Upper Bounding Fault Coverage by Structural Analysis and Signal Monitoring*

Vishwani D. Agrawal
Auburn Univerity, Dept. of ECE
Auburn, AL 36849

Soumitra Bose and Vijay Gangaram
Intel Corporation, Design Technology
Folsom, CA 95630

Abstract

A new algorithm for identifying stuck faults in combinational circuits that cannot be detected by a given input sequence is presented. Other than pre and post-processing steps, certain signal conditions are monitored during logic simulation. These signal conditions are specified by an analysis of dominators and signal reconvergences in the circuit graph. After simulation, a post-processing step identifies 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% over that of a logic simulator. Experimental data show a substantial reduction of error in statistical estimates obtained by a stuck-fault coverage estimator when corrected for faults found by this algorithm as guaranteed to be undetected by the given sequence. An effective application of this technique is demonstrated for scan-based test point selection in an industrial scenario where circuit size and vector length prohibit the use of fault simulation.

1 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 in various areas of test, most predominant being fault coverage estimation of functional vectors as required for custom chips such as microprocessors. A strong motivation for this work comes from this critical need of the industry. For multi-million gate VLSI circuits, logic simulation is considered feasible but fault simulation is often impractical. Statistical fault simulators [5], though viable complexity-wise, can be too inaccurate in situations where vectors are written to specifically detect faults in certain modules. Because of the approximations used, a statistical fault simulator may estimate a non-zero detection probability for many undetectable faults, hence treating them as detected. The upper bounding technique of this paper can help revise such statistical coverage estimates.

At least two upper bounding algorithms have been proposed in the literature [1, 3]. Critical path tracing (CPT) [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 is individually detected (see example in Figure 1(a)). The method of Akers *et al.* [3] analyzes necessary conditions for fault detection. Both algorithms require fault analysis after simulation of each vector. We borrow ideas from both algorithms. While we monitor pre-specified conditions during simulation, unlike the two algorithms, the coverage is analyzed only after the simulation is completed. Thus, the per-vector computation overhead is reduced. Also, in contrast to CPT, we avoid tracing through gates, with multiple dominant input logic values, that are points of reconvergence for paths with different parity. However, at points of reconvergence with the same parity, the algorithm is exhaustive and repeats the analysis for each possible input choice that can justify the output.

Some basic definitions are presented in Section 2. An illustrative example runs through Sections 3 and 5. The details of the algorithm are presented in Section 4. Section 6 presents experimental results for ISCAS combinational benchmarks. We also provide a real-life application. For three industry designs, we show the use of the upper bounding algorithm to select scanout points to meet higher fault coverage requirements.

2 Definitions

A circuit is modeled as a directed graph where nodes of the graph represent gates, and edges correspond to signals. We will use the following definitions.

Checkpoints of a circuit consist of primary inputs and fanout signals [4].

Each gate is classified into one or more of three categories:

Fanout gates – Gates that drive fanout stems.

Reconvergent gates – Gates that are points of reconvergences.

Non-reconvergent gates – Gates that are not a point of reconvergence.

Note that recovergent and non-reconvergent categories are exclusive but each can have fanout gates.

Dominator – In a classical sense, a node A of a directed graph is a dominator of node B if every path from B to any output passes through A [2]. The set of

*This work was supported in part by a grant from Intel Corp.

nodes that dominates a given node is called its dominator set. Our definition of dominators differs from this classical graph theory definition. First, in the classical sense every node is a dominator of itself, but we exclude such cases. In addition, *we require that a dominator 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. Besides, a single-input gate will not be treated as a dominator.

Trivial dominator set – A given node is said to have a trivial dominator set if its immediate fanout is the only dominator.

Non-trivial dominator set – For a given node, a *non-trivial dominator set* necessarily includes at least one dominator node that is not an immediate fanout point of the given node.

Output sensitizing condition – Given a gate with a specified output value and its dominator set, this condition requires that all off-path sensitizing values for the dominators have appropriate non-controlling values.

Input sensitizing condition – Given a gate with a specified value on a specific input pin and its dominator set, this condition requires non-controlling values on all other inputs to the given gate and off-path inputs to the dominators.

3 An Example

We consider a small circuit (c17) and a sequence of two vectors to illustrate how the algorithm determines an upper bound on fault coverage. 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 5 after the algorithm is detailed in Section 4.

3.1 Structural Analysis

Structural graph analysis for finding dominators [2, 6] and reconvergence points [7, 8, 9] has been used in test generation systems. We use a similar analysis to monitor signal conditions at the end of each cycle in a vector sequence. These conditions are derived for single output circuits. For multi-output circuits, the conditions are obtained separately from each cone of logic that drives a primary output. These signal conditions are *necessary* for fault detection. If some of these conditions never occur throughout a vector sequence then certain faults are guaranteed to remain undetected by that 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

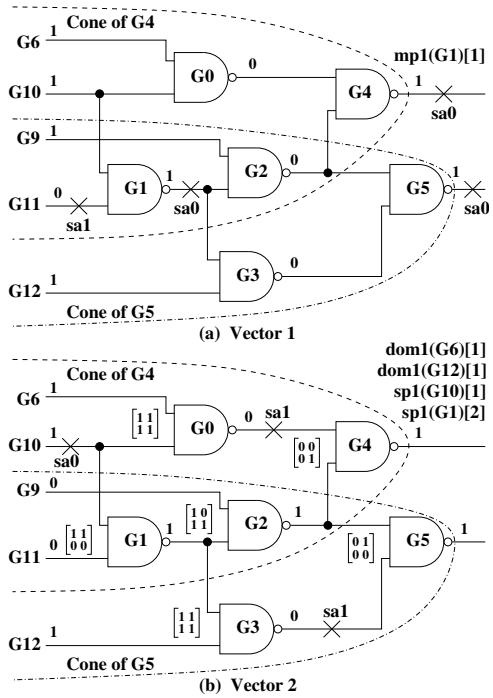

Figure 1: Simulation of c17 circuit for two vectors.

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 dominator set $\{G0, G4\}$. The other dominated gates are $G3$ and $G12$, with dominator sets $\{G5\}$ (trivial) and $\{G3, G5\}$ (non-trivial), respectively. Since this dominator analysis is performed for each cone, additional dominators are found for gates $G1$ and $G2$. For the cone of $G4$, $G2$ has a trivial dominator set $\{G4\}$, while $G1$ has a non-trivial dominator set $\{G2, G4\}$. In the cone of $G5$, $G2$ has a trivial dominator set $\{G5\}$.

In a given cone, for gates with trivial dominator sets, input sensitizing conditions are monitored explicitly, unless an input is a checkpoint in that cone. For non-trivial dominators, only output sensitizing conditions are monitored. Output sensitizing conditions for trivial dominators are ignored because they are obtained automatically by monitoring input states for the dominator gate. For the circuit of Figure 1, all inputs of both gates $G0$ and $G3$ are checkpoints and are skipped. For $G6$, the output sensitizing condition is the simultaneous occurrence of $\{$G6=v, G10=1, G2=1$\}$ for $v \in \{0, 1\}$. For $G12$, the sensitizing condition consists of $\{$G12=v, G1=1, G2=1$\}$ for $v \in \{0, 1\}$. In the cone of $G4$, 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 considered because $G1$ is not a checkpoint in this cone. The non-trivial dominator condition for $G9$ yields $\{$G9=v, G1=1, G0=1$\}$. For the cone of $G5$, gates $G2$ and $G9$ have domina-

Table 1: Sensitizing conditions for dominators in c17.

$dom0(G6)[1]$	=	{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}

tor sets $\{G5\}$ and $\{G2, G5\}$. Both inputs of $G2$ 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 $v \in \{0, 1\}$. Table 1 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 ([ ]).

A dominator set {G1, G2, G4} exists for gate $G11$ in the cone of $G4$. However, no dominator exists for this gate in the cone of $G5$. This is because $G5$, not having an input independent of $G11$, does not qualify as a dominator of $G11$ according to out definition. If no conditions are obtained for a gate from a cone, then all conditions for that gate obtained from other cones are also dropped from consideration. A fault that appears in multiple cones is considered undetectable if the relevant conditions in *all* cones remain unsatisfied during simulation. Gates $G10$ and $G1$ 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 contains them. These fanout gates in a cone are origins of reconvergent paths. Some fanout gates may have no reconvergent fanout in any cone and are not considered in this step, e.g., $G2$. Different reconvergent paths may have different (odd and even) inversion parity and fault propagation requires that paths with different parity be not simultaneously sensitized. However, along paths that have the same parity, simultaneous fault effect propagation may occur.

For the cone of $G4$, $G10$ has reconvergent paths with different 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). These are shown in Table 2. The notation "*sp*" is used for single paths, while "*mp*" will denote multiple paths. For $sp0(G10)[1]$, $\{G9 = 0 \| G11 = 0\}$ represents the disabling condition for propagation paths with different parity. For the cone of $G5$, there are reconvergent paths with the same parity from $G1$. Considering that it is possible to activate any subset of these paths, including all of them simultaneously, the conditions derived for $G1$ are as shown in Table 3.

Table 2: Different parity sensitizing conditions for $G10$.

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

Table 3: Same parity sensitizing conditions 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 $mp1(G)$) is ever satisfied during simulation, the $sa1$ ($sa0$) fault at the output of G will remain undetected. In addition, all faults whose propagation requires gate G to have a value 0 (1) also remain undetected.

3.2 Monitoring in Logic Simulator

Figure 1 shows a two 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 is complete, the set of input value combinations for each gate and the conditions of Section 3.1 that are satisfied are analyzed and faults that are guaranteed not to be detected are identified. The algorithm for the analysis of these input value combinations and the satisfied dominator conditions is presented in Section 4, following which we revisit this example in Section 5.

4 Algorithm for Post-processing

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

For a given node n, we evaluate two predicates, $oneProp(n)$ and $zeroProp(n)$, that denote possibilities for 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.

The immediate input value combinations of each gate are also stored. We denote by $\rho(n = 1)$ ($\rho(n = 0)$) the condition that node (signal) n attains the value 1 (0) during simulation. For a gate with input 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 are $s_i = v_i, 1 \leq i \leq N$, simultaneously. 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 study an AND gate. The formulas are similar for other gate types.

4.1 Propagation at Non-Fanout Inputs

For input s_i of a gate with output s_o, 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,

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

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

4.2 Propagation at Stem Inputs

Fanout stems can vary depending on the inversion parity of reconvergent paths. If reconvergent paths have different parity, a simultaneous fault propagation will cancel out the fault effects at the point of reconvergence. For paths that have the same parity, simultaneous fault effect propagation along multiple paths is possible and needs to be modeled.

4.2.1 Same Parity Reconvergence

As in Section 4.1, for each branch b_i, predicates $localOneProp(b_i)$ and $localZeroProp(b_i)$ are evaluated. At 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 has N branches. First, assume that 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 a parity different from that of s,

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

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

4.2.2 Different Parity Reconvergence

For each branch b_i, predicates $localOneProp(b_i)$ and $localZeroProp(b_i)$ are evaluated as in Section 4.1. At stem s,

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

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

where stem s has N branches.

4.3 Deducing Undetectability

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

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

Similarly, for a stuck-at-1 fault f,

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

5 Example: Post-processing

Returning to the example of Figure 1, we consider how the algorithm of Section 4 computes an upper bound on fault coverage. We assume that faults have been collapsed and only representative faults in equivalent classes are being considered. 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 to 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 formulas of Section 4.2.1, $oneProp(G1)$ ($zeroProp(G1)$) 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 formulas of Section 4.3, only the faults shown in Figure 1(a) are found to be detected.

For the second vector, the values of the propagation predicates are shown in Table 4. Two cones are considered separately. Note that $G2$ has no fanout in

Table 4: Propagation predicate values for 2nd vector.

Signal	Cone of G4		Cone of G5	
	zeroProp	oneProp	zeroProp	oneProp
G4	0	1	-	-
G5	-	-	0	1
G0	1	0	-	-
G2	0	0	0	0
G3	-	-	1	0
G1-G2	-	-	0	0
G1-G3	-	-	0	1
G1	0	0	1	1
G11	0	0	1	0
G9	0	0	0	0
G10-G0	0	1	-	-
G10-G1	0	0	-	-
G10	0	1	0	0
G6	0	1	-	-
G12	-	-	0	1

either of the cones. Considering vector 2 and the cone of $G4$ (columns 2 and 3), 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 a similar reason, both predicates for $G2$ are also 0. Tracing back from $G2$, all predicates for the 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* also evaluates to 1 for signal *G1-G3* and stem $G1$. The $sa0$ fault on stem $G1$ is correctly identified as detected by these two vectors by this analysis. However, as explained in the previous paragraph, this fault was also correctly identified as detected after the first vector. Predicate *oneProp* evaluates to 1 for $G12$ also. The $sa0$ at $G12$ is identified as detected by this analysis. Notice that this fault is equivalent to $sa1$ on $G3$, and is not shown in Figure 1(b). The three faults detected by the second vector are shown in Figure 1(b).

6 Results and Application

A set of 100 random vectors was simulated for each of the ISCAS combinational benchmarks. The results from fault simulation were compared to upper bounds obtained from algorithms outlined in this paper. For each circuit, monitored data were collected during simulation of the 100 vectors, and the post-processing algorithm of Section 4 was used only once after the simulation was complete.

Results are shown in Table 5. Exact coverages (%) from fault simulation are shown in column 2. A cumulative detection probability for 100 vectors was obtained for each fault using an algorithm similar to Stafan [5]. A 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 here. The estimated values are shown in the third column (marked "vanilla estimate"). The fourth column (marked "upper bound") shows the upper bounds obtained using the method outlined in this paper. Faults incorrectly identified as detected by the Stafan-like [5] analysis (column 3), but guaranteed to remain undetected by algorithms of this paper, were removed leading to the improved estimates of column 5, which are much closer to column 2.

Columns 6-9 show improvements in the errors of classifying individual faults as detected or undetected by the statistical method [5]. When the estimated detection probability of a fault is higher than 0.5, it is classified as detected, otherwise it is classified as undetected. The percentage of faults that is incorrectly classified as detected (compared to exact fault simulation) is referred to as *overshoot* error. Similarly, the percentage incorrectly classified as undetected is referred to as *undershoot* error. These errors for the original Stafan-like method (vanilla of column 3) and for the statistical method with upper bound improvement (column 5) are shown in columns 6 through 9. Upper bounding significantly reduces both types of errors. Overshoot errors decrease because dominator analysis finds faults that are guaranteed not to be detected. The decrease in undershoot errors is due to cases like $G1$ $sa0$ fault in Figure 1(a), where the stem fault is detected while all branch faults remain undetected. Both Stafan [5] and critical path tracing [1] tend to underestimate fault coverages when fault effect propagation occurs simultaneously along multiple fanout branches, without any fault on the fanout branch being detected.

Algorithm Complexity: The computing cost of the upper-bounding algorithm was determined as the overhead in logic simulation. *Plain* logic simulation (without monitoring) had execution times on Intel P4 CPU shown in column 10 of Table 5. When signal conditions were monitored runtimes of column 11, or percentage overheads of column 12 were seen. These numbers ignore the time of the pre-processing and the post-processing steps that are fixed overheads, independent of the length of the input sequence. For an n node (gate) circuit, the main contributor to pre-processing is the dominator analysis with $nlog(n)$ complexity [2]. Post-processing is linear and its time was negligible.

An Industry Application: The upper bounding technique was used for selection of scanout test points in large industrial designs. Table 6 shows data on three circuits. Column 2 gives the circuit size and column 3 gives the number of functional vector sequences. These sequences were of varying sizes, the larger ones having a million or more vectors. The maximum possible number of scanout points, permitted by critical timing consideration, is shown in column 4. Fault simulation with fault dropping determined the coverage in column 5. The CPU time in column 6 is for a multiprocessor. The goal was to maintain a coverage as close as possible

Table 5: Upper bounding algorithm results for benchmark circuits.

circuit name (1)	fault coverage (%) exact f. sim. (2)	vanilla estim. (3)	upper bound (4)	improv. estim. (5)	error (%) overshoot vanilla (6)	overshoot improv. (7)	undershoot vanilla (8)	undershoot improv. (9)	logic sim CPU s plain sim. (10)	upper bound. (11)	over-head % (12)
c432	93.15	96.95	94.47	93.32	4.96	1.34	1.15	1.15	0.03	0.04	33.3
c880	91.08	93.42	92.99	92.04	3.82	1.70	1.49	0.74	0.05	0.06	20.0
c1355	87.93	88.88	88.82	88.82	0.95	0.89	0.00	0.00	0.09	0.11	22.2
c1908	69.79	82.90	73.15	73.10	13.2	3.36	0.15	0.05	0.13	0.17	30.8
c2670	77.28	85.59	80.04	78.56	8.50	2.15	1.18	0.88	0.19	0.26	36.8
c3540	70.79	76.32	72.02	71.99	7.95	1.23	0.43	0.03	0.26	0.34	30.8
c5315	91.74	96.02	96.39	93.64	5.55	3.25	1.27	1.25	0.43	0.58	34.9
c6288	99.74	99.95	99.74	99.74	0.21	0.00	0.00	0.00	1.89	2.20	16.4
c7552	84.47	93.67	88.06	87.99	9.50	3.56	0.30	0.04	0.67	1.02	52.2

Table 6: Scanout testpoint selection by upper bounding fault coverage estimator.

circuit (1)	gates 10^3 (2)	vector seq. (3)	maximum scanout scan pts. (4)	% cov. (5)	CPU time (6)	designer's choice scan pts. (7)	% cov. (8)	upper bounding choice scan pts. (9)	% cov. (10)	CPU time (11)
d1	130	220	14540	77.55	8 days	2000	72.69	2000	74.13	45 mins
d2	180	45	8500	88.93	12 days	200	81.40	200	85.67	1.5 hours
d3	494	112	12550	78.48	15 days	1600	73.96	1600	76.77	4 hours

to that of column 5 with fewest scanout points. This would require fault simulation without fault dropping, which was considered too expensive. Columns 7 and 8 show the number of scanout points based on designer's knowledge and heuristics and the corresponding coverages by the fault simulator, which took about the same CPU time as in column 6.

As an alternative, upper bounding fault coverage estimates were obtained for all test point (scanout) candidates, and an optimal set of the same size as in column 7 was chosen such that estimated coverage was maximized. The runtimes of the estimator are shown in column 11. Fault simulation, again requiring about the same CPU time as in column 6, produced the coverages of column 10, which are indeed higher than the designer's choice coverages of column 8. The upper bounding coverage estimates were very close to those in column 10 and the estimator was one to two orders of magnitude faster than the fault simulator, which could only be run with fault dropping.

7 Conclusion

The upper bounding algorithm finds a strict upper bound for stuck fault coverage and reduces the error of a fault coverage estimator. Although the specific details of the statistical estimator are skipped, the reduction of overshoot errors is independent of the choice of the estimator used. The number of undershoot errors is also reduced by finding reconvergent paths with the same parity. Except for the pre-processing part, all algorithms are linear in circuit size. The runtime overhead added to logic simulation is about 30-40%. Application of this algorithm in a custom design environment has proven successful. Very large vector sets demand linear complexity fault analysis and the accuracy of the undetected fault data is necessary for improving the tests for increased coverage.

References

[1] M. Abramovici, P. R. Menon, and D. T. Miller, "Critical Path Tracing: An Alternative to Fault Simulation," *IEEE Design & Test of Computers*, vol. 1, no. 1, pp. 83–93, Feb. 1984.

[2] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *Design and Analysis of Computer Algorithms*. Reading, Massachusetts: Addison-Wesley, 1974.

[3] 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.

[4] M. L. Bushnell and V. D. Agrawal, *Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI Circuits*. Boston: Springer, 2000.

[5] S. K. Jain and V. D. Agrawal, "Statistical Fault Analysis," *IEEE Design & Test of Computers*, vol. 2, no. 1, pp. 38–40, Feb. 1985.

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

[7] W. Kunz and D. K. Pradhan, "Accelerated Dynamic Learning for Test Pattern Generation in Combinational Circuits," *IEEE Trans. CAD*, vol. 12, pp. 684–694, May 1993.

[8] W. Kunz and D. K. Pradhan, "Recursive Learning: A New Implication Technique for Efficient Solutions to CAD Problems - Test, Verification and Optimization," *IEEE Trans. CAD*, vol. 13, pp. 1143–1158, Sept. 1994.

[9] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A Highly Efficient Automatic Test Pattern Generation System ," *IEEE Trans. CAD*, vol. 7, pp. 126–37, Jan. 1988.