Automatic Diagnosis may Replace Simulation for Correcting Simple Design Errors

Ayman M. Wahba Dominique Borrione
Laboratoire TIMA - UJF, BP 53, 38041 Grenoble Cedex 9, France

Abstract

An automated tool for diagnosing simple design errors in VHDL descriptions is presented. The tool is tested on benchmark circuits, and the results show that the error is localized precisely, after the application of a small number of specially generated test patterns. This tool is now integrated within the PREVAIL^{TM} system, and is being tested on industrial circuits.

1. Introduction

Design fault diagnosis plays an essential role in providing correct circuits. Although automated synthesis tools are used to provide correct by construction products, manual changes are made to achieve some critical design aspects such as speed and area requirements, or to carry out small specification changes. Industrial experience shows that a phase of design correction is always necessary, and many examples can be cited. For instance, in the EWSD-CCS7E processor developed by SIEMENS, a total of 320 errors were discovered during the simulation phase [3].

The place of the diagnosis in the design process is shown in Figure 1. We assume that a *specification* is given and validated using simulation and/or formal techniques. After a synthesis step (whether automatic or manual, or a combination of both), a description of the *implementation* is produced. A suitable verifier is then used to check the correctness of the implementation with respect to the specification. If an error is detected, counter examples are generated; the diagnosis and the correction are carried out, and the verification is done again. This diagnosis-correction-verification cycle is repeated until a correct implementation is obtained.

Existing verification tools can discover the existence of design errors, but provide no information about their nature or how to correct them. In the best case, verifiers issue counter examples in the form of input patterns that witness a difference between the behavior of the implementation and the specification. The task of finding and correcting the error is then left for the designer who uses the counter examples to reason about the type and the location of the error. This manual diagnosis takes a very long time, that can exceed the design time itself. It is thus necessary to mechanize it.

In this paper we present an automated diagnostic system for simple design errors, in both combinational and sequential circuits. The system is based on the close cooperation of three basic modules: a test pattern generator, a simulator and a diagnoser. The pattern generator generates special, diagnosis-oriented test patterns. The simulator simulates the implementation and the specification under the application of these patterns, and gives the simulation result to the diagnoser which, in turn, uses this result to limit the suspected area of the circuit. This information is passed to the pattern generator to guide it in generating new patterns, and the same operation is repeated until the error is found. Figure 2 shows the information flow among these three modules.

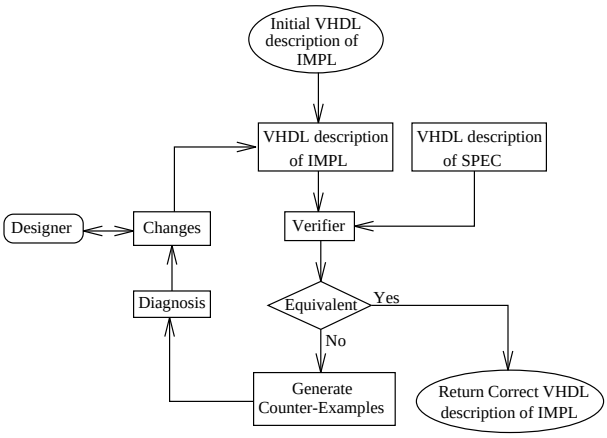

Figure 1. Diagnosis in the design process.

The diagnosis system was implemented in PROLOG, and it accepts circuit descriptions in a special PROLOG format. To integrate this system with the existing CAD tools, a translator was built to transform a subset of VHDL into the required PROLOG format. The translated format keeps information about the original VHDL descriptions, so that the diagnostic system can determine exactly the line number of the VHDL source file where the error exists.

Section 2 introduces the basic definitions and terminology used through out the paper. Section 3 describes the concept and the main rules of the diagnostic system. The results are given in Section 4, and our conclusions are finally presented in section 5.

2. Definitions and Terminology

Throughout this paper we consider a circuit specification *SPEC*, and its implementation *IMPL*. The specification out-

EURO-DAC '96 with EURO-VHDL '96
0-89791-848-7/96 $4.00

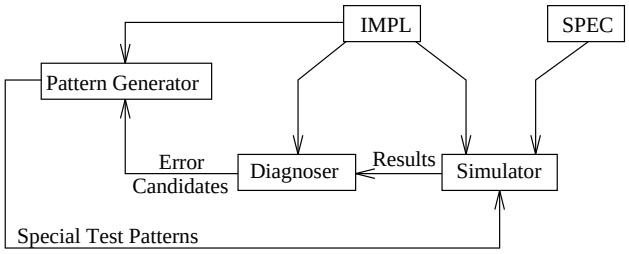

Figure 2. The overall diagnosis system.

put 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. In the following, whenever we refer to a gate, it is a gate in the implementation (since the specification may be described functionally).

Definition 2.1 : Test Patterns.
For a circuit with n inputs, a *test pattern* is an n-bit vector over the ternary domain $\mathcal{T}^n$, where $\mathcal{T} = \{0, 1, X\}$.
$\mathcal{T} = \{0, 1, X\}$ - The Ternary Domain.
X is an unspecified value, with the semantics that the outputs shouln't depend on it.

Definition 2.2 : The *cone of influence* of a gate *G*, *COI(G)*, is the set of all the gates and primary inputs that lie on any signal path directed from the primary inputs to *G*. $\diamond$

Definition 2.3 : Boundary of a set of gates
A gate G_2 is called a *successor* of a gate G_1, ($G_1 \neq G_2$), if the output of G_1 is connected to at least one input of G_2. The *boundary* $B(R)$ of a set of gates R, under the application of a test pattern, is the subset of gates of R that have no successors in R, and whose output value is '0' or '1'. $\diamond$

Error Model and Basic Assumptions

The error model for our diagnosis algorithms is based on the study presented in [2] about simple design errors, and the problems addressed to us by the design engineers at Thomson-TCS. Two main categories of errors are covered: gate errors, and connection errors (see Figure 3). Gate errors include missing or extra inverters, the replacement of a gate by another one of a different type, and extra gate errors. Connection errors include missing or extra connections at the input of a gate, the replacement of a connection by another one, and bad connections to '1' or '0'.

The following assumptions are made concerning the implementation to be diagnosed:

1. Only one error, at most, is assumed to occur in the cone of influence of each primary output.
2. The error is one of the above mentioned ones.
3. The gate types used are AND, NAND, OR, NOR, XOR, XNOR, BUFfer, and NOT.
4. The error does not introduce loops in the design.

The Principle of Diagnosis by Error Hypothesis

If we consider a circuit of n gates, and assume that one of m possible errors can occur at any gate, then under the assumption of a single error the number of possible errors

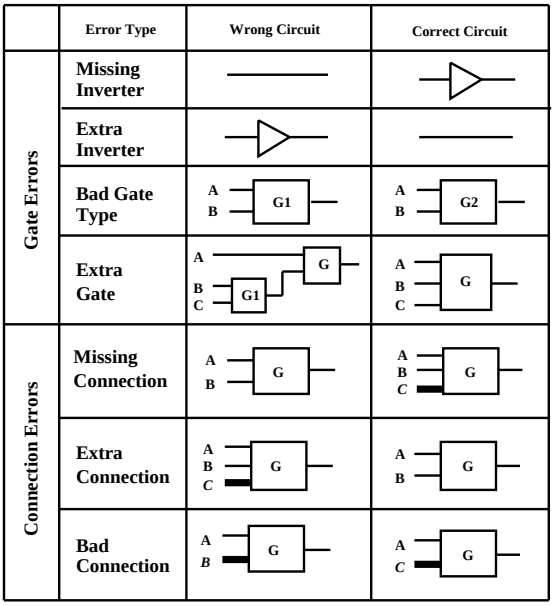

Figure 3. The design errors model

is $m \times n$. If we consider only one error type at a time, this number is reduced to n and the analysis is easier.

For this reason we adopted the methodology of diagnosis by error hypothesis: an error type is assumed, and the diagnosis is made. If the error is not found, another type is chosen, and a (possibly) different diagnosis procedure is applied, and so on until the error is found. The error hypotheses are selected in this order (decreasing probability of occurrence, according to [1]):

- HYP-0: An extra/missing inverter.
- HYP-1: A gate replacement of type 1. (OR $\leftrightarrow$ AND, NOR $\leftrightarrow$ NAND).
- HYP-2: A gate replacement of type 2. (OR $\leftrightarrow$ NAND, NOR $\leftrightarrow$ AND).
- HYP-3: An extra wire error.
- HYP-4: A missing wire error.
- HYP-5: A bad connection error.

The replacements (AND $\leftrightarrow$ NAND) and (OR $\leftrightarrow$ NOR) are covered by HYP-0. It was shown in [11] that HYP-0, HYP-1 and HYP-2 also cover extra gate errors, and the replacement of XOR/XNOR gates by other gates, if each XOR/XNOR gate is replaced by an equivalent network of AND, OR, NOR, NAND and NOT gates.

Definition 2.4 : Suspected and Correct gate sets.
Let *P* and *P'* be test patterns, *P* witnesses the error, and *P'* not necessarily. The *suspected-gate set SG(P)* contains any gate G such that $Y = W$ under the application of P, if a correction is made at G according to the assumed error hypothesis. If under the application of *P'*, $y_i = w_i$ for some $i \in \{1, 2, ...m\}$, then the *correct-gate set CG(P')* contains any gates G such that $w_i \neq y_i$ if G is changed according to the assumed error hypothesis. $\diamond$

Definition 2.5 : $CV(G, P)$ is the *current value* at the output of a gate G, when the pattern P is applied to the implementation primary inputs. $\diamond$

Hypothesis	Gate	$CV(G,P)$	$RV(G,P)$ = '0'	$RV(G,P)$ = '1'	$RV(G,P)$ = 'X'
HYP-1	OR	'0'		Not Suspected	Not Suspected
		'1'	$\exists e \mid v(e) =' 0'$		$\forall e, v(e) \neq' 0' \wedge \exists e \mid v(e) =' X'$
	AND	'0'		$\exists e \mid v(e) =' 1'$	$\forall e, v(e) \neq' 1' \wedge \exists e \mid v(e) =' X'$
		'1'	Not Suspected		Not Suspected
HYP-2	OR	'0'		Suspected	Not Suspected
		'1'	$\forall e, v(e) =' 1'$		$\forall e, v(e) \neq' 0' \wedge \exists e \mid v(e) =' X'$
	AND	'0'		$\forall e, v(e) =' 0'$	$\forall e, v(e) \neq' 1' \wedge \exists e \mid v(e) =' X'$
		'1'	Suspected		Not Suspected
HYP-3	OR	'0'		Not Suspected	Not Suspected
		'1'	$\exists e \mid v(e) =' 1' \wedge$ $\forall i \neq e, v(i) =' 0'$		$\exists e, j \mid v(e) =' 1' \wedge v(j) =' X' \wedge$ $\forall i \notin \{e, j\}, v(i) \neq' 1'$
	AND	'0'		$\exists e \mid v(e) =' 0' \wedge$ $\forall i \neq e, v(i) =' 1'$	$\exists e, j \mid v(e) =' 0' \wedge v(j) =' X' \wedge$ $\forall i \notin \{e, j\}, v(i) \neq' 0'$
		'1'	Not Suspected		Not Suspected
HYP-4	OR	'0'		Suspected	Suspected
		'1'	Not Suspected		Not Suspected
	AND	'0'		Not Suspected	Not Suspected
		'1'	Suspected		Suspected
HYP-5	OR	'0'		Suspected	Suspected
		'1'	$\exists e \mid v(e) =' 1' \wedge$ $\forall i \neq e, v(i) =' 0'$		$\exists e \mid v(e) =' 1' \wedge$ $\forall i \neq e, v(i) \neq' 1'$
	AND	'0'		$\exists e \mid v(e) =' 0' \wedge$ $\forall i \neq e, v(i) =' 1'$	$\exists e \mid v(e) =' 0' \wedge$ $\forall i \neq e, v(i) \neq' 0'$
		'1'	Suspected		Suspected
	NOT	'0'		Suspected	Suspected
		'1'	Suspected		Suspected

Table 1. Rules for determining SG (HYP-1 to HYP-5)

Type	$CV(G,P)$	$RV(G,P)$ = '0'	$RV(G,P)$ = '1'	$RV(G,P)$ = 'X'
OR	'0'		$CI = \{e \mid e \in inputs(G)\}$	$CI = \phi$
	'1'	CI = $B\left(\bigcap_{v(e)\neq 0} COI(e)\right)$		if $\exists e \mid v(e) =' X'$ then $CI = B\left(\bigcap_{v(e)='1'} COI(e)\right)$ else $CI = \phi$
AND	'0'		CI = $B\left(\bigcap_{v(e)\neq 1} COI(e)\right)$	if $\exists e \mid v(e) =' X'$ then $CI = B\left(\bigcap_{v(e)='0'} COI(e)\right)$ else $CI = \phi$
	'1'	$CI = \{e \mid e \in inputs(G)\}$		$CI = \phi$
NOT	'0'		$CI = \{e \mid e \in inputs(G)\}$	$CI = \phi$
	'1'	$CI = \{e \mid e \in inputs(G)\}$		$CI = \phi$

Table 2. Rules for determining changeable inputs

Definition 2.6 : Required value at a gate output
Let $Y_m(G,V,P)$ denote the value obtained at the output of the implementation, under the application of an input pattern P, if the current value at the output of a gate G, $CV(G,P)$, is replaced by another value V.
The *required value* at the output of a gate G, $RV(G,P)$, under the application of an error detecting pattern P, is the value which satisfies $Y_m(G, RV(G,P), P) = W(P)$. $\diamond$

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

3. Diagnosis by Backward-Propagation

Diagnosis by backward-propagation is based on simulating the specification and the implementation under the application of ternary test patterns. Depending on the simulation result, each test pattern is classified as an Error-Detecting pattern, *EDP*, or a Non-Detecting Pattern, *NDP*, and the implementation is then analyzed to extract the Suspected-Gate sets and Correct-Gate sets. A basic principle is that the test patterns used in our method always produce a definite value ('0' or '1') at the *implementation* outputs.

If under the application of a pattern P, the error is detected at one output y_i, then two possibilities may arise: either $w_i(P) = \overline{y}_i(P)$, or $w_i(P) = 'X'$.

First case: $w_i(P) = \overline{y}_i(P)$

The diagnosis starts at the gate G whose output is y_i. If the correction at G according to the assumed error hypothesis makes $y_i(P) = w_i(P)$ then G is put in the suspected gate set SG(P). But the value of $w_i(P)$ *may* also be obtained by making no correction at G and *complementing* the value of one of its inputs e (e is a changeable input). If this is possible, the diagnosis algorithm investigates also the gate from which e originates, and checks whether the correction at this gate results in complementing the value of e, and consequently makes $y_i(P) = w_i(P)$. The operation is repeated recursively until the primary inputs are reached.

Second case: $w_i(P)$ = **'X'**

The same procedure is applied but the investigation begins at the inputs of the gates that inhibit the propagation of the value 'X', (i.e. those gates that have 'X' at their inputs but not at their outputs). This greatly reduces the search area.

Rules that determine if a gate is suspected

These rules are defined for any gate G of the implementation under the application of a test pattern P, and for each error hypothesis.

Hypothesis HYP-0: If for a gate G in the implementation, $CV(G,P) = \overline{RV(G,P)}$, then G is suspected, and an inverter may be placed at its output to correct the implementation. Fictitious gates of type *buffer* are created at the primary inputs to include the case of a missing inverter at some input.

Hypotheses HYP-1 to HYP-5: The rules for determining whether a gate G is suspected or not, under these error hypotheses, are listed in Table 1. The entries of this table represent the conditions on the inputs of G under which G is considered a suspected gate. In this table e, i, j represent any inputs of the gate G, and $v(e)$, $v(i)$, $v(j)$ are their current values. The entries marked *Not Suspected* mean that the gate G is not suspected, and the entries marked *Suspected* mean that the gate G is suspected with no conditions on the values of its inputs.

For instance, the second line of the table reads as follows. Column 4: if the current value of an OR gate G is '1', and the required value is '0', then G is suspected according to HYP-1 if at least one of its inputs has the value '0'. Column 6: if the current value of an OR gate G is '1', and the required value is 'X', then G is suspected according to HYP-1 if none of its inputs is '0', and at least one input is 'X'.

In tables 1 to 4, we only give the rules for OR and AND gates, plus NOT when applicable. Rules for NAND and OR gates (resp. for NOR and AND) are the same, if you interchange the occurences of '0' and '1' in the rules.

Rules that determine the changeable inputs of a gate

The required value at the output of one gate may be obtained by complementing the value of one of its inputs. However, only certain inputs, called *changeable inputs*, (see definition 2.7), can achieve this goal. The rules for determining the set of changeable inputs (*CI set*) of a gate G under the application of a test pattern P are shown in Table 2. These rules are valid under all error hypotheses, since the *CI* sets depend only on the current value of the gate inputs, the value of the gate output, and the gate type. In this table e represents any input of the gate G, $v(e)$ its current value, COI(e) the cone of influence of the gate from which e originates, and ϕ denotes the empty set.

If the error is not detected on some output y_i under the application of a pattern P, then $y_i(P)$ must not be changed after the correction of the circuit. The circuit is scanned from this correct output backward to the primary inputs, but in this case we extract a set of correct gates $CG(P)$. The rules used to get $CG(P)$ vary according to the error hypothesis:

Hypothesis	Type	Gate Output $'0'$	Gate Output $'1'$
HYP-1	OR	Not Correct	$\exists e \mid v(e) \neq' 1'$
	AND	$\exists e \mid v(e) \neq' 0'$	Not Correct
HYP-2	OR	Correct	$\forall e, v(e) =' 1'$
	AND	$\forall e, v(e) =' 0'$	Correct

Table 3. Rules for getting CG (HYP-1, HYP-2)

HYP-0: Any inverter under investigation belongs to CG, if its output value is $'0'$ or $'1'$.

HYP-1 and HYP-2: Table 3 contains the necessary conditions on the inputs of a gate G, for G to be correct. In this table, the entries marked *Not Correct* mean that the gate G cannot be put in the correct-gate set CG(P). The entries marked *Correct* mean that G is a correct gate without conditions on its inputs.

HYP-3, HYP-4 and HYP-5: Here we can't determine correct gate sets due to the nature of the connection errors.

When a gate is found to be correct, its inputs are also investigated to extract the *fixed inputs* (see definition 2.8). The rules used to determine the fixed-inputs set, *FI*, of a gate G under the application of a test pattern P are listed in Table 4. In this table, e and i are any inputs of the gate G, and $v(e)$ and $v(i)$ are their current values.

In the case of connection errors, it is not sufficient to determine the suspected gate, but also which connection is the erroneous one. For extra-connection errors, the extra input of the gate must be specified. For missing connection errors, the node from which the missing connection originates must be determined, and for bad-connection errors, the bad and the good connections must be specified. The algorithms for determining the erroneous connections may be found in [13], as there is no place to detail them here.

3.1. The Diagnosis Algorithm

Given a test pattern *TP* and assuming a certain error hypothesis *HYP*, the algorithm scans the circuit from the primary outputs backward to the primary inputs. This is made by the recursive function *scan-circuit* described below. Before calling the algorithm for the first time, the suspected gate set SG contains all the circuit gates, and then it is reduced with the repeated application of the algorithm with different test patterns. The operation stops if the size of SG is reduced to zero, which means that the *HYP* is a wrong hypothesis and another one must be tried, or when no more test patterns are supplied by the test pattern generator.

```
algorithm diagnose-circuit(TP, HYP);
begin
    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 SG = φ;
        CG = φ;
        if y_i ≠ w_i then Pat-Type := EDP else Pat-Type := NDP;
        scan-circuit(Pat-Type, G_{y_i}, HYP);
        if Pat-Type = EDP then
            SG = New SG;
        else
            SG = SG - CG;
        endif;
    endfor;
end.
```

Type	Gate Output	
	$'0'$	$'1'$
OR	$FI = \{e \mid e \in inputs(G)\}$	if $\exists e \mid v(e) =' 1' \wedge$ $\forall i \neq e, v(i) \neq' 1'$ then $FI = \{e\}$, else $FI = \phi$
AND	if $\exists e \mid v(e) =' 0' \wedge$ $\forall i \neq e, v(i) \neq' 0'$ then $FI = \{e\}$, else $FI = \phi$	$FI = \{e \mid e \in inputs(G)\}$
NOT	$FI = \{e \mid e \in inputs(G)\}$	$FI = \{e \mid e \in inputs(G)\}$

Table 4. Rules for determining FI

```
procedure scan-circuit(Pat-Type,G,HYP);
begin
    if G is a primary input then
        exit;
    endif;
    if Pat-Type = EDP then
        if (G ∈ SG) ∧ (G is suspected) then
            New SG = New SG ∪ {G};
        endif;
        for every input i ∈ Changeable-Inputs(G) do
            scan-circuit(Pat-Type,i,HYP);
        endfor;
    else
        if (G ∈ SG) ∧ (G is correct) ∧ HYP = HYP-0, HYP-1 or HYP-2 then
            CG = CG ∪ {G};
        endif;
        for every input i ∈ Fixed-Inputs(G) do
            scan-circuit(Pat-Type,i,HYP);
        endfor;
    endif;
end.
```

3.2. 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 test patterns specially generated for the diagnosis. The method is as follows: a gate G is selected from the suspected gate set, and two patterns are generated for it. One pattern can detect the error if G was the erroneous one, while the other one does not activate the error and thus does not detect it, if G was the erroneous one. It was proven in [11] that this method allows to reduce very rapidly the suspected gate set. In *favorable cases*, the error can be exactly located after the application of only one pair of these patterns.

The complete diagnosis algorithm is thus given below:

```
algorithm diagnose(Error Hypothesis);
begin
    SG = All the circuit gates;
    Tested ← φ;
    while (| SG | > 1) and (SG ⊈ Tested) do
        Let G be a gate in SG and G ∉ Tested
        Tested ← Tested ∪ {G};
        TP1 = Pattern that detects the error if G is erroneous.
        TP2 = Pattern that does not detect the error if G is erroneous.
        diagnose-circuit(TP1,Error Hypothesis);
        diagnose-circuit(TP2,Error Hypothesis);
    enddo
    write(Error Hypothesis, SG);
end
```

3.3. Diagnosis of Sequential Circuits

We extended this diagnosis algorithm to synchronous sequential circuits under HYP-0, HYP-1 and HYP-2. The implementation and the specification may have different number of states and state encoding. The specification is regarded as a black box on which only the primary inputs and the primary outputs are observable, and which can be initialized in an initial state corresponding to the initial state of the implementation.

We introduced the new concept of *Possible Next States*. They are the set of states reachable from a given initial state, or a given set of initial states, due to the existence of several possible locations of the error. The implementation is represented by its *iterative logic array model* [5], and simulated in each time frame separately. It is then diagnosed by applying combinational diagnosis rules, where the present-state lines are treated as primary inputs, and the next-state lines as primary outputs. Before proceeding to the next time frame, the set of possible next states is computed, and then the analysis is done in the next time frame under each one of these possible next states. This operation is repeated until the error is found. The method is fully described in [12].

4. Results

A prototype diagnosis system implements the above algorithms in PROLOG, including the test pattern generator and the logic simulator. The software was applied to circuits taken from the ISCAS'85 and ISCAS'89 benchmarks [7, 6].

The results obtained on a SPARC-10 workstation with 10 Megabytes of memory are shown in Table 5 for combinational circuits , and Table 6 for sequential circuits. The columns titled "No. of Exp." give, for each circuit, the number of diagnosis experiments made. Each experiment is made on a randomly inserted error of a random type according to the six error hypotheses HYP-(0-5). The columns titled "Pat. Used" gives the average number of test vectors applied before the diagnosis report is generated. The average CPU time is given in seconds. This time includes pattern generation, simulation and diagnosis times. The average number of suspected gates returned as result is given in the columns titled "Cand."

Result Analysis:

In 100% of the experiments, the actual error was among the final suspected gates. The error is always found with a fine diagnostic resolution: in almost all of the cases only one error candidate is proposed. In the case of combinational circuits the number of test patterns applied before finding the error is small compared with other methods [8, 10, 14], thanks to the mixed use of error detecting and non-detecting patterns [11]. The execution time grows almost linearly with the product of the number of gates and the applied test patterns. The number of applied test patterns, before a diagnostic report is made, depends highly on the topology of the circuit under test: 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 shows on a large number of outputs, then the erroneous gate must reside within the common cone of influence of all these outputs, which reduces the search area of the circuit.

Some sequential circuits require a large number of test vectors because they are difficult to control, and a long sequence of vectors is needed to witnesses the error (e.g.

Name	Number of			No. of Exp.	Average of		
	In	Out	Gates		Pat. used	CPU time	Cand.
c17	5	2	6	35	4.09	0.14	1.00
c432	36	7	160	686	18.11	58.62	1.16
c499	41	32	202	929	18.70	101.69	1.36
c1355	41	32	546	2728	28.08	411.97	1.16
c1908	33	25	880	2556	22.67	602.92	1.19
c2670	233	140	1193	1785	15.29	461.17	1.03
c3540	50	22	1669	1255	26.48	2187.08	1.17
c5315	178	123	2307	931	17.17	1681.16	1.15
c6288	32	32	2416	698	22.01	2102.75	1.65
c7552	207	108	3512	177	14.36	2993.50	1.02

Table 5. Results for ISCAS'85 benchmarks

s208, s382, s838, and s1423). Many factors may affect the length of the test sequences, such as the number of *loops per flip/flop* and the number of *state controlling inputs* [9].

The CPU times mentioned here are based on our experimental implementation of the algorithms. We didn't devote a large effort to program optimization, as our main interest was to show the applicability of our approach. We believe that these time performances can be significantly improved by rewriting the software using faster languages like C.

Name	Number of				No. of Exp.	Average of		
	In	Out	Gates	f/f		Pat.	CPU time	Cand.
s27	4	1	10	3	14	2.43	0.61	1.64
s208	11	1	96	8	114	132.79	251.05	3.44
s298	3	6	119	14	126	17.62	59.89	2.09
s344	9	11	160	15	174	6.55	64.88	1.16
s349	9	11	150	15	177	6.53	69.55	1.26
s382	3	6	158	21	104	125.21	1298.99	2.37
s386	7	7	159	6	222	18.10	49.18	3.26
s400	3	6	148	21	90	69.59	1009.89	2.22
s420	19	1	196	16	95	79.18	775.13	4.02
s510	19	7	211	6	209	18.07	127.75	2.32
s641	35	24	379	19	174	4.07	306.79	1.06
s820	18	19	289	5	111	8.41	199.52	2.31
s838	35	1	390	32	10	129.70	8427.23	3.60
s953	16	23	395	29	61	20.61	1360.41	3.95
s1196	14	14	529	18	378	2.69	167.37	1.91
s1423	17	5	657	74	10	211.00	14245.10	3.60
s5378	35	49	2779	179	10	20.30	14445.42	6.30

Table 6. Results for ISCAS'89 benchmarks

5. Conclusion

In this paper an original automatic diagnosis tool is presented. This tool finds precisely the type and location of simple design errors in VHDL descriptions of combinational and sequential circuits. The specification may be given at any level of abstraction, and the implementation is given at the RTL level. This means that in the case of sequential circuits, the implementation and the specification may have different number of states and state encodings.

We are now working on several extensions: first, we try to include other error hypotheses, in particular complex component replacements instead of the simple gate types considered here. A longer term research concerns multiple errors, for which no satisfactory method exists currently.

Finally, we believe that automatic diagnosis is an essential element in any CAD environment. Our prototype diagnosis system is now integrated within the PREVAIL^{TM} environment [4], and has been used successfully to find errors in industrial circuits supplied by Thomson-TCS.

6. Acknowledgments

This work is partially supported by the EUREKA "JESSI-AC3" project, and the ESPRIT Basic Research Action CHARME Working Group #6018.

References

[1] E. J. Aas, K. A. Klingsheim, and T. Steen. Quantifying design quality: A model and design experiments. In *Proc. EURO-ASIC'92*, pages 172–177. IEEE Computer Society Press, 1992.

[2] M. S. Abadir, J. Ferguson, and T. E. Kirkland. Logic design verification via test generation. *IEEE Trans on CAD*, 7(1):138–148, Jan. 1988.

[3] T. W. Albrecht. Concurrent design methodology and configuration management of the SIEMENS EWSD-CCS7E processor system simulation. In *Proc. DAC'95*, pages 222–227, 1995.

[4] D. Borrione, L. Pierre, and A. Salem. Formal verification of vhdl descriptions in the PREVAIL environment. *IEEE Design & Test of Computers*, 9(2):42–56, June 1992.

[5] M. A. Breuer and A. D. Friedman. *Diagnosis and Reliable Design of Digital Systems*. Computer Science Press, New York, 1976.

[6] F. Brglez, D. Bryan, and K. Kozminski. Combinational profiles of sequential benchmark circuits. In *Proc. IEEE Int. Symp. Circuits and Systems*, Portland, OR, May 1989.

[7] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinatorial benchmark circuits and a target translator in FORTRAN. In *Proc. IEEE Int. Symp. Circuits and Systems*, pages 663–698, June 1985.

[8] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin. Error diagnosis for transistor-level verification. In *Proc. DAC'94*, pages 218–224, 1994.

[9] A. Lioy, P. L. Montessoro, and S. Gai. A complexity analysis of sequential ATPG. In *Proc. IEEE Int. Symp. Circuits and Systems*, pages 1946–1949, May 1989.

[10] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano. Rectification of multiple logic design errors in multiple output circuits. In *Proc. DAC'94*, pages 212–217, 1994.

[11] A. Wahba and D. Borrione. Design error diagnosis in logic circuits using diagnosis-oriented test patterns. Research Report RR-940-I, ARTEMIS-IMAG, Grenoble, France, June 1994.

[12] A. Wahba and D. Borrione. Design error diagnosis in sequential circuits. In *Proc. Correct Hardware Design and Verification Methods, CHARME'95*, number 987 in Lecture Notes in Computer Science, pages 171–188. Springer Verlag, October 1995.

[13] A. Wahba and D. Borrione. Connection errors location and correction in combinational circuits. Research Report 96.1.I, TIMA-INPG, Grenoble, France, Feb 1996.

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