.85	602.13	1.07
c3540	451	18.86	2681.60	1.46	16.25	1290.74	1.19	33.36	4462.18	1.31
c5315	571	13.26	806.03	1.06	16.14	1482.12	1.13	22.56	2319.41	1.01
c6288	347	22.63	7016.23	1.00	20.63	1983.16	1.63	32.37	7515.72	1.10
c7552	233	18.53	4762.11	1.27	10.20	1425.78	1.04	12.13	1661.92	1.00

Table 4: Diagnosis results on the benchmark circuits

dustrial request. A prototype software is implemented and its results on benchmark circuits show that the exact location of the error is always found using a small number of specially generated test patterns. The CPU time is almost proportional to the product of the circuit size and the number of patterns.

The use of this diagnosis tool, combined with the use of our previous software that identifies and corrects functional gate errors, can effectively reduce design time, by either finding errors, or eliminating the single erroneous gate hypothesis. The overall system examines in turn each possible error hypothesis, and applies the corresponding program, until one or more correction is suggested, or until all error hypotheses are found impossible. For human efficiency reasons, we believe that every step of manual modification in a design should be followed by a step of formal verification. Whenever equivalence between the initial and modified design (for a manual optimization), or the modified specification and the modified design (for a functional change in the specification) is denied, the diagnosis system is invoked. If few changes have been made, industrial experience shows that the single error hypothesis is realistic, and the correction can be obtained automatically.

Future works will include the extension of connection errors diagnosis to sequential circuits, along the lines of what has been done in [12], and the consideration of more complex functional errors.

References

[1] L. Yang, D. Gao, J. Mostoufi, R. Joshi, and P. Loewenstein, "System Design Methodology of UltraSPARC^{TM}-I," in *Proc. DAC'95*, pp. 7-12, 1995.

[2] A. Bartsch, H. Eveking, H. Faerber, M. Kelelatchew, J. Pinder and U. Schellin, "LOVERT - A Logic Verifier of Register Transfer Descriptions," in *Proc. IFIP Int. Workshop on Applied Formal Methods for Correct VLSI Design*, Houthalen (Belgium), Nov. 1989.

[3] M. S. Abadir, J. Ferguson, and T. E. Kirkland, "Logic Design Verification via Test Generation," *IEEE Transactions on Computer-Aided Design*, Vol. 7, No. 1, pp. 138-148, January 1988.

[4] G. Odawara, M. Tomita, O. Okuzawa, T. Ohta, and Z.-Q. Zhuang, "A Logic Verifier Based on Boolean Comparison," in *Proc. DAC'86*, pp. 208-214, 1986.

[5] K. A. Tamura, "Locating Functional Errors in Logic Circuits," in *Proc. DAC'89*, pp. 185-191, 1989.

[6] J. C. Madre, O. Coudert, J. P. Billon, "Automating the Diagnosis and the Rectification of Design Errors with PRIAM," in *Proc. ICCAD'89*, pp. 30-33, 1989.

[7] H.-T. Liaw, J.-H. Tsaih, C.-S. Lin, "Efficient Automatic Diagnosis of Digital Circuits," in *Proc. ICCAD'90*, pp. 464-467, 1990.

[8] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano, "Rectification of Multiple Logic Design Errors in Multiple Output Circuits," in *Proc. DAC'94*, pp. 212-217, 1994.

[9] P.-Y. Chung, Y. M. Wang, I. N. Hajj, "Diagnosis and Correction of Logic Design Errors in Digital Circuits," in *Proc. DAC'93*, pp. 503-508, 1993.

[10] Q. Zhang, "Logic Verification and Design Error Diagnosis for Combinational Circuits," *Ph.D. Thesis*, Université Catholique de Louvain, Belgium, Feb. 1995.

[11] A. Wahba, and D. Borrione, "A Method for Automatic Design Error Location and Correction in Combinational Logic Circuits," *Journal of Electronic Testing: Theory and Applications*, Kluwer, Vol. 8, No. 2, April 1996.

[12] A. Wahba, and D. Borrione, "Design Error Diagnosis in Sequential Circuits," in *Proc. Correct Hardware Design and Verification Methods, CHARME'95, LNCS* No. 987, pp. 171-188, Springer Verlag, Oct. 1995.

[13] A. Wahba, and D. Borrione, "Connection Errors Location and Correction in Combinational Circuits," *Research Report TIMA 96.1-I*, INPG, Grenoble, France, Feb. 1996.

[14] Prabhakar Goel, "An Implicit Enumeration Algorithm to Generate Tests for Combinational Logic Circuits," *IEEE Transactions on Computers*, Vol. C-30, No. 3, pp. 215-222, March 1981.

[15] F. Brglez, and H. Fujiwara, "A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN," in *Proc. ISCAS'85*, pp. 663-698, June 1985.

[16] D. Borrione, L. Pierre, and A. Salem, "Formal Verification of VHDL Descriptions in the Prevail environment," *IEEE Design & Test of Computers*, Vol. 9, No.2, pp. 42-56, June 1992.