A Fast, Accurate, and Non-statistical Method for Fault Coverage Estimation

Michael S. Hsiao (mhsiao@ece.rutgers.edu)
Department of Electrical and Computer Engineering, Rutgers University, Piscataway, NJ

Abstract

We present a fast, dynamic fault coverage estimation technique for sequential circuits that achieves high degrees of accuracy by significantly reducing the number of injected faults and faulty-event evaluations. Specifically, we dynamically reduce injection of two types of faults: (1) hyperactive faults that never get detected, and (2) faults whose effects never propagate to a flip-flop or primary output. The cost of fault simulation is greatly reduced as injection of most of these two types of faults is prevented. Experiments show that our technique gives very accurate estimates with frequently greater speedups than the sampling techniques for most circuits. Most significantly, the proposed technique can be combined with the sampling approach to obtain speedups equivalent of small sample sizes and retain estimation accuracy of large fault samples.

I Introduction

Fault coverage estimation techniques attempt to quickly and accurately estimate the fault coverage from a given test set. As complexity and size of VLSI circuits steadily increase, fast and reliable fault coverage estimation will become more widely used, especially for relieving the high cost of full fault simulation in large circuits.

Traditionally, statistical methods have been used for fault coverage estimation; these include statistical fault analysis techniques [1, 2], fault-sampling methods [3, 4], as well as vector-sampling techniques [5]. Fault analysis techniques [1, 2] are based on detection probabilities of faults via fault-free simulation. Though fast, this technique is useful only to combinational circuits for ensuring reliable estimates, since most faults in sequential circuits frequently require propagation of several time frames before detection. Statistical vector-sampling [5] is based on a similar idea of detection probabilities, except that the total number of vectors simulated is reduced. Again, only combinational circuits will benefit from sampling vectors because sequential circuits place considerable constraints on the traversal of the state space to ensure detection of a fault. Statistical fault sampling [3, 4], unlike the previous two techniques, can be applied to both combinational and sequential circuits with good confidence of accuracy when the sample size is sufficiently large.

While the previous fault coverage estimation methods [1-5] attempted to reduce the execution time by statistically reducing the work involved, the approach presented in this paper is not based on statistical analysis or sampling. Instead, it avoids simulation of faults that are unlikely to be detected. Since fault simulators spend most of their time in evaluating sporadic events [6], it would be extremely beneficial if the sporadic events that never lead to detections could be avoided. We observed that much of the simulation time is wasted on evaluating faulty-circuit events that never propagate to a primary output (PO) or flip-flop (FF). This large amount of activity is due to faults that are never detected at all by the test set. Based on this observation, we propose a fast fault coverage estimation technique that achieves high degrees of accuracy by dynamically reducing injection of active but unlikely to be detected faults. Furthermore, unlike the sampling techniques where the fault coverage estimates may *either exceed or fall below* the actual fault coverage, our estimate will always be equal to or slightly less than the actual fault coverage, as we will at most miss detection of few faults. Finally, our approach can be combined with the sampling technique to obtain speedups equivalent of small sample sizes while retaining high estimation accuracy of large fault samples.

The *single stuck-at fault model* is considered in this paper since this fault model has been widely used in many of today's fault simulators. However, our technique can be applied to other fault models as well.

The remainder of the paper is organized as follows: Section II further explains the motivation and terminology used in this work. Section III describes the technique for avoiding injection of hyperactive faults. Section IV explains avoidance of injecting unactivated faults. Experimental results are discussed in Section V. Finally, Section VI concludes the paper.

II Motivation and Terminology

During fault simulation, each fault injected into the circuit causes a certain number of circuit elements to be evaluated, these evaluations are termed *faulty-circuit event evaluations*. Because the execution time during fault simulation is directly proportional to the number of faulty-circuit event evaluations, parallel-fault simulation [7] is frequently used to reduce execution costs by injecting a group of faults simultaneously. Execution time can be reduced further by avoiding injection of all the undetected faults. Traditionally, a fault does not need to be injected if we know a

priori that the given fault cannot be detected at a PO or propagated to a FF. The Star-Algorithm [8] presented several methods for identifying unexcitable and unpropagatable faults in combinational circuits. Other techniques of fault-injection removal were proposed in [9] in which 1 and 2-level fault-effect blocking was proposed for sequential circuits. In their approach, a fault is not injected if either it cannot be *excited* (i.e., the fault-free circuit value on the fault site is identical to the faulty stuck-at value), or if its fault effects cannot propagate beyond two levels of successor gates. We will call this scheme the *lookahead* scheme.

In this paper, an *inactive* fault is one that is not excited by the current input and state vectors. On the other hand, if a fault has its effects propagated to at least one FF in the previous time-frame, it is called an *activated fault*. Finally, an *active* fault is one that is either excited or activated in the current time frame. Thus, an activated fault is always an active fault, but an active fault may not be an activated fault.

Inactive faults can easily be prevented from injection by the lookahead technique, since they are not excited in the given time frame. On the contrary, only a small portion of active faults which are never detected can be prevented from injection by the lookahead technique, because lookahead techniques always injects faults whose fault-effects propagate beyond one or two levels of successive logic. Furthermore, hyperactive faults which never get detected will always be injected since they cause a lot of activity.

Significantly more reductions than the lookahead schemes in fault injection can be achieved if we are willing to allow for loss of a little accuracy on the fault coverage. Instead of having merely three categories of faults (inactive, blocked active, and active) as used in [9], we divide the faults in each time-frame into seven categories to account for hyperactivity. A fault is hyperactive if the number of faulty events exceeds a certain percentage of the total number of nodes in the circuit. The user may set the hyperactivity *threshold* at any level. The seven categories of faults for further fault injection reduction are as follows:

1. Inactive faults
2. Active faults that are detected quickly
3. Active faults that require long sequence for detection
4. Active faults that are never detected
5. Hyperactive faults that are detected quickly
6. Hyperactive faults requiring long sequence to detect
7. Hyperactive faults that are never detected

The cost of fault simulation can be greatly reduced if most of the hyperactive faults and faults whose effects never propagate to a FF or PO can be prevented from being injected. Unlike the static propagation technique in [8] and the lookahead in [9], we use *dynamic* techniques to reduce injection of both undetected hyperactive and unactivated faults for fast and accurate fault coverage estimation.

By avoiding faults that either require long subsequence for detection or are never detected, significant speedups in simulation can be obtained. However, slight inaccuracies in our fault coverage estimate may result from underdetections of a few active and/or hyperactive faults that require long sequences for detection. The amount of savings in faulty-event evaluations will be determined by how much we are willing to *tolerate* for hyperactivity during simulation. *Tolerance level* is defined as the number of consecutive time-frames an activated fault can be injected.

The notion of tolerance level requires us to define propagation and detection subsequences associated with each fault. A *propagation subsequence* T^f_{prop} for a particular fault f is a subsequence $T[v_i, v_{i+1}, ..., v_j]$ such that the fault-effects of f, starting from state at vector v_i, are propagated through all time-frames within the subsequence via FF's. A *detection subsequence* T^f_{det} for fault f is a propagation subsequence $T[v_i, v_{i+1}, ..., v_{j-1}, v_j]$ such that f is detected in time-frame j. Computing the *exact* propagation and detection subsequences requires backtracing of fault-effects through the circuit, and this may be computationally prohibitive for all faults. Instead, a continuous sequence of time-frames in which fault-effects are propagated is taken as the propagation subsequence.

III Avoiding Hyperactive Faults

Most hyperactive faults are never detected (but may be potentially detected). These faults cause exceedingly long propagation and/or detection subsequences and are responsible for generating significant amount of activity during fault simulation.

Table 1 shows the maximum and average lengths for detection and propagation subsequences using STRATEGATE [14] vectors on ISCAS89 [11] and several synthesized circuits. STRATEGATE [14] is a state-of-the-art genetic-based sequential test generator that achieves very high fault coverages in short execution times. In this table, the total number of faults, test set length, number of detected faults by the test set are first given for each circuit. Then, maximum and average detection subsequence lengths are reported for all (100%) of detected faults and 99% of detected faults that require shorter detection subsequences. Propagation subsequence lengths for the undetected faults are reported also in terms of maximum and average lengths. The ratio between the average propagation subsequence lengths to the average 99% detection subsequence lengths is shown in the right-most column.

As indicated in Table 1, the average detection subsequence lengths are generally very short compared to the test set sizes. If we neglect the 1% of faults that require longest detection subsequences, the maximum subsequence lengths drop significantly. For instance, the maximum length for circuit s5378 drops from 1629 to only 7! But the average subsequence length only dropped 0.48, indicating that the number of faults requiring long detection subsequences is very small.

Table 1: Maximum and Average Lengths for Propagation & Detection Subsequences

Ckt	Total Faults	Vec	Det	Detection Subsequences for Detected Faults				Propagation Subsequences for Undetected Faults		
				All Faults		99% Faults		All Undetected		
				Max	Avg	Max	Avg	Max	Avg	Ratio
s298	308	194	265	141	2.43	2	1.87	194	36.1	19.3
s344	342	86	329	8	1.18	2	1.14	86	39.5	34.6
s382	399	1486	364	163	16.6	89	15.7	1486	636.9	40.6
s400	426	2424	384	94	14.7	38	14.3	2424	865.7	60.5
s444	474	1945	424	93	12.2	12	11.8	1945	583.5	49.4
s526	555	2642	454	132	15.5	41	14.9	2642	313.9	21.1
s641	467	166	404	18	0.81	1	0.77	166	16.0	20.8
s713	581	176	476	21	1.12	3	1.04	176	10.1	9.7
s820	850	590	814	7	0.90	2	0.88	590	16.4	18.6
s832	870	701	818	8	0.83	1	0.81	701	13.5	16.7
s1196	1242	574	1239	23	0.44	1	0.41	48	23.0	56.1
s1238	1355	625	1282	94	0.49	1	0.40	9	0.32	0.80
s1423	1515	3943	1414	204	23.4	45	23.0	3943	2303.4	100.1
s1488	1486	593	1444	4	0.49	1	0.48	593	28.2	58.9
s1494	1506	540	1453	3	0.49	1	0.49	540	20.4	41.6
s5378	4603	11481	3639	1629	4.17	7	3.69	11481	407.3	110.4
s35932	39094	257	35100	51	3.82	27	3.82	257	0.64	0.17
am2910	2391	2509	2198	136	4.47	5	4.36	2509	178.4	40.9
mult16	1708	1696	1665	184	2.97	34	2.52	1696	868.3	344.6
div16	2147	1008	1815	275	6.73	7	6.59	1098	62.8	9.5
Average					4.96		4.75		280.5	**48.1**

Det: # of faults detected without fault sampling **Vec:** Test set length
Max: Maximum length of subsequence **Avg:** Average length of subsequences
Ratio: Ratio of average prop. seq. lengths to average dest. seq. lengths for 99% detected faults

Let us now turn to the propagation subsequence lengths for the undetected faults. Most of the undetected faults are activated throughout the test set. Some of the undetected faults are activated in every time-frame, as indicated by the maximum propagation subsequence lengths. There are two circuits in which propagation subsequences for the undetected faults are shorter: s1238 and s35932. In these two circuits, the majority of the undetected faults are *combinationally redundant faults*; these faults cannot be excited at all and can easily be identified by the lookahead technique. In particular, of the 3994 undetected faults in s35932, 3984 faults are untestable faults! Across all twenty circuits, the average propagation subsequence lengths are **48.1** times longer than the average detection subsequences for the 99% detected faults, and the long sequences often suggest hyperactivity. Similar results using test sets generated by other test generators such as HITEC [13] have also been obtained, although not shown here.

By avoiding injection of hyperactive faults that have been activated over a long time, we may miss detections of a few faults that require a long sequence for detection. Nevertheless, as shown in Table 1 and also supported by our experiments, the number of detected faults that require long detection subsequences is very small. Furthermore, we will *never* over-estimate the fault coverage as in the sampling techniques, since we will only miss detections of a few hyperactive faults.

IV Avoiding Unactivated Faults

Lookahead techniques fail to prevent injection of a fault if the fault effects are not blocked in one or two levels of successor logic. More levels into the circuit can be checked for fault-effect blocking; however, this may be counterproductive because the time and effort needed to check many levels may be more expensive than simply injecting the fault and simulate.

We propose an unpropagatability propagation algorithm in which previously learned nodes through which no fault-effects can propagate are dynamically propagated backward. Consider a fanout-free circuit fragment illustrated in Figure 1. The input value of 0 to the AND gate G essentially forces any fault-effects to be unable to propagate through the other input of gate G. If we had foreknown this information, faults associated with gates H and I would not have to be injected for this time frame.

There is one problem: if there are too many predecessors, marking of fault-effect-unpropagatable nodes can be costly, eventually defeating the benefits of unpropagatabil-

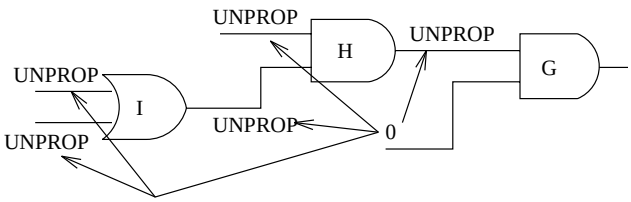
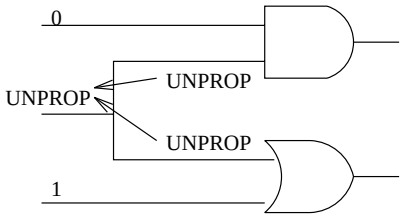


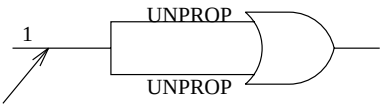
Figure 1: Backward Unpropagatability Propagation.

ity propagation. To avoid this, marking of unpropagatability is done *concurrently* with fault injection. For example, using the same circuit fragment of Figure 1, when we check whether the first input fault of gate G needs to be injected, we discover that its fault-effect is blocked by the controlling value (0) of the second input. Thus, we mark this node to be unpropagatable (**UNPROP**). Next, we also mark the inputs of its predecessor gate, H, to be unpropagatable as well, since the cost of marking the immediate predecessor is small. By the time we come to determine whether faults associated with gate H need to be injected, indication of **UNPROP** on this node tells us that no fault-effects on this node will be able to propagate to a PO or FF. Therefore, we refrain from injecting these faults and propagate the unpropagatability to its predecessor gate I. This process continues until all faults have been considered for each time frame.

For non-fanout-free regions of the circuit, propagation of unpropagatability should halt when a fanout stem is reached because the fault-effect may propagate via the alternate path. However, if we knew that the alternate path is also unpropagatable, we can mark the fanout stem to be unpropagatable, with the following exception: the fanouts must not lead to the same gate. If the alternate unpropagatable path leads to the same gate, the combined path may allow the fault-effect to propagate through. Figure 2 illustrates two cases of marking fanout stems.



(a) Fanouts that lead to different unpropagatable gates



Fault effects here can still propagate across the OR gate

(b) Fanouts lead to the same gate

Figure 2: Marking Unpropagatability on Fanout Stems.

Since we are only marking nodes as we encounter them, we may miss marking some nodes in the circuit from which fault-effect could also be blocked. It is a trade-off between marking more nodes and saving time for marking fewer nodes. Finding the optimal point where the maximal benefit is gained is difficult and is very circuit dependent, because the circuit structure dictates how unpropagatability can be propagated. When the number of levels the unpropagatability information can be propagated is small, the results are similar to those of the lookahead methods. Thus, little gain can be achieved by this technique. However, if the number of levels of propagation is great, the number of faults injected may be reduced significantly.

For this technique to be effectively applied, unpropagatability information needs to be *discovered* from the nodes furthest from the primary inputs so that the faults closer may benefit from the unpropagatability information. This implies that the ordering of faults should be such that the faults closest to the PO's and/or FF's be ordered first.

V Experimental Results

The fault simulator was implemented in C++. A basic algorithm based on PROOFS [10] was used, and the new features of avoidance of hyperactive and undetectable unactivated faults were added on top of the basic fault simulator. To evaluate the effectiveness of the fault simulator, results for fault sampling and our technique are compared in terms of fault coverage estimates and execution times. The basic fault simulator without the added features is used to evaluate fault coverages using sampled faults.

The results based on fault sampling shown in Table 2 using STRATEGATE [14] vectors are first discussed. The original STRATEGATE test set sizes are given in parenthesis under circuit names. Then, the full fault simulation (without any fault sampling) results are given. Next, fault coverage estimates and execution times for various sampling sizes, ranging from 2% to 30%, are reported for each circuit. Averages of ten runs are reported for each circuit, with the standard deviations shown in parentheses.

Variation in the results were the greatest in the smallest sample sizes. Up to 10% difference in standard deviation were observed for the 2% fault sample size. Under the assumption of normal distribution, the difference from the average estimate can be as much as 20% with a confidence of 95%! A 20% marginal error in fault coverage estimate is unacceptable for many designs. To reduce variation, larger samples are needed, however, at a higher computation cost. In terms of execution times, there is always an overhead simulation cost due to the fault-free simulation; thus, with sample size of 2%, the speedup is less than 50. Speedup is defined as $Time_{full\_simulation}/Time_{estimate}$ and depends on the fault-free simulation cost, fault sample size, and the characteristics of the fault sample. Although the execution times for most circuits are short for all benchmark circuits with or without sampling, they can be significant in large production circuits, for which fault coverage estimation techniques can be very helpful.

Fault coverage estimation accuracy from our approach depends on how much we are willing to *tolerate* hyperactivity. Tolerance level of x% allows for injection of a fault consecutively for at most x% of original test sequence length. Beyond the tolerance level, the given fault is withdrawn from being injected for the remainder of the test set. Therefore, more accurate estimates can be obtained with greater tolerance, but at a higher cost of simulation. The results are shown in Table 3. Again, the results for full fault simulation are first given for each circuit. Next, the estimates and

Table 2: Fault-Sampling Based Fault Coverage Estimations

Ckt (Vec)	No Sampling		Sampling Size									
	100%		2%		5%		10%		20%		30%	
	Det	Time	Est	Time	Est	Time	Est	Time	Est	Time	Est	Time
s298	265	0.20	268.5	0.06	269.8	0.08	271.3	0.07	269.2	0.10	267.0	0.11
(196)			(38.2)	(0.01)	(29.4)	(0.04)	(22.9)	(0.02)	(16.1)	(0.03)	(10.4)	(0.03)
s344	329	0.15	327.6	0.05	328.7	0.05	327.5	0.06	329.3	0.06	328.7	0.07
(88)			(24.7)	(0.02)	(15.7)	(0.03)	(12.1)	(0.02)	(5.9)	(0.02)	(3.6)	(0.01)
s382	364	1.88	363.3	0.32	369.2	0.49	369.2	0.71	369.3	0.83	366.6	0.97
(1488)			(27.3)	(0.25)	(14.3)	0.26)	(8.0)	(0.14)	(8.6)	(0.10)	(6.2)	(0.14)
s400	384	2.93	376.7	0.46	379.7	0.86	381.8	0.91	383.2	1.20	382.2	1.39
(2426)			(50.0)	(0.42)	(13.1)	(0.34)	(15.4)	(0.34)	(14.9)	(0.21)	(11.0)	(0.19)
s444	424	2.98	396.2	0.58	407.9	0.77	419.2	0.80	420.5	1.12	422.1	1.38
(1947)			(39.9)	(0.39)	(33.0)	(0.36)	(18.9)	(0.33)	(15.3)	(0.31)	(11.1)	(0.20)
s526	454	3.58	464.0	0.64	457.7	1.10	443.6	1.45	447.4	1.92	447.0	2.27
(2644)			(48.9)	(0.28)	(45.1)	(0.54)	(32.1)	(0.35)	(14.9)	(0.29)	(12.2)	(0.33)
s641	404	0.28	395.9	0.11	396.3	0.13	395.4	0.13	391.8	0.15	399.1	0.18
(168)			(60.6)	(0.02)	(28.3)	(0.01)	(30.3)	(0.02)	(17.5)	(0.04)	(14.7)	(0.03)
s713	476	0.38	518.4	0.14	492.7	0.15	474.4	0.17	474.4	0.17	475.0	0.22
(178)			(44.9)	(0.02)	(22.9)	(0.03)	(17.5)	(0.04)	(15.8)	(0.04)	(16.4)	(0.07)
s820	814	1.01	825.8	0.31	813.5	0.38	816.9	0.45	816.7	0.50	815.3	0.54
(592)			(26.5)	(0.19)	(22.7)	(0.18)	(12.7)	(0.16)	(12.0)	(0.16)	(9.6)	(0.16)
s832	818	1.13	816.4	0.31	815.1	0.38	818.7	0.39	824.1	0.46	822.4	0.65
(703)			(32.7)	(0.03)	(31.5)	(0.15)	(15.5)	(0.13)	(11.5)	(0.19)	(10.0)	(0.27)
s1196	1239	0.90	1229.6	0.34	1233.5	0.36	1237.9	0.39	1240.0	0.45	1239.0	0.50
(576)			(26.2)	(0.07)	(11.1)	(0.05)	(5.3)	(0.03)	(2.6)	0.04)	(2.5)	(0.04)
s1238	1282	1.01	1277.9	0.39	1289.3	0.44	1285.2	0.44	1278.8	0.54	1277.2	0.62
(627)			(61.0)	(0.03)	(29.3)	(0.01)	(26.0)	(0.02)	(16.7)	(0.07)	(15.1)	(0.08)
s1423	1414	28.7	1420.1	3.74	1423.2	4.83	1421.2	6.77	1421.0	8.79	1420.1	11.0
(3945)			(73.0)	(2.18)	(45.4)	(1.88)	(28.4)	(0.88)	(20.3)	(1.19)	(13.9)	(1.24)
s1488	1444	1.89	1440.8	0.47	1444.7	0.66	1439.7	0.71	1440.0	0.93	1442.3	1.16
(595)			(53.4)	(0.09)	(35.0)	(0.30)	(21.0)	(0.22)	(10.6)	(0.36)	(8.8)	(0.37)
s1494	1453	2.00	1448.9	0.48	1452.9	0.60	1459.5	0.65	1455.5	0.85	1454.3	1.05
(542)			(44.3)	(0.08)	(27.2)	(0.25)	(10.0)	(0.21)	(14.1)	(0.33)	(10.0)	(0.31)
s5378	3639	158	3746.0	26.2	3687.4	36.4	3656.5	44.0	3657.2	57.9	3652.7	70.3
(11483)			(158.7)	(6.18)	(95.0)	14.2)	(72.5)	(12.6)	(30.4)	(14.5)	(27.9)	(18.7)
s35932	35100	47.5	34783.7	7.11	34914.9	9.75	35073.1	12.4	35080.6	15.8	35072.3	19.8
(259)			(334.5)	(1.22)	(268.9)	(3.55)	(180.3)	(3.39)	(93.0)	(3.44)	(92.5)	(4.33)
am2910	2198	11.5	2164.5	2.04	2214.6	2.31	2210.4	3.17	2200.5	4.13	2199.7	5.86
(2511)			(121.8)	(0.24)	(43.7)	(0.51)	(34.0)	(0.68)	(26.2)	(1.06)	(15.5)	(0.95)
mult16	1665	7.41	1656.8	1.63	1655.4	2.66	1664.4	3.35	1668.0	3.74	1667.0	4.26
(1698)			(63.4)	(1.50)	(31.6)	(1.27)	(15.7)	(0.65)	(10.3)	(0.48)	(8.3)	(0.41)
div16	1815	5.20	1846.0	0.93	1811.1	1.28	1810.5	1.76	1830.5	1.91	1819.8	2.52
(1100)			(128.7)	(0.09)	(88.7)	(0.65)	(43.6)	(0.87)	(33.7)	(0.91)	(23.5)	(0.92)

Vec: # of vectors in the test set **Det**: # of faults detected without fault sampling
Time: Execution time in seconds **Est**: Avg # of estimated detected faults (standard deviation in parentheses)

execution times are listed for tolerance levels ranging from 2% to 10%. The fault coverage estimates that are within *one* standard deviation of the 30% sample size are highlighted in bold, while those that are within one standard deviation of the 2% sample are italicized. Please note that our technique never over-estimates the fault coverage.

In our approach, significant reductions have been reported for most circuits without much loss in estimation accuracy. For instance, speedup of over 5.50 was achieved in circuit s1423 with 2% hyperactivity tolerance level; only 32 faults were missed. In fault sampling, all estimates were off by at least 32 faults from the actual fault coverage for sample sizes of 10% or less. In the sample size of 20%, speedup of only 3.26 was obtained. Similarly, for circuits s5378 and mult16, our estimation approach achieved speedups of 2.90 and 4.28, respectively, for these two circuits with very accurate estimates, while fault sampling techniques achieved speedups of only 2.27 and 1.74 if the same error margin was to be achieved. In circuit div16, with 2% tolerance, we have achieved a very accurate estimate (off the actual coverage by 16 faults); no sampling size could offer similar results. Great speedups with very accurate estimates were also obtained by our approach for the smaller circuits as well, such as s298, s344, s382, s400, s444, s526.

Most importantly, we can combine our technique with sampling to further reduce the execution times without additional loss of accuracy. Table 4 shows the results of using 10% tolerance on 30% and 10% sample sizes. The fault coverage estimates (standard deviations in parentheses) and speedups are reported for both the original fault sample (from Table 2) and the combined approach for each circuit. Greatest speedups are highlighted in the table.

In all of the circuits, the combination of sampling and tolerance gave accuracies (in terms of estimate average and

Table 3: Tolerance-Based Fault Coverage Estimations

Ckt	Full Simulation		Tolerance Level									
			2%		3%		4%		5%		10%	
	Det	Time	Est	Time	Est	Time	Est	Time	Est	Time	Est	Time
s298	265	0.20	*249*	0.06	*254*	0.07	**258**	0.07	**259**	0.09	**265**	0.09
s344	329	0.15	209	0.08	281	0.06	*310*	0.07	**329**	0.06	**329**	0.06
s382	364	1.88	**358**	0.48	**360**	0.52	**362**	0.49	**364**	0.53	**364**	0.53
s400	384	2.93	**380**	0.67	**381**	0.75	**383**	0.70	**383**	0.77	**384**	0.86
s444	424	2.98	**422**	0.53	**424**	0.61	**424**	0.56	**424**	0.65	**424**	0.65
s526	454	3.58	**443**	1.08	**445**	1.06	**446**	1.04	**446**	1.10	**451**	1.55
s641	404	0.28	*358*	0.16	*381*	0.16	*389*	0.16	**391**	0.18	**400**	0.16
s713	476	0.38	405	0.17	*443*	0.19	*449*	0.20	*457*	0.21	**463**	0.23
s820	814	1.01	681	0.55	730	0.51	753	0.60	768	0.61	*798*	0.60
s832	818	1.13	707	0.62	741	0.57	756	0.61	771	0.65	*804*	0.67
s1196	1239	0.90	1040	0.68	1106	0.68	1131	0.73	1153	0.70	*1216*	0.70
s1238	1282	1.01	1079	0.75	1148	0.75	1178	0.88	1197	0.91	*1249*	0.88
s1423	1414	28.7	*1382*	5.16	*1390*	5.58	**1402**	6.15	**1407**	6.95	**1414**	10.5
s1488	1444	1.89	1249	0.95	1317	1.03	1355	1.03	*1390*	0.99	**1435**	1.13
s1494	1453	2.00	1240	0.85	1320	0.98	1368	1.02	1385	1.02	*1438*	0.98
s5378	3639	158	*3590*	54.5	*3602*	55.8	*3607*	57.9	*3609*	57.9	**3621**	65.9
s35932	35100	47.5	27614	16.2	30858	19.5	31976	21.5	32904	23.7	**35058**	30.8
am2910	2198	11.5	*2137*	4.16	*2157*	4.39	*2174*	4.72	**2184**	4.68	**2197**	5.26
mult16	1665	7.41	*1648*	1.73	*1654*	1.83	*1654*	1.89	*1654*	2.03	**1663**	2.24
div16	1815	5.20	**1799**	1.77	**1804**	1.73	**1804**	1.80	**1805**	1.93	**1813**	2.04

Det: # of faults detected without fault sampling **Time**: Execution time in seconds **Est**: # of estimated detected faults
Estimates for which differences from the actual fault coverage are smaller than standard deviation of 30% sample size in fault-sampling are highlighted in **bold**, while those smaller than the standard deviation of 2% are *italicized*

standard deviations) similar to those obtained using the sampling technique alone. The additional speedups obtained from the combination makes our technique very attractive. Typically, speedups from sampling alone would never exceed the ideal speedup of 3.33 with 30% fault sample, However, by combining sampling with tolerance methods, we often achieved speedups greater than 3.33! For instance, in circuit s1423, a 30% fault sample gave us a speedup of only 2.61; after combining with our technique, we achieved a new speedup of **5.23** with no sacrifice in estimation accuracy! Similarly, the speedups went from 2.25 to 4.31 and from 1.74 to 5.22 for circuits s5378 and mult16, respectively. For small circuits such as s382, s400, s444, and s526, great speedups were obtained using the combination approach without loss of fault coverage estimate accuracies. Speedups have nearly tripled in the 30% sample size, and they have more than doubled in the 10% sample size.

In general, the combination of 30% fault sample with 10% tolerance gave us speedups similar to the sampling technique using 10% sample size alone while retaining small deviations of the 30% sample size. Likewise, the combination of 10% fault sample with 10% tolerance frequently gave us speedups of using 2% fault sample, and retain smaller standard deviations of the 10% sample size. In short, the combination approach achieves speedups similar to that of a smaller sample size but still retain small variance in large samples. The benefit of adding tolerance to pure sampling is illustrated in Figure 3, where the speedup curve is raised substantially by the combined approach.

In circuits such as s820, s832, s1196, s1238, etc., because the pure sampling technique already underestimated the fault coverage, addition of tolerance-based technique further under-estimates the coverage. However, the differences in the estimates are generally less than 0.2%.

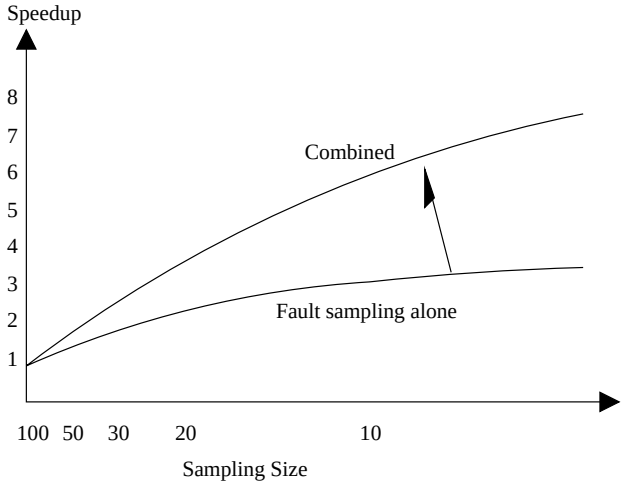


Figure 3: Benefit of Combining Both Techniques.

VI Conclusions

A dynamic technique for fault coverage estimation has been proposed in which very accurate fault coverage estimates have been obtained with significantly reduced execution time. Tolerance-based fault-injection technique is used to prevent injection of hyperactive faults, and unpropagatability information is propagated dynamically in the circuit

Table 4: Combining Sampling and Tolerance Methods

Ckt	Orig. 30% Sample			Combination			Orig. 10% Sample			Combination		
	Est		Spdup	Est		Spdup	Est		Spdup	Est		Spdup
s298	267.0	(10.4)	1.82	263.5	(9.3)	**2.70**	271.3	(22.9)	2.87	269.4	(25.2)	**3.28**
s344	328.7	(3.6)	2.14	327.2	(4.5)	**2.73**	327.5	(12.1)	2.50	325.3	(11.4)	**3.26**
s382	366.6	(6.2)	1.94	366.6	(6.2)	**5.22**	369.2	(8.0)	2.65	369.2	(8.0)	**6.27**
s400	382.2	(11.0)	2.11	383.2	(11.0)	**5.33**	381.8	(15.4)	3.22	381.8	(15.4)	**6.81**
s444	422.1	(11.1)	2.16	423.9	(11.1)	**6.77**	419.2	(18.9)	3.73	421.0	(18.9)	**7.64**
s526	447.0	(12.2)	1.58	443.9	(12.3)	**3.93**	443.6	(32.1)	2.47	439.4	(29.0)	**5.26**
s641	399.1	(14.7)	1.56	395.8	(15.2)	**2.00**	395.4	(30.3)	**2.15**	394.3	(30.2)	2.00
s713	475.0	(16.4)	1.73	461.6	(14.8)	**2.38**	474.4	(17.5)	2.24	463.4	(20.0)	**2.71**
s820	815.3	(9.6)	1.87	801.2	(11.5)	**2.53**	816.9	(12.7)	2.24	804.9	(16.8)	**3.16**
s832	822.4	(10.0)	1.74	810.4	(10.5)	**2.51**	818.7	(15.5)	2.90	807.3	(14.4)	**2.97**
s1196	1239.0	(2.5)	1.80	1213.4	(6.8)	**1.91**	1237.9	(5.3)	**2.31**	1218.0	(11.4)	2.20
s1238	1277.2	(15.1)	1.63	1247.2	(18.4)	**1.83**	1285.2	(26.0)	**2.30**	1257.1	(28.1)	**2.30**
s1423	1420.1	(13.9)	2.61	1420.1	(13.9)	**5.23**	1421.2	(28.4)	4.24	1421.2	(28.4)	**8.49**
s1488	1442.3	(8.8)	1.63	1434.5	(11.7)	**2.49**	1439.7	(21.0)	2.66	1427.9	(22.6)	**3.00**
s1494	1454.3	(10.0)	1.90	1436.5	(11.9)	**2.78**	1459.5	(10.0)	3.07	1427.7	(12.2)	**3.02**
s5378	3652.7	(27.9)	2.25	3632.7	(27.6)	**4.31**	3656.5	(72.5)	3.59	3632.7	(78.6)	**5.70**
s35932	35072.3	(92.5)	2.40	35027.8	(92.4)	**3.32**	35073.1	(180.3)	3.83	35034.0	(172.9)	**5.06**
am2910	2199.7	(15.5)	1.96	2199.3	(15.2)	**3.70**	2210.4	(34.0)	3.63	2210.4	(34.0)	**4.98**
mult16	1667.0	(8.3)	1.74	1664.3	(8.4)	**5.22**	1664.4	(15.7)	2.21	1660.3	(14.1)	**6.50**
div16	1819.8	(23.5)	2.06	1817.2	(23.2)	**4.16**	1810.5	(43.6)	2.95	1808.5	(44.3)	**4.48**

Spdup: Speedup of estimation over full fault simulation
Greatest speedups for each sample size are highlighted in **bold**

to avoid injection of excited but undetectable faults. As a result, great speedups were obtained by using the proposed approach; very accurate fault coverages were achieved compared with the sampling technique alone. Combination of our technique with the sampling method can further improve the speedup without sacrificing the estimation accuracy, which makes our technique very attractive. By this combination, we are able to achieve speedups of small sample sizes while retaining small variances in large samples. Speedups have nearly tripled for some circuits and more than doubled for most other circuits.

References

[1] S. K. Jain and V. D. Agrawal, "Statistical fault analysis," *IEEE Design & Test of Computers,* vol. 2, no. 1, pp. 38-44, Feb., 1985.

[2] S. K. Jain and D. M. Singer, "Characteristics of statistical fault analysis," *Proc. Int'l Conf. Computer Design,* pp. 24-30, 1986.

[3] V. D. Agrawal, "Sampling techniques for determining fault coverage in LSI circuits," *Journal of Digital Systems,* vol. 5, no. 3, pp. 189-202, Fall, 1981.

[4] R. L. Wadsack, "Design verification and testing of the WE32100 CPUs," *IEEE Design & Test of Computers,* vol. 1, no. 3, pp. 66-75, Aug. 1984.

[5] K. Heragu, V. D. Agrawal, and M. Bushnell, "Fault coverage estimation by test vector sampling," *IEEE Trans. CAD,* vol. 14, May, 1995.

[6] C. R. Graham, E. M. Rudnick, and J. H. Patel, "Dynamic fault grouping for PROOFS: a win for large sequential circuits," *Proc. Int. Conf. VLSI Design,* pp. 495-501, 1997.

[7] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design.* New York, NY: Computer Science Press, 1990.

[8] S. B. Akers, B. Krishnamurthy, S. Park, and A. Swaminathan, "Why is less information from logic simulation more useful in fault simulation?" *Proc. Int. Test Conf.*, pp. 786-800, 1990.

[9] E. M. Rudnick, T. M. Niermann, and J. H. Patel, "Methods for reducing events in sequential circuit fault simulation," *Proc. IEEE Intl. Conf. on CAD*, pp.546-549, 1991.

[10] T. Niermann, W. T. Cheng, and J. H. Patel, "PROOFS: A fast, memory-efficient sequential circuit fault simulator," *IEEE Trans. on CAD*, vol. 11, no. 2, pp. 198-207, Feb. 1992.

[11] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int. Symp. Circuits and Systems*, pp. 1929-1934, 1989.

[12] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Automatic test generation using genetically-engineered distinguishing sequences," *Proc. VLSI Test Symp.,* pp. 216-223, 1996.

[13] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," *Proc. European Conf. Design Automation (EDAC)*, pp. 214-218, 1991.

[14] M. S. Hsiao, E. M. Rudnick, and J. H. Patel, "Sequential circuit test generation using dynamic state traversal," *Proc. European Design and Test Conf.,* pp. 22-28, 1997.