Multiple Error Diagnosis in Large Combinational Circuits Using an Efficient Parallel Vector Simulation

Yu-Lin Hsiao Chun-Yao Wang Yung-Chih Chen
Department of Computer Science
National Tsing Hua University, HsinChu, Taiwan
{mr944351, wcyao, ycchen}@cs.nthu.edu.tw

Abstract—This paper presents a parallel vector simulation-based approach to locating multiple errors in large combinational circuits. Two heuristics are proposed to avoid the explosion of the error space. Experimental results on a set of ISCAS'85 and two large benchmarks show that our approach efficiently identifies a small set of correctable nodes that contains the actual error sources. Thus, further error correction can be conducted on the erroneous implementation.

I. Introduction

Design errors possibly occur during the synthesis procedure, or can be injected occasionally by designers who manually rectify the implementation to meet the specification. Logic verification tools confirm the existence of design errors in the incorrect implementation by identifying functional mismatches between the implementation and the specification. Then, the design errors have to be located and further corrected. Since the error space is usually very large, it is inefficient to manually identify the error locations. Thus, error diagnosis algorithms are proposed to efficiently report some possible error candidates such that the subsequent correction procedure can be performed.

Previous work on error diagnosis can be classified into three groups: BDD-based approaches [4], [7], [8], simulation-based approaches [5], [6], [9], [12]-[15], and SAT-based approaches [2], [3], [11]. The BDD-based approaches rely on manipulation of Boolean function. Their idea is to identify a set of gates that can fix all incorrect output functions. These approaches pin-point the error sources exactly and can be easily extended to multiple error diagnosis. However, they use Ordered Binary Decision Diagram (OBDD) to represent the circuits. Therefore, they are not adequate for the large circuits which have no efficient OBDD representations.

On the other hand, simulation-based approaches are more applicable to large circuits. They simulate a large number of vectors to distinguish the erroneous implementation from the correct specification, and gradually prune the error space. In general, they are less accurate than BDD-based approaches due to the incomplete vector set. Two kinds of heuristics were found in the different simulation-based approaches: one is error simulation [5], [9], [12], [14], the other is logic reasoning [6], [13]. In comparison, the error simulation is more accurate and more extendable for multiple errors than the logic reasoning. However, the logic reasoning heuristics are more efficient in general. The SAT-based approaches first generate a SAT-instance to present the relation between the erroneous implementation and the correct specification. Then they use a SAT solver to solve the instance.

In our approach, we generate parallel random vectors (up to 2^{14}) and simulate them at a time. Then the mismatches in the outputs between the implementation and specification can be identified simultaneously. We derive two kinds of erroneous vectors to guide single error diagnosis and multiple error diagnosis, respectively. The details will be discussed in Section 3.

Compared with the previous work, our approach has the following three advantages: (1) Efficiency. Since we apply parallel vector simulation, the performance is improved by using efficient bit-wise operations. Moreover, some heuristics are also proposed to speedup the diagnosis process. (2) Accuracy. The experimental results show that we can prune error space such that the number of error candidates is small enough. Therefore, the accuracy of our approach can keep pace with that of BDD-based approaches. (3) Applicability and Scalability. Like BDD-based approaches, our approach does not rely on any error models and is suitable for multiple error diagnosis. Furthermore, it is applicable to large circuits that do not have efficient BDD representations.

Since the number of error sources in an erroneous implementation is unknown, we treat this problem as locating the set of minimal correctable sites rather than obtaining the actual error sources. In other words, this set of minimal correctable sites is considered as the equivalent error sources. In this paper, the term *design error* is referred to the actual or equivalent error.

The remainder of this paper is organized as follows. Section 2 gives our problem formulation, assumptions, and basic definitions. Section 3 describes the details of our approach. Section 4 provides the experimental results. Section 5 concludes this paper.

II. Preliminaries

In this work, we are given an erroneous implementation denoted as C_I, and the correct specification denoted as C_S. Both are combinational netlist but with different structures. *n* primary inputs are denoted as $\{X_1, X_2, ..., X_n\}$, and *m* primary outputs are denoted as $\{S_1, S_2, ..., S_m\}$ for C_S and $\{F_1, F_2, ..., F_m\}$ for C_I respectively. We assume both C_S and C_I are composed of primitive gates (AND, OR, NOT) for simplicity.

Both C_I and C_S are simulated by the same set of input vectors, denoted as *VEC* (the number of vectors is 2^p, p = 0 $\sim$ 14), which are simultaneously generated from a parallel random vector generator (PRVG) [17]. The simulation results of each node are stored in a corresponding *bit list*. The *bit list* is also known as signature. The i^{th} index of the bit list contains the logic value obtained by simulating the i^{th} random vector. The bit list at a node *l* is denoted as *l-blist*. The i^{th} index in the *l-blist* is denoted as *l-blist(i)*, i = 1 $\sim$ 2^p. In this paper, the index of the bit list always starts with 1 and is counted from the right-most bit to the left-most bit. The bit string of the *l-blist* is denoted as *l-blistval*. The value in the position of *l-blist(i)* is denoted as *l-blistval(i)*. The i^{th} input vector of *VEC* is denoted as *vec(i)*, and *vec(i)* = *(X_1-blistval(i), X_2-blistval(i), ..., X_n-blistval(i))*.

For example in Fig. 1, The bit string of X_1-*blist* is X_1-*blistval* = (01010101) and the value of X_1-*blist(3)* is X_1-*blistval(3)* = 1. *vec(2)* = *(X_1-blistval(2), X_2-blistval(2), X_3-blistval(2))* = (0, 1, 1).

(S_i, F_i) is called an output pair. For an output pair (S_i, F_i), if S_i-*blistval(j)* and F_i-*blistval(j)* are different, F_i-*blist(j)* is called an *erroneous bit*; otherwise, F_i-*blist(j)* is called a *correct bit*, for j = 1 $\sim$ 2^p. If *vec(j)* can create an erroneous bit in any F_i, *vec(j)* is called an *erroneous vector* and F_i is called an *erroneous output*; otherwise, F_i is called a *correct output*. The set of all the erroneous outputs caused by *vec(j)* is denoted as *EPO(vec(j))*, and the set of all the correct outputs caused by *vec(j)* is denoted as *CPO(vec(j))*.

To detect whether F_i-*blist(j)* is an erroneous bit or not, we can perform bit-wise EXOR operation between F_i-*blistval* and S_i-*blistval*. Therefore, all erroneous bits at an erroneous output can be identified simultaneously. For example in Fig. 1, S_1-*blistval* $\oplus$ F_1-*blistval* = (00011101) $\oplus$ (11011111) = (11000010). Thus, we can identify F_1-*blist(2)*, F_1-*blist(7)*, and F_1-*blist(8)* are erroneous bits, the others are correct bits. Similarly, F_2-*blist(1)*, F_2-*blist(2)*, F_2-*blist(7)*, and F_2-*blist(8)* are also erroneous bits. *vec(1)*, *vec(2)*, *vec(7)*, and *vec(8)* are erroneous vectors. Therefore, *EPO(vec(2))* = *EPO(vec(7))* = *EPO(vec(8))* = $\{F_1, F_2\}$, *EPO(vec(1))* = $\{F_2\}$, and

978-1-4244-1617-2/08/$25.00 ©2008 IEEE

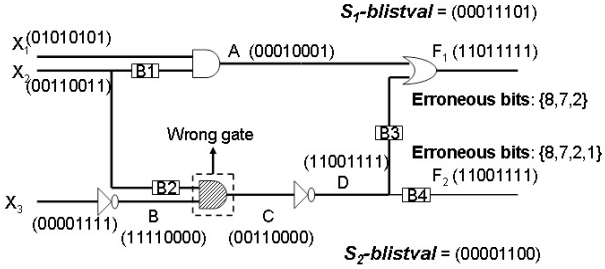

Fig. 1. A circuit with a single error. The primary inputs are X_1, X_2, and X_3. The primary outputs are F_1 and F_2. The internal wires (nodes) are A, B, C, and D. $B1 \sim B4$ are branch wires. The gate marked as "wrong gate" is the error source.

CPO(vec(2)) = CPO(vec(7)) = CPO(vec(8)) = $\{\emptyset\}$, *CPO(vec(1))=* $\{F_1\}$.

If the inverse of *l-blistval(j)* at a node l can cause F_i*-blistval(j)* to change, *l-blist(j)* is called a *sensitized bit*. The maximal set of sensitized bits at l for an F_i is called a *sensitization set* of F_i, and is denoted as *SEN(l,* F_i*)*.

In Fig. 1, *A-blistval(5)* is 1, when *A-blistval(5)* changes to 0, i.e., inverse, the affected F_1*-blistval(5)* is also inverted after simulation. So does *A-blistval(6)*. Therefore, *SEN(A,*F_1*)* = $\{5, 6\}$.

If C_I can be corrected by re-synthesizing a single node l, this node is called a *single correctable node*; otherwise, C_I has to be corrected by re-synthesizing a set of N nodes, where $N > 1$, these nodes are called *N-correctable nodes* = $\{l_1, l_2, ..., l_N\}$.

III. Diagnosis Approach

In this section, we describe our diagnosis approach. After identifying all erroneous bits and erroneous vectors on all erroneous outputs, we derive the sensitization sets for nodes in C_I. Then we use the sensitization sets to prune the nodes that are not likely to be responsible for correcting C_I. Next, the remaining nodes are examined for single error and multiple errors. Finally, the sets of correctable nodes are returned.

A. Sensitization Set Derivation

The sensitization sets of nodes can be derived based on two techniques. The first one is the critical path tracing technique [1], which performs the backward propagation. The second one is the parallel pattern single fault propagation (PPSFP) technique [16], which performs the forward simulation. The process of sensitization set derivation starts from the erroneous outputs toward to primary inputs in C_I. If it encounters non-fanout nodes, it performs backward propagations. For stems, it performs forward simulations. These two techniques are alternatively applied on C_I so that the sensitization sets of nodes are accurately derived.

B. Single Error

As we mentioned, the objective of this work is to identify the set of minimal correctable nodes for C_I. Therefore, in this paper, the single error case is referred as attaining a single correctable node. To identify a single correctable node, we construct an *observable erroneous vector* to guide the diagnosis process on single error.

Theorem 1: An erroneous vector *vec(j)* can be corrected by re-synthesizing a node l if and only if it both satisfies the following two conditions:

1) F_i*-blist(j)* is an erroneous bit, and *l-blist(j)* $\in$ *SEN(l,*F_i*)*, for every F_i in *EPO(vec(j))*.
2) F_i*-blist(j)* is a correct bit, and *l-blist(j)* $\notin$ *SEN(l,*F_i*)*, for every F_i in *CPO(vec(j))*.

This erroneous vector *vec(j)* is called an *Observable Erroneous Vector (OEV)* for *l*.

Therefore, if an erroneous vector *vec(j)* is an *OEV* for a node *l*, every erroneous bit F_i*-blist(j)* for F_i in *EPO(vec(j))*, can be corrected after performing the correction on *l*. Moreover, *vec(j)* will not create any additional erroneous bit for any F_i in *CPO(vec(j))*.

For example in Fig. 1, *vec(1)*, *vec(2)*, *vec(7)*, and *vec(8)* are the erroneous vectors. Here we want to know if they are *OEVs* for X_2. First, we derive the sensitization sets of X_2, *SEN(*X_2*,*F_1*)* = $\{8, 6\}$, and *SEN(*X_2*,*F_2*)* = $\{8 \sim 5\}$. The erroneous bits of F_1 are $\{8, 7, 2\}$ and those of F_2 are $\{8, 7, 2, 1\}$. Both F_1*-blist(8)* and F_2*-blist(8)* are the erroneous bits, and 8 is in both *SEN(*X_2*,*F_1*)* and *SEN(*X_2*,*F_2*)*. Therefore, according to Theorem 1, *vec(8)* is an *OEV* for X_2. On the other hand, F_2*-blist(1)* is an erroneous bit, and F_1*-blist(1)* is a correct bit. Although X_2*-blist(1)* is indeed not in *SEN(*X_2*,*F_1*)*, it is not in *SEN(*X_2*,*F_2*)*, either. Therefore, *vec(1)* is not an *OEV* for X_2. Similarly, we can determine *vec(2)* and *vec(7)* are not *OEVs* for X_2, either.

Theorem 2: A node l is a single correctable node if all erroneous vectors are *OEVs* for *l*.

Based on Theorem 2, we can determine the single correctable nodes in C_I. However, due to large error space in C_I, we propose an effective heuristic to reduce the error space.

Heuristic 1: If C_I has a single correctable node, this node can be tracked by tracing from the fanin cone of any erroneous F_i.

Therefore, in the process of deriving the sensitization sets, we only choose an erroneous output F_i, and derive the sensitization sets of the nodes in the fanin cone of F_i. These nodes will be examined subsequently by using Theorem 2. In our approach, we always choose the first erroneous output for the sake of easily extending our approach to multiple error diagnosis process. Next, we use Example 1 to demonstrate our single error diagnosis process.

Example 1: In Fig. 1, the erroneous bits of F_1 and F_2 are in the braces. We derive the sensitization sets of the nodes in the fanin cone of F_1. Next, we examine these nodes according to Theorem 2. For example, the sensitization sets of node *A* are *SEN(A,*F_1*)*=$\{6, 5\}$ and *SEN(A,*F_2*)*=$\{\emptyset\}$. It is clear that the erroneous vector *vec(1)* is not an *OEV* for *A*. This is because F_1*-blist(1)* is a correct bit, and F_2*-blist(1)* is an erroneous bit, but *A-blist(1)* is not in *SEN(A,*F_2*)*. Similarly, *vec(2), vec(7), and vec(8)* are not *OEVs* for *A*, either. Therefore, *A* is not a single correctable node. On the other hand, we can detect all the erroneous vectors are *OEVs* for *C* and *D*. Consequently, the actual error *C* and the equivalent error *D* are both identified as the single correctable nodes.

C. Multiple Errors

In this subsection, we extend our approach to identify *N*-correctable nodes in C_I. For simplicity, we introduce two errors into C_I to describe our idea.

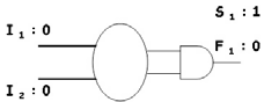

Fig. 2. A pseudo circuit with two error sources.

In Fig. 2, we suppose there are two error sources which cause I_1 = 0, I_2 = 0 and an erroneous output F_1 = 0 in C_I. Assume C_I has been identified as containing no single correctable node. It is obvious that the set (I_1, I_2) has four possible combinations of logic values, (0, 0), (0, 1), (1, 0), and (1, 1). Since (I_1, I_2) = (0, 0) has been simulated at the beginning to determine whether F_1 can be correctable, we perform the error simulations [14] under (I_1, I_2) = (0, 1), (I_1, I_2) = (1, 0), and (I_1, I_2) = (1, 1). We have to restore the original values of (I_1, I_2) and that of its fanout cones after completing one error simulation for the next error simulation. If at least one of the three error simulation results can produce the same response as S_1 = 1, F_1 can be corrected. However, the first two error simulations, (I_1, I_2) = (0, 1), (1, 0) are the same as identifying *OEVs* in single error diagnosis process. Thus, we propose the *co-observable erroneous vector* for the last simulation (I_1, I_2) = (1, 1).

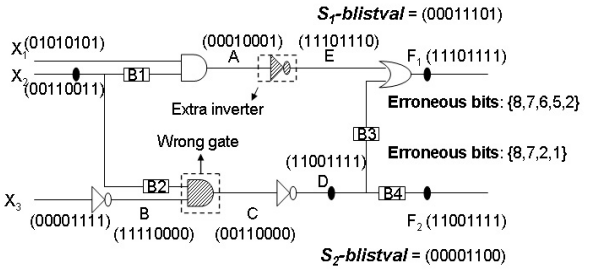

Fig. 3. A circuit with two error sources: Wrong gate and Extra inverter.

Theorem 3: An erroneous vector *vec(j)* can be corrected by re-synthesizing multiple nodes if and only if it both satisfies the following two conditions:

1) F_i*-blist(j)* is an erroneous bit, but after performing the error simulation with inversing multiple bit strings at nodes, F_i*-blist(j)* is not an erroneous bit, for every F_i in *EPO(vec(j))*.
2) F_i*-blist(j)* is a correct bit, but after performing the error simulation with inversing multiple bit strings at nodes, F_i*-blist(j)* is still a correct bit, for every F_i in *CPO(vec(j))*.

This erroneous vector *vec(j)* is called a *Co-Observable Erroneous Vector (C-OEV)* for the set of *N* nodes.

For example in Fig. 3, *vec(1)* is an erroneous vector. Assume that we choose the set (*A, D*) to examine. We simultaneously inverse *A-blistval* and *D-blistval* and perform the error simulation, the results are (00110001) for F_1 and (00110000) for F_2. Therefore, *vec(1)* is a *C-OEV* for (*A, D*) because the simulation results at the 1^{st} bit of F_1 and F_2 are $\{1, 0\}$, which are the same with S_1*-blistval(1)* and S_2*-blistval(1)*, respectively.

In general, *N*-correctable nodes in C_I are identified after performing 2^N-1 error simulations. Therefore, every erroneous vector *vec(j)* can be examined to determine if it can rectify all the erroneous bits F_i*-blist(j)* for F_i in *EPO(vec(j))*, and will not create any new erroneous bits for F_i in *CPO(vec(j))* via any of these error simulations.

Theorem 4: A set of *N* nodes is the *N*-correctable nodes if all erroneous vectors are *OEVs* or *C-OEVs* for this set.

To identify *N*-correctable nodes, we derive the sensitization set for every node in the fanin cones of every erroneous F_i. However, the error simulation process is very time consuming if we examine every set of *N* nodes in C_I. Thus, we propose a heuristic to screen out a large number of sets in C_I before performing the *OEV* and *C-OEV* detections.

Before introducing our heuristic, we first discuss the results produced from the sensitization set derivation process. For every erroneous bit F_i*-blist(j)* in an erroneous output F_i, the sensitization set of a node *l*, *SEN(l,*F_i*)*, falls into one of the following categories:

- Every corresponding *l-blist(j)* is in *SEN(l,*F_i*)*. Hence we say that the erroneous bits of F_i are *fully covered* by *l*.
- Part of the corresponding *l-blist(j)* are in *SEN(l,*F_i*)*. Hence we say that the erroneous bits of F_i are *partially covered* by *l*.
- Every corresponding *l-blist(j)* is not in *SEN(l,*F_i*)*, Hence we say that the erroneous bits of F_i are *emptily covered* by *l*.

Heuristic 2: Only the set of *N* nodes that simultaneously satisfies the following two conditions will be further examined by the *OEV* and *C-OEV* detections:

1) For every erroneous output F_i, there exists at least one node l in the set such that all the erroneous bits of F_i are fully covered by l or, there exists at least two nodes l_j in the set such that all the erroneous bits of F_i are partially covered by each l_j.
2) For every node l_j in the set, there exists at least one erroneous output F_i such that all the erroneous bits of F_i are fully covered or partially covered by l_j.

Node	SEN(l_j, F_1)	SEN(l_j, F_2)
X_1	{6, 5}	∅
X_2	{7, 5}	{8 ~ 5}
X_3	{5, 1}	{6, 5, 2, 1}
A	{6, 5}	∅
B	{5 ,1}	{6, 5, 2, 1}
C	{5, 1}	{8 ~ 1}
D	{5, 1}	{8 ~ 1}
E	{6, 5}	∅
B1	{5}	∅
B2	{5}	{8 ~ 5}
B3	{5, 1}	∅
B4	∅	{8 ~ 1}
F_1	{8 ~ 1}	∅
F_2	∅	{8 ~ 1}

Fig. 4. The sensitization sets of every node in the circuit of Fig. 3.

Since we use a large number of vectors to simulate C_I at a time, the probability that all the erroneous vectors are identified as the *C-OEVs* for *N*-correctable nodes is very low. In other words, for *N*-correctable nodes, some of the erroneous vectors also can be identified as the *OEVs*. Therefore, we approximate the correction ability of a set of *N* nodes by observing the sensitization sets of them. For an erroneous F_i, F_i can be corrected by only re-synthesizing a node among the *N*-correctable nodes; otherwise, F_i may need to be corrected by re-synthesizing at least two nodes among the *N*-correctable nodes. Thus, as long as the erroneous bits of F_i are fully covered by a node or partially covered by at least two nodes, F_i has a higher probability to be corrected. For the same reason, we also set the constraint for a node *l* that must have at least one *l-blist(j)* is a sensitized bit with respect to the erroneous bit F_i*-blist(j)* in F_i.

For example, again, in Fig. 3 and Fig. 4, we will not choose the set (*D,* F_2) to perform the *OEV* and *C-OEV* detections because all the erroneous bits of F_1 are only partially covered by *D*. It is obvious that F_1 cannot be corrected by only re-synthesizing *D*. Moreover, all the erroneous bits of F_1 are emptily covered by F_2. Therefore, (*D,* F_2) is pruned. However, all the erroneous bits of F_1 are fully covered by F_1, and all the erroneous bits of F_2 are also fully covered by *D*. It indicates that F_1 and F_2 are possibily to be corrected by re-synthesizing F_1 and *D*, respectively. Therefore, the set (*D,* F_1) can be further examined by the *OEV* and *C-OEV* detections.

We further apply two-stage technique [5] to speed up the diagnosis process. In the first stage, we only consider the sets of *N* nodes that are all not dominated by any other node. These sets are called *key node sets*, and they are examined by Heuristic 2 and the *OEV*, *C-OEV* detections sequentially. In the second stage, only the subordinate sets of the survived key node sets in the first stage are examined. The nodes in the subordinate sets are all dominated by the nodes in the survived key node sets respectively.

Example 2 illustrates the multiple error diagnosis process.

Example 2: In Fig. 3, there are two error sources in the circuit. The nodes that are not dominated by any other node are marked with the black dots. We generate the key node sets from these nodes. These key node sets are examined in the first stage. Based on Heuristic 2, the key node sets, (*D,* F_1), (F_1, F_2), and (*D,* X_2) are examined by the *OEV* and *C-OEV* detections. Only (*D,* F_1) and (F_1, F_2) are survived. Therefore, in the second stage, the subordinate sets are generated from (*D,* F_1) and (F_1, F_2). For example, the nodes, {*C,* B_2, *B,* X_3}, are dominated by *D*, and the nodes, {*E,* B_3, *A,* B_1, X_1}, are dominated by F_1. Therefore, the subordinate sets of (*D,* F_1) are (*D, E*), (*D,* B_3), ..., (*C,* F_1), (*C, E*), ..., (X_3, B_1), (X_3, X_1), 29 sets in total. Again, the previous examined procedures are performed on these subordinate sets. Finally, the diagnosis process returns the sets that pass the *OEV* and *C-OEV* detections in the first and second stages. Those sets are the error candidates.

TABLE I

EXPERIMENTAL RESULTS ON SINGLE ERROR DIAGNOSIS.

Circuits	Error space	# of vectors	# of single correctable nodes	Suspect ratio (%)	CPU time (Sec.)
C432	596	8192	7.4	1.24	0.18
C499	1244	8192	5.4	0.43	0.8
C880	956	8192	6.4	0.67	0.12
C1355	1562	8192	7.1	0.45	1.38
C1908	1654	8192	6.5	0.39	0.79
C2670	2490	16384	7.6	0.31	0.29
C3540	3237	16384	5.3	0.16	1.81
C5315	5317	16384	8.9	0.17	1
C6288	7639	1024	2.9	0.04	4.28
C7552	7005	16384	11.6	0.17	3.22
des	12457	16384	6.1	0.05	2.49
m32×32	24012	1024	3.3	0.01	78.94

TABLE II

EXPERIMENTAL RESULTS ON TWO-ERROR DIAGNOSIS.

Circuits	Error space	# of vectors	$N1$	Pass	$N2$	Pass	$N3$	Suspect ratio(%)	$T1$	$T2$
C432	1.77E5	8192	1396.6	1.6	131.6	14.1	15.7	8.87E-3	3.25	3.65
C499	7.73E5	8192	3417.5	1	343.1	30.1	31.1	4.02E-3	19.92	22.45
C880	4.56E5	8192	113.5	1.2	416	18.6	19.8	4.34E-3	0.33	0.89
C1355	1.22E6	8192	2974.2	1.2	460.5	45.9	47.1	3.86E-3	21.83	26.32
C1908	1.37E6	8192	1505	1.8	80	46.9	48.7	3.55E-3	9.64	11.86
C2670	3.1E6	16384	80.9	1.5	528.2	41	42.5	1.37E-3	5.06	6.77
C3540	5.24E6	16384	958.1	1.4	42.5	34.5	35.9	6.85E-4	17.21	24.71
C5315	1.41E7	16384	143.9	1.1	130	66.3	67.4	4.78E-4	2.28	11.52
C6288	2.92E7	1024	10786.3	1.3	18.4	6.9	8.2	2.81E-5	85.07	117.59
C7552	2.45E7	16384	103	2.9	84.7	84.4	87.3	3.56E-4	5.82	15.49
des	7.76E7	16384	36.1	1.7	29.1	29.1	30.8	3.96E-5	3.52	14.79
m32×32	2.88E8	1024	7472.3	1.1	186	13.4	14.5	5.03E-6	299.6	587.72

$N1$: # of sets examined in the first stage. $N2$: # of sets examined in the second stage.
$N3$: # of two-correctable nodes.
$T1$: Error simulation time (Sec). $T2$: Total CPU time (Sec).

IV. EXPERIMENTAL RESULTS

We implemented our algorithm in C language within SIS [10] environment. The experiments are conducted over the *ISCAS'85* and two large benchmarks, *des*, a 32×32 multiplier circuit on a 1280MHz Sun Blade 2500 machine with 4GB memory. These benchmarks are in BLIF format. Each circuit is optimized by using *"script.rugged"* script, and then decomposed into 2-input AND, OR, and NOT gates by mapping to the SIS library (22-1.genlib). To obtain an erroneous implementation, we randomly inject errors to the circuit. The error models we used are *missing inverter*, *extra inverter*, *gate replacement*, and *incorrectly placed wire*. Note that these four error models introduced are just for the convenience of our experiments. Our approach is error model-free.

We show the results of single and two-error diagnosis in Table I and Table II, respectively. We run each circuit 10 times and measure the average results. Note that when the number of injected errors N > 2, our approach can work as well. However, due to large error space, it requires more computation time. Here we omit these diagnosis results.

In both Table I and Table II, the first column contains the circuit names, the last one *m32×32* is a 32 × 32 multiplier designed by us. The second column shows the error space of each circuit which can be calculated by the following equation.

$$\textit{Error Space} = \sum_{i=1}^{n} C_n^m \quad (1)$$

Where m is the number of nodes in the erroneous implementation, n is the number of nodes chosen from m nodes. The number of random vectors we used is shown in the third column.

In Table I, the column 4 shows the average number of single correctable nodes returned by our approach in 10 experiments. In the column 5, the suspect ratio obtained by *(# of single correctable nodes) / (error space)* indicates our approach prunes most of nodes in a circuit. The final column shows the average CPU time for this diagnosis process.

In Table II, each of the erroneous implementation is run single error diagnosis process first and proven containing no single correctable node. Since we apply the two-stage technique, the column 4 shows the average number of key node sets examined by the *OEV* and *C-OEV* detections in the first stage. The next column shows the average number of the survived key node sets. We can see that only a few sets are passed to the second stage. Similarly, the next two columns show the average number of subordinary sets of the survived key node sets examined and passed. The column 8 shows the average number of two-correctable nodes in each circuit. The suspect ratio in the column 9 indicates our approach returns a small set of the candidates as compared to the error space. The last two columns show the average CPU time used for the error simulation and the total CPU time respectively. The total CPU time also includes the execution time on the examination of single error.

In our experiments, the actual error nodes are always included in the set of single or two-correctable nodes returned by our approach. In practice, the locations of actual error sources are unknown. We identify the actual error nodes for the sake of demonstrating the accuracy of our approach. The memory usage in our experiments does not exceed 200MBytes.

V. CONCLUSIONS

We presented a parallel random vector simulation-based approach for locating multiple errors in an erroneous implementation. We determine a set of *N* nodes which is the most responsible for correcting the erroneous implementation by examining whether all erroneous vectors are observable or co-observable erroneous vectors. Two heuristics are also proposed to speedup our approach without sacrificing the accuracy. Experimental results show the efficiency and the effectiveness of our approach. In addition, the experiments on *des* and *m32×32* benchmarks show that our approach is also applicable to large circuits without efficient BDD representations.

REFERENCES

[1] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical path tracing - an alternative to fault simulation," in *Proc. 20th Conf. on Design Automation*, pp. 214-220, 1983.

[2] M. Ali, S. Safarpour, A. Veneris, M. Abadir, and R. Drechsler, "Post-verification debugging of hierarchical designs," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 871-876, 2005.

[3] M. Ali, A. Veneris, S. Safarpour, R. Drechsler, A. Smith, and M. Abadir, "Debugging sequential circuits using Boolean satisfiability," in *Proc. Int. Conf. on Computer-Aided Design*, pp. 204-209, 2004.

[4] P. - Y. Chung, Y. - M. Wang, and I. N. Hajj, "Diagnosis and correction of logic design errors in digital circuits," in *Proc. Design Automation Conf.*, pp. 503-508, June. 1993.

[5] S. - Y. Huang, K. - T. Cheng, K. - C. Chen, and D. - I. Cheng "ErrorTracer: A fault simulation-based approach to design error diagnosis," in *Proc. Int. Test Conf.*, pp. 974-981, Feb. 1997.

[6] A. Kuehlmann, D. I. Cheng, A. Srinivasan, and D. P. LaPotin, "Error diagnosis for transistor-level verification," in *Proc. Design Automation Conf.*, pp. 218-224, June. 1994.

[7] C. - C. Lin, K. - C. Chen, S. - C. Chang, M. Marek - Sadowska, and K. - T. Cheng, "Logic synthesis for engineering change," in *Proc. Design Automation Conf.*, pp. 647-652, June. 1995.

[8] C. - C. Lin, K. - C. Chen, D. - I. Cheng, and M. Marek - Sadowska, "Logic rectification and synthesis for engineering change," in *Proc. Asian and South Pacific Design Automation Conf.*, pp. 301-309, 1995.

[9] I. Pomeranz and S. M. Reddy, "On correction of multiple design errors," *IEEE Trans. CAD.*, vol. 14, pp. 255-264, Feb. 1995.

[10] E. M. Sentovich, K. J. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. R. Stephan, R. K. Brayton, and A. Sangiovanni - Vincentelli, "SIS: A system for sequential circuit synthesis," Technical Report UCB/ERL M92/41, Electronics Research Lab, Univ. of California, Berkeley, CA 94720, May. 1992.

[11] A. Smith, A. Veneris, and A. Viglas, "Design diagnosis using Boolean satisfiability," in *Proc. Asian and South Pacific Design Automation Conf.*, pp. 218-223, 2004.

[12] M. Tomita, T. Yamamoto, F. Sumikawa, and K. Hirano "Rectification of Multiple Logic Design Errors in Multiple Output Circuits," in *Proc. Design Automation Conf.*, pp. 212-217, June. 1994.

[13] A. Veneris and I. N. Hajj, "A fast algorithm for locating and correcting simple design errors in VLSI digital circuits," in *Proc. Great Lake Symp. on VLSI Design*, pp. 45-50, Mar. 1997.

[14] A. Veneris and I. N. Hajj, "Design error diagnosis and correction via test vector simulation," *IEEE Trans. CAD.*, vol. 18, no. 12, pp. 1803-1816, Dec. 1999.

[15] A. Veneris, J. - B. Liu, M. Amiri, and M. S. Abadir, "Incremental Diagnosis and Correction of Multiple Faults and Errors," *Proc. Design, Automation and Test in Europe Conference and Exhibition*, pp. 716-721, 2002.

[16] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom, and T. McGarthy, "Fault simulation for structured VLSI," in *VLSI Syst. Design*, pp. 20-32, Dec. 1985.

[17] S. - C. Wu and C. - Y. Wang, "PEACH: A Novel Architecture for Probabilistic Combinational Equivalence Checking," in *VLSI-SoC*, pp.104-109, Oct. 2006.