Upper Bounding Fault Coverage by Structural Analysis and Signal Monitoring

Abstract

A new algorithm for determining stuck faults in combinational circuits that cannot be detected by a given input sequence is presented in this paper. Other than pre and post-processing steps, certain signal conditions are monitored during logic simulation. These signal conditions are determined by dominator and signal reconvergence analysis. After simulation, a post-processing step determines faults that cannot be detected by the sequence. For combinational ISCAS benchmarks, the runtime overhead for the algorithm is found to be around 30-40%. Experimental data show a substantial reduction of error in estimates obtained by a stuck-fault coverage estimator that uses this algorithm to eliminate faults that are guaranteed to remain undetected by the given sequence.

I. Introduction

Upper bounding of fault coverage involves the identification of stuck faults that are guaranteed to remain undetected by a vector sequence. There are several applications of upper bounding to various problems in test. If such upper bounds are obtained with resources comparable to that of logic simulation, fault simulation can be speeded up by removing such faults from the list that is explicitly simulated. Another application of upper bounding is test selection where tests have to be selected from a larger pool such that the selected tests can detect a specified set of faults. A third application of interest is fault coverage estimation, where stuck fault coverage has to be estimated quickly with a high degree of accuracy. Upper bounding can reduce the errors obtained by an estimator by eliminating faults that would otherwise be incorrectly estimated to be detected. In this paper, we present a new algorithm for obtaining such upper bounds, and demonstrate its applicability to fault coverage estimation.

Similar algorithms have been proposed in the literature [1], [2]. However, both of these algorithms require circuit analysis after simulation of each vector. Moreover, critical path tracing [1] underestimates fault coverage when a stem fault is sensitized by simultaneous propagation of fault effects on all fanout branches, while none of the branch faults individually are detected (see example in Figure 2(a)). The work that is closest to the research presented here is that of Akers et al [2]. While similar to critical path tracing, this algorithm avoids tracing through gates with multiple dominant input logic values and are points of reconvergence of paths with opposite parities. However, at points of reconvergence with identical parities, the algorithm is exhaustive and repeats the analysis for each possible choice of input that can justify the output. Unlike critical path tracing and the work presented here, this algorithm does not analyze dominators that may exist in the circuit graph structure.

Some basic definitions are presented in Section II. We illustrate our algorithm with an example in Sections III and V, while the algorithm itself is presented in Section IV. Section VI presents experimental results.

II. Definitions

A *signal* is a single line in the circuit, and can be identified with the output of a gate if it does not fanout to multiple gates. At fanout stems, each fanout branch is a signal and is considered different from every other fanout branch and the stem itself. A circuit can be modeled as a directed graph where nodes of the graph represents gates, and edges correspond to signals in the circuit. Circuits and graphs are used interchangeably in this paper. The *checkpoints* of a circuits consist of primary inputs and fanout gates.

In any given circuit, each gate can be classified as one or more of three categories: *fanout, reconvergent* and *non-reconvergent*. Gates that drive a fanout stem are referred to as fanout gates. Gates that are a point of reconvergence constitute the second category, while non-fanout gates that are not a point for reconvergence constitute the third category.

Note that these three classes are not exclusive. A fanout gate can also be a reconvergent gate.

In a directed graph, node *A* *dominates* node *B* if every path from *B* to any output passes through *A* [3]. The set of nodes that dominates a given node is called a dominator set. Our definition of dominators is slightly different from the graph theoretic sense. Even though every node dominates itself, we exclude such considerations in this paper. In addition, a dominating gate must have at least one input that is not reachable from the dominated gate. This eliminates dominators that exist due to reconvergent fanouts only. As an example, consider the cone driving gate $G5$ in Figure 1, and gate $G11$. $G5$ is not considered a dominator for $G11$ because it has no input that is not reachable from $G11$. The only dominator for $G11$ is $G1$. If node *A* has a single fanout node *B*, *B* may dominate *A* (depending on whether *B* has inputs other than *A*). A given node is said to have a *trivial dominator set* if its only immediate fanout is its only dominator. A *non-trivial dominator set* necessarily includes at least one dominating node that is not an immediate fanout point of the given node.

The *output sensitizing condition* for a gate and its dominator set denotes the condition when the gate output is at a specified value and all other off-path sensitizing values for its dominators have appropriate non-controlling values. The *input sensitizing condition* with respect to a specific input pin of a gate and its dominator set consists of the specified input having a specified value with all other inputs to the gate and off-path inputs to its dominators having non-controlling sensitizing values.

III. An Example

We consider a small circuit (c17) and a sequence of four vectors, and outline how the algorithm determines an upper bound on fault coverage. We also assume fault collapsing has been performed, and consider the collapsed fault set in our analysis. There are three distinct steps to upper bounding fault coverage: (1) a preprocessing step consisting of structural analysis, (2) monitoring of signal conditions during simulation and (3) post-processing of results and identification of undetectable faults. The first two stages are outlined in this section. The final step for this example is presented in Section V after the algorithm is presented in Section IV.

A. Structural Analysis

Structural graph analysis for finding dominators [4] and reconvergence points [5], [6], [7] is common in test generation systems. In our system, several signal conditions are monitored at the end of each cycle in a vector sequence. These conditions are derived for single output circuits. For multi-output circuits, these conditions are obtained from the cones of logic that drive each output. These signal conditions are necessary for fault detection. If some of these conditions never occur in any cycle, certain faults are bound to be remain undetected in the sequence.

For each gate, all input value combinations are stored in a table. These input combinations are analyzed once simulation of the entire sequence is complete. Along with gate input combinations, each logic cone is analyzed separately and dominators for each *non-fanout* gate to the cone output are obtained. Such dominators found for non-fanout gates are either trivial or non-trivial. For the circuit of Figure 1, gate $G0$ has a trivial dominator set $\{G4\}$, while $G6$ has a non-trivial set $\{G0, G4\}$. The other dominated gates are $G3$ and $G12$, with dominating sets that are $\{G5\}$ (trivial) and $\{G3, G5\}$ (non-trivial) respectively. Since this dominator analysis is performed for each cone, additional dominators are found for gate $G1$ and $G2$. For the cone driving $G4$, $G2$ has a trivial dominator set $\{G4\}$, while $G1$ has a non-trivial dominator set $\{G2, G4\}$. In the cone driving $G5$, $G2$ has a trivial dominator set $\{G5\}$.

For gates with trivial dominator sets, input sensitizing conditions are monitored explicitly, unless an input is a checkpoint. For non-trivial dominators, only output sensitizing conditions are monitored. Output sensitizing conditions from trivial dominators are ignored because they are obtained automatically by monitoring input states for the dominating gate. Going back to the circuit of Figure 1, all inputs for both $G0$ and $G3$ are checkpoints and are skipped. For $G6$, the output sensitizing condition is the simultaneous occurence of {G6=v, G10=1, G2=1} for $v \in \{0, 1\}$. For $G12$, the non-trivial condition consists of {G12=v, G1=1, G2=1} for $v \in \{0, 1\}$. In the cone driving $G4$, the trivial dominator set for $G2$ yields two conditions for its second input: {G1=v, G9=1,G0=1} for $v \in \{0, 1\}$. These conditions are not skipped because $G1$ is not a checkpoint in this cone. The non trivial dominator condition for $G9$ yields

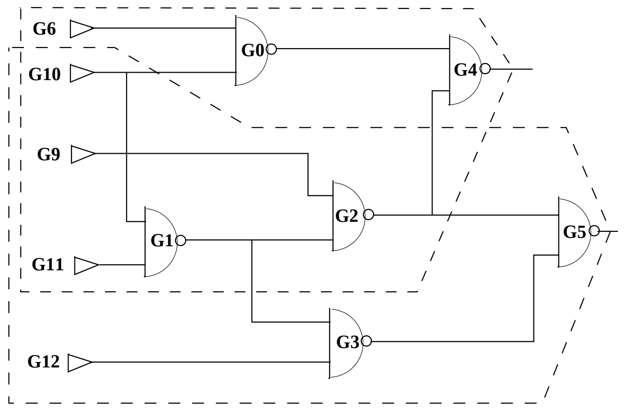

Fig. 1. Cone and dominator example

$\{$G9=v, G1=1, G0=1$\}$. For the cone driving $G5$, gates $G2$ and $G9$ have dominator sets $\{G5\}$ and $\{G2, G5\}$. Both inputs for $G2$ are checkpoints for the cone and are ignored. The output sensitizing conditions for $G9$ and its dominator set are $\{$G9=v, G1=1, G3=1$\}$, for both $v \in \{0, 1\}$. The following table lists the conditions that are obtained for both signal values at each gate. Multiple conditions at a gate are shown with an index in square brackets ([]).

$dom0(G6)[1]$	=	{G6=0,G10=1, G2=1}
$dom0(G12)[1]$	=	{G12=0, G1=1, G2=1}
$dom0(G9)[1]$	=	{G9=0,G1=1, G3=1}
$dom0(G9)[2]$	=	{G9=0, G1=1, G0=1}
$dom0(G1)[1]$	=	{G1=0, G9=1, G0=1}
$dom0(G10)[1]$	=	{G10=0, G11=1, G9=1, G3=1}
$dom1(G6)[1]$	=	{G6=1,G10=1, G2=1}
$dom1(G12)[1]$	=	{G12=1, G1=1, G2=1}
$dom1(G9)[1]$	=	{G9=1,G1=1, G3=1}
$dom1(G9)[2]$	=	{G9=1, G1=1, G0=1}
$dom1(G1)[1]$	=	{G1=1, G9=1, G0=1}
$dom1(G10)[1]$	=	{G10=1, G11=1, G9=1, G3=1}

A dominator set ($\{$G1, G2, G4$\}$) exists for gate $G11$ in the cone driving $G4$. However, no dominators exist for this gate in the cone driving $G5$. If no conditions are obtained for a gate from a cone, all conditions for that gate obtained from other cones are also dropped from further consideration. Gates $G10$ and $G1$ also have conditions from one cone only. However, these are not dropped because they are fanout points in the other cone, and additional conditions due to reconvergence in the other cone are generated, as explained below.

Fanout gates are analyzed specific to each output cone that contain them. These fanout gates in a cone are origination points for reconvergent paths. Some fanout gates in the circuit may have no fanouts in each individual cone and are not considered in this step, eg. $G2$. Different reconvergent paths may have opposite parities and fault propagation requires paths with opposite parities to be disabled. Moreover, along paths that have the same parity, simultaneous fault effect propagation may also occur.

For the cone driving $G4$, $G10$ has reconvergent paths with opposite parity. We denote the two sensitizing conditions for paths originating at $G10$ by $sp0(G10)[1]$ ($sp1(G10)[1]$) and $sp0(G10)[2]$ ($sp1(G10)[2]$) for logic value 0 (1).

$$\begin{array}{r}
sp0(G10)[1] = \{G10=0, G6=1, G2=1, G9=0 || G11=0\} \\
sp0(G10)[2] = \{G10=0, G11=1, G9=1, G0=1, G6=0\} \\
\hline
sp1(G10)[1] = \{G10=1, G6=1, G2=1, G9=0 || G11=0\} \\
sp1(G10)[2] = \{G10=1, G11=1, G9=1, G0=1, G6=0\}
\end{array}$$

For $sp0(G10)[1]$, $\{G9 = 0 || G11 = 0\}$ represents the disabling condition for the propagation path with opposite parity. For the cone driving $G5$, there are reconvergent paths with same parity from $G1$. Since it is possible to activate any subset of these paths, including all of them simultaneously, the following conditions are derived for $G1$:

$sp0(G1)[1]$	=	$\{G1=0, G9=1, G3=1\}$
$sp0(G1)[2]$	=	$\{G1=0, G12=1, G2=1\}$
$mp0(G1)[1]$	=	$\{G1=0, G9=1, G12=1\}$
$sp1(G1)[1]$	=	$\{G1=1, G9=1, G3=1\}$
$sp1(G1)[2]$	=	$\{G1=1, G12=1, G2=1\}$
$mp1(G1)[1]$	=	$\{G1=1, G9=1, G12=1\}$

For a given gate G, if none of the conditions $dom0(G)$, $sp0(G)$ or $mp0(G)$ ($dom1(G)$, $sp1(G)$ or $mp1(G)$) are ever satisfied during simulation, the $s@1$ ($s@0$) fault at the output of G will remain undetected. In addition, all faults whose propagation require gate G to have a value 0 (1) also remain undetected.

B. Monitoring During Simulation

Figure 2 shows a four vector simulation sequence of the example circuit, along with specific vectors where these conditions are *first* satisfied. Also shown in the figure are the gate input combinations that are observed after simulation of each vector. Once simulation of a subsequence

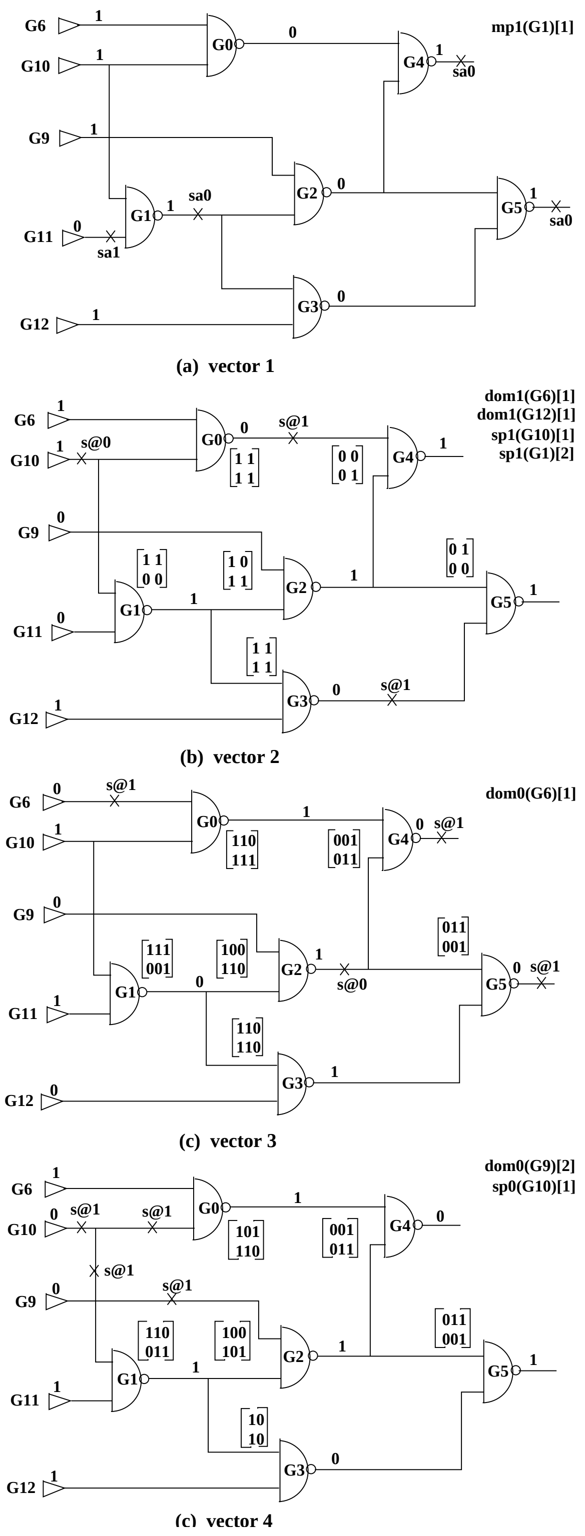

Fig. 2. Four vector simulation for example

is complete, the set of input value combinations for each gate and the conditions of Section III-A that are satisfied are analyzed and faults that are guaranteed not to be detected are obtained. The algorithm for the analysis of these input value combinations and the satisfied dominator conditions is presented in Section IV. The specifics for this example are presented in Section V.

IV. Algorithm for Post-processing

For node n, we assume signal conditions $dom0(n)$, $dom1(n)$, $sp0(n)$, $sp1(n)$, $mp0(n)$ and $mp1(n)$ are monitored during simulation. Note these conditions may not exist for all signals, in which case they are assumed to be trivially true (1). When any of these conditions do exist, and is not satisfied during simulation, the condition is assigned a value false (0).

For a given node n, we evaluate two predicates, $oneProp(n)$ and $zeroProp(n)$, that denotes the possibility of logic values 1 and 0, respectively, to propagate from n to some primary output. For an output that attains a logic value 1 (0) sometime during simulation, $oneProp(n)$ ($zeroProp(n)$) is assigned a value 1. Otherwise, this value is set to 0. Using backward traversal from the outputs, these predicates are evaluated at gate inputs. We next describe how this backward traversal is performed from circuit outputs to primary inputs.

Input value combinations at each gate are also stored for each gate. We denote by $\rho(n = 1)$ ($\rho(n = 0)$) the condition that node n attains the value 1 (0) during simulation. For signals $s_1, s_2, \ldots, s_N$, the predicate $\rho(s_1 = v_1, s_2 = v_2, \ldots, s_N = v_N)$ denotes the condition that signals $s_i = v_i, 1 \leq i \leq N$ simultaneously, where these signals are assumed to be inputs of a gate. The predicate ρ will be referred to as the reachability predicate. Since this algorithm analyzes each cone of the circuit separately, all fanout stems in a cone are necessarily reconvergent. We show how $oneProp(s_i)$ and $zeroProp(s_i)$ are evaluated given the values of these predicates at the output of the gate s_o. Without loss of generality, we assume the gate to be an AND gate. The formulae are similar for other gate types.

TABLE I

PROPAGATION PREDICATE VALUES FOR LAST THREE VECTORS

	Cone of G4						Cone of G5					
	zeroProp			oneProp			zeroProp			oneProp		
signal (1)	v2 (2)	v3 (3)	v4 (4)	v2 (5)	v3 (6)	v4 (7)	v2 (8)	v3 (9)	v4 (10)	v2 (11)	v3 (12)	v4 (13)
G4	0	1	1	1	1	1	-	-	-	-	-	-
G5	-	-	-	-	-	-	0	1	1	1	1	1
G0	1	1	1	0	1	1	-	-	-	-	-	-
G2	0	0	0	0	1	1	0	0	0	0	1	1
G3	-	-	-	-	-	-	1	1	1	0	1	1
G1-G2	-	-	-	-	-	-	0	0	0	0	0	0
G1-G3	-	-	-	-	-	-	0	0	0	1	1	1
G1	0	0	0	0	0	0	0	0	0	1	1	1
G11	0	0	0	0	0	0	1	1	1	0	0	0
G9	0	0	1	0	0	0	0	0	0	0	0	0
G10-G0	0	0	1	1	1	1	-	-	-	-	-	-
G10-G1	0	0	0	0	0	0	-	-	-	-	-	-
G10	0	0	1	1	1	1	0	0	0	0	0	0
G6	0	1	1	1	1	1	-	-	-	-	-	-
G12	-	-	-	-	-	-	0	0	0	1	1	1

A. Propagation at Non-Fanout Inputs

If input s_i of gate, with output s_o, is not a fanout stem, we evaluate $localOneProp(s_i)$ from $oneProp(s_o)$ as

$$localOneProp(s_i) = oneProp(s_o) \wedge \rho(s_i = 1, \ldots, s_N = 1)$$

and finally

$$oneProp(s_i) = localOneProp(s_i) \wedge dom1(s_i)$$

Similarly,

$$localZeroProp(s_i) = zeroProp(s_o) \wedge \rho(s_i = 0, \ldots, s_N = 1)$$

$$zeroProp(s_i) = localzeroProp(s_i) \wedge dom0(s_i)$$

B. Propagation at Stem Inputs

Fanout stems can vary depending on the parities of reconvergent paths. If reconvergent paths have opposite parities, fault propagation cannot occur simultaneously because such fault effects cancel out at the point of reconvergence. For paths that have identical parities, simultaneous fault effect propagation along multiple paths is possible and needs to be modeled.

1) Equal Parity Reconvergence: As in Section IV-A, for each branch b_i, predicates $localOneProp(b_i)$ and $localZeroProp(b_i)$ are evaluated. At the stem s,

$$localOneProp(s) = \bigvee localOneProp(b_i), 1 \leq i \leq N$$

$$localZeroProp(s) = \bigvee localZeroProp(b_i), 1 \leq i \leq N$$

where stem s is assumed to have N branches. Assume all fanout branches reconverge at node r with the same parity as the stem s.

$$oneProp(s) = localOneProp(s) \bigvee (oneProp(r) \wedge mp1(s))$$

$$zeroProp(s) = localZeroProp(s) \bigvee (zeroProp(r) \wedge mp0(s))$$

If all fanout branches reconverge at r with parity opposite to that of s,

$$oneProp(s) = localOneProp(s) \bigvee (zeroProp(r) \wedge mp1(s))$$

$$zeroProp(s) = localZeroProp(s) \bigvee (oneProp(r) \wedge mp0(s))$$

2) Unequal Parity Reconvergence: For each branch b_i, predicates $localOneProp(b_i)$ and $localZeroProp(b_i)$ are evaluated as in Section IV-A. At the stem s,

$$oneProp(s) = \bigvee_{i=1}^{N} (localOneProp(b_i) \wedge sp0(b_i))$$

TABLE II

DETECTABILITY OF INDIVIDUAL FAULTS

id	location	type	det vector
1	$G4$	$s@0$	1
2	$G5$	$s@0$	1
3	$G1$	$s@0$	1
4	$G11$	$s@1$	1
5	$G0$	$s@1$	2
6	$G3$	$s@1$	2
7	$G10$	$s@0$	2
8	$G4$	$s@1$	3
9	$G6$	$s@1$	3
10	$G5$	$s@1$	3
11	$G2$	$s@0$	3
12	$G9$	$s@1$	4
13	$G10 - G0$	$s@1$	4
14	$G10 - G1$	$s@1$	4
15	$G10$	$s@1$	4
16	$G2 - G4$	$s@1$	UND
17	$G2 - G5$	$s@1$	UND
18	$G2$	$s@1$	UND
19	$G1 - G2$	$s@1$	UND
20	$G1 - G3$	$s@1$	UND
21	$G1$	$s@1$	UND
22	$G12$	$s@1$	UND

$$zeroProp(s) = \bigvee_{i=1}^{N} (localZeroProp(b_i) \wedge sp1(b_i))$$

where stem s is assumed to have N branches.

C. Deducing Undetectability

A $s@0$ fault, f, at signal s is considered undetected is either the fault was not excited or the logic value 1 cannot be propagated to a primary output. Therefore,

$$detectability(f) = \rho(s = 1) \wedge oneProp(s)$$

Similarly, for a $s@1$ fault f,

$$detectability(f) = \rho(s = 0) \wedge zeroProp(s)$$

V. EXAMPLE: POST-PROCESSING

Returning to the example of Figure 2, we consider how the algorithm of Section IV computes an upper bound on fault coverage. After simulation of the first vector, knowing the possible values at the two outputs, the values of $oneProp$ ($zeroProp$) at $G4$ and $G5$ are evaluated to 1 (0). Tracing back from $G4$ ($G5$), both $zeroProp$ and $oneProp$ at gate inputs are evaluated to 0 because the only input state seen at these gates is 00. This backward tracing continues at gates $G0$, $G2$ and $G3$, with identical results. However, at the fanout stem driven by $G1$, the predicate $mp1(G1)$ evaluates to 1. Using the formulae presented in Section IV-B.1, $oneProp(G1)$ ($zeroProp(G1)$) evaluates to 1 (0). Tracing backwards from $G1$, $zeroProp(G11)$ ($oneProp(G11)$) evaluates to 1 (0). Using these values of $zeroProp$ and $oneProp$, and the formulae presented in Section IV-C, only the faults shown in Figure 2(a) are estimated to be detected.

For each of the remaining three vectors, the values of the propagation predicates are shown in Table I. This table shows the values computed for each logic cone separately, in columns (2)-(7) and (8)-(13) respectively. Note that $G2$ has no fanout in either of the logic cones. Considering vector 2 and the cone driving $G4$ (columns 2 and 5), the $zeroProp$ predicate at $G4$ evaluates to 0 because this output never goes to 1 in the first two vectors. Tracing back from $G4$, only the $zeroProp$ predicate at $G0$ evaluates to 1 because the two input states at $G4$ are 00 and 01. Due to this same reason, both predicates for $G2$ are also 0. Tracing back from $G2$, all predicates for all logic driving $G2$ are also 0, including signals $G1$, $G11$, $G9$, and *G10-G1*. Tracing back from $G0$, $oneProp$ at $G6$ and signal *G10-G0* evaluates to 1. In the cone of $G5$, only the $zeroProp$ predicate at $G3$ evaluates to 1. Note that $oneProp$ evaluates to 1 for signal *G1-G3* and stem $G1$ also. The $s@0$ fault on the stem $G1$ is assumed to be detected by the first two vectors by this analysis correctly. However, as explained in the previous paragraph, this fault is also correctly estimated to be detected after the first vector also. Predicate $oneProp$ evaluates to 1 for $G12$ also. The $s@0$ at $G12$ is estimated to be detected by this analysis. However, this fault is equivalent to the $s@1$ at $G3$, and is not shown in Figure 2(b).

After simulation of vector 3, the values of the predicates are shown in columns (3), (6), (9) and (12). In the cone driven by $G4$, $zeroProp$ evaluates to 0 for $G9$ because both $dom0(G9)[1]$ and $dom0(G9)[2]$ remain unsatisfied till the third vector. However, the second condition is found to be satisfied in vector 4, and this predicate evaluates to 1 for $G9$ subsequently (column 4 of Table I). The remaining values of these predicates obtained after both vectors 3 and 4 are shown in that table and is left to the reader.

TABLE III

UPPER BOUNDING RESULTS

ckt (1)	exact cov (%) (2)	vanilla est (%) (3)	upper bound (%) (4)	bounded est (%) (5)	lsim time (s) (6)	lsim with monitoring (s) (7)	overhead (%) (8)
c432	93.15	96.95	94.47	93.32	0.03	0.04	33.3
c880	91.08	93.42	92.99	92.04	0.05	0.06	20.0
c1355	87.93	88.88	88.82	88.82	0.09	0.11	22.2
c1908	69.79	82.90	73.15	73.10	0.13	0.17	30.8
c2670	77.28	85.59	80.04	78.56	0.19	0.26	36.8
c3540	70.79	76.32	72.02	71.99	0.26	0.34	30.8
C5315	91.74	96.02	96.39	93.64	0.43	0.58	34.9
C6288	99.74	99.95	99.74	99.74	1.89	2.20	16.4
C7552	84.47	93.67	88.06	87.99	0.67	1.02	52.2

TABLE IV

ERROR ANALYSIS

		initial estimate			final estimate		
ckt (1)	exact cov (%) (2)	est (%) (3)	overshoot (%) (4)	undershoot (%) (5)	est (%) (6)	overshoot (%) (7)	undershoot (%) (8)
c432	93.15	96.95	4.96	1.15	93.32	1.34	1.15
c880	91.08	93.42	3.82	1.49	92.04	1.70	0.74
c1355	87.93	88.88	0.95	0.00	88.82	0.89	0.00
c1908	69.79	82.84	13.2	0.15	73.10	3.36	0.05
c2670	77.28	84.60	8.50	1.18	78.56	2.15	0.88
c3540	70.79	78.31	7.95	0.43	71.99	1.23	0.03
C5315	91.74	96.02	5.55	1.27	93.64	3.15	1.25
C6288	99.74	99.95	0.21	0.00	99.74	0.00	0.00
C7552	84.47	93.67	9.50	0.30	87.99	3.56	0.04

Table II shows the detection status for each fault after simulation of each vector. The post-processing algorithm of Section IV was executed after each vector. If post-processing is performed at the end of the last vector only, any fault shown to be detected by an earlier vector is also estimated to be detected by some earlier vector in the sequence. Table II shows the faults that remain undetected after four vectors. This follows from the input state values that never occur during simulation. The input combinations of 10 at $G4$, 01 at $G5$, 10 at $G2$ and 01 at $G3$ are the missing values that are necessary for detecting these faults. Note $mp1(G1)[1]$, $sp1(G1)[1]$ and $sp1(G1)[2]$ were never satisfied during simulation of these vectors. The detection status for each fault matches that of any exact fault simulator, which can be easily verified by the reader.

VI. RESULTS

For experimental verification, a randomly chosen set of 100 vectors was fault simulated for each of the ISCAS combinational benchmarks. These results from fault simulation were compared to upper bounds obtained from algorithms outlined in this paper. All monitoring data were collected during simulation of these 100 vectors, and the post-processing algorithm of Section IV was performed after the simulation was complete.

Results are shown in Table III. Exact fault coverages (%) are shown in column 2 of that table. To show the benefits of the work presented in this paper, fault coverage estimates were first obtained for each of these circuits. These estimates were obtained using an algorithm similar, but not identical, to that of Stafan [8]. For each individual fault, this estimator computes a detection probability which is compared to a threshold. The fault is assumed to be detected if this probability is greater than 50%. Further details of this estimation method are beyond the scope of this paper and are not presented in this paper. These estimated values are shown in the third column (marked "vanilla est"). The fourth column (marked "upper bound") shows the upper bounds obtained using the work outlined in this paper. Column 5 was obtained by executing the estimation algorithm (of column 3) only on faults that were not guaranteed to remain undetected, ie. faults obtained in column 4. It is evident that fault coverage estimates in column 5 are closer to actual fault coverages than those in column 3.

To measure the overhead of this algorithm, logic simulation without any monitoring was first performed. These execution times (in seconds) are shown in column 6 of Table III. The runtimes when signal conditions were monitored are shown in column 7, while column 8 shows the percentage overhead. These figures ignore the time required for both the pre-processing and the post-processing steps because they are fixed overheads independent of the length of the input sequence.

Table IV presents data that analyzes the previous results of Table III in greater detail, and shows the improvement in quality of estimates obtained by using our algorithms. For the random vectors for each ISCAS benchmark, Table IV presents the exact coverage (column 2), an initial estimate using statistical methods (column 3), and a refined statistical estimate using upper bounding (column 6). Any fault that is incorrectly estimated to be detected is referred to an *overshoot* and errors in this category are reported in columns 4 and 7. Any fault that is incorrectly estimated to remain undetected is referred to an *undershoot* and errors in this category are reported in columns 5 and 8. It is evident that both types of errors decrease when these algorithms are used. Overshoot errors decrease because dominator analysis is guaranteed to find additional faults that cannot be detected. The decrease in undershoot errors represent cases like the $G1\ s@0$ fault in Figure 2(a), where the stem fault is detected while all branch faults remain undetected. Both Stafan [8] and critical path tracing [1] tend to underestimate fault coverages when fault effect propagation occurs simultaneously along multiple fanout branches, without any fault on a fanout branch being detected.

VII. Conclusions

An algorithm to find a strict upper bound for stuck fault coverage was presented in this paper, and was shown to reduce the error in a fault coverage estimator. While the details of the estimator were skipped in this paper, the reduction of overshoot errors is independent of the exact type of estimator used. The number of undershoot errors is also reduced by finding reconvergence paths with equal parity. All algorithms are linear in circuit size, and the analysis consists of pre and post-processing steps. The runtime overhead during logic simulation is about 30-40%.

References

[1] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," in *IEEE Design and Test of Computers*, February 1984, pp. 83–93.

[2] S. B. Akers, B. Krishnamurthy, S. Park, and A. Swaminathan, "Why is Less Information from Logic Simulation More Useful in Fault Simulation," in *Proc. International Test Conference*, 1990, pp. 786–800.

[3] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Design and Analysis of Computer Algorithms*, Addison Wesley Publishing Company, 1974.

[4] T. Kirkland and M. R. Mercer, "A Topological Search Algorithm for ATPG," in *Proc. 24th Design Automation Conf.*, June 1987, pp. 502–508.

[5] M. H. Schulz, E. Trischler, and T. M. Sarfert, "Socrates: A highly efficient automatic test pattern generation system," in *IEEE Transactions on Computer Aided Design*, Jan 1988, vol. 7, pp. 126–37.

[6] W. Kunz and D. K. Pradhan, "Accelerated dynamic learning for test pattern generation in combinational circuits," in *IEEE Transactions on Computer Aided Design*, May 1993, vol. 12, pp. 684–694.

[7] W. Kunz and D. K. Pradhan, "Recursive learning: A new implication technique for efficient solutions to cad problems - test, verification and optimization," in *IEEE Transactions on Computer Aided Design*, Sept 1994, vol. 13, pp. 1143–1158.

[8] S. K. Jain and V. D. Agrawal, "Statistical fault analysis," in *IEEE Design and Test of Computers*, Feb 1985, vol. 2, pp. 38–40.