PARALLEL SEQUENCE FAULT SIMULATION FOR SYNCHRONOUS SEQUENTIAL CIRCUITS

Chen-Pin Kung and Chen-Shang Lin

Department of Electrical Engineering
National Taiwan University

Abstract

A parallel sequence fault simulation (PSF) algorithm for synchronous sequential circuits is presented. The algorithm divides a given test sequence into subsequences of equal length and then performs multiple-pass fault simulation with these subsequences in parallel. The experimental results on the benchmark circuits show that the speedup ratio over a differential serial fault simulator is 4.90 on average for random test sequences. Conceptually, the PSF is an extension of the parallel pattern single fault propagation (PPSFP) and it reduces to PPSFP for combinational circuits.

1 Introduction

Fault simulation has been used extensively in grading the quality of test sets. In the VLSI era, the sizes of circuits and test sets are very large, hence an efficient fault simulator is needed. For the combinational circuits, the *parallel pattern single fault propagation* (PPSFP) [1] is one of the most efficient fault simulation algorithms. It takes the advantages of the independence of test patterns and word parallelism in computer operations as well as low memory requirement of single fault propagation. PPSFP simulates each fault with many patterns in parallel to reduce the fault simulation time significantly. Although PPSFP is shown to be extremely efficient for combinational circuits, it can not be applied to sequential circuits since the behavior of a sequential circuit at a specified time depends not only on the present input vector but also the previous state which, in turn, may depend on the entire preceding input sequence.

In this paper, we propose a parallel sequence fault simulation (PSF) algorithm which extends the concept of PPSFP to synchronous sequential circuits. The algorithm partitions a given test sequence into subsequences of equal length and then performs the fault simulation with these subsequences in parallel. To solve the problem of initial-state dependency of sequential

circuits, a multiple-pass method is developed and a detection mechanism is devised to use minimal simulation passes. The experimental results on the benchmark circuits show that the speedup ratio over a differential serial fault simulator is 4.90 on average for random sequences.

The remaining parts of the paper are organized as follows. The principle of parallel sequence logic simulation for synchronous sequential circuits will be introduced in the next section. Then based on similar principle, the parallel sequence fault simulation (PSF) and the implementation will be described. The experimental results of PSF on sequential benchmark circuits will be shown in Section four. Finally the conclusions will be given.

2 Parallel Sequence Logic Simulation

In this section, the principle of parallel sequence logic simulation will be described. Given a completely specified initial state S_0 and an input sequence T , a synchronous sequential circuit by logic simulation produces an output sequence and a final definite state S_f . Namely, the final values on FFs, or the final state (vector), can be completely determined as either 0 or 1. When the initial state is not specified, the logic simulator in general can only determine a final state (vector) with indeterminate values (X's) on some FFs. An unspecified state is *indeterminate* since all values of its state vector are X (indeterminate). However, it is easy to see that, with the same input sequence, the final state obtained from unspecified initial state covers that from specified one. A state (vector) S_1 is said to cover another state (vector) S_2 if each value in S_1 is either X or is equal to the corresponding value in S_2 . In general, we have the following observation.

Observation 1:

Let S_{1f} and S_{2f} be the final states of a sequential circuit M after the same input sequence I is applied with initial states S_1 and S_2 respectively. If S_1 covers S_2 then S_{1f} covers S_{2f} .

The implication of the observation is that with less information on the initial state, i.e. more X's, the logic simulation can still produce a final state containing the actual one. Furthermore, given a better specified initial state, the final state will have less uncertainties, X's. The observation is the basis of parallel sequence logic simulation.

Figure 1

A test sequence T can be viewed as a concatenation of a number of equal-length subsequences, say k subsequences, $T_0, T_1, T_2, \dots, T_{k-1}$. When the whole sequence is applied serially to simulate a circuit M , the states at the end of each subsequence are denoted by $S_i, i = 1, \dots, k-1$. This is illustrated in Figure 1 with $k = 4$. We call the simulation with input sequence applied serially as *serial sequence simulation*.

Figure 2

In the parallel sequence logic simulation, the status of circuit elements for these separate subsequences are arranged in a computer word, and the subsequences are simulated in parallel. Since, in a sequential circuit, the responses may depend on the whole preceding test sequence, the correct result in general can not be obtained by only one pass of parallel sequence simulation. However, by Observation 1, the parallel sequence result will contain the correct one, if the initial state for each subsequence is indeterminate. In order to obtain the correct result, multiple passes of simulation is needed. At the beginning of each simulation pass, the final state of each subsequence (excluding the last) in the previous pass becomes the initial state of next subsequence in the new pass as illustrated in Figure 2. With the transporting of final states, in the second simulation pass, the final states as well as the responses of first two subsequences

will be same as the correct responses. And for the rest of subsequences, their final states still cover the correct states possibly with less X's than those of the first pass. When the number of simulation passes is equal to the number of subsequences, the response of every subsequence will be as same as the responses of corresponding subsequence in serial sequence simulation. Obviously, in this extreme case, it is more efficient to use the serial sequence simulation. Fortunately, in most cases, the parallel sequence simulation needs only a few passes to produce the correct result and it outperforms the serial simulation as will be shown in our experiment. What we need now is a mechanism to detect when the correct result is obtained so that the simulation can be terminated. This mechanism is summarized in the following.

Definition :

During the parallel sequence simulation, the simulation is said to be stable if final state of each subsequence is identical to that of preceding simulation pass.

Observation 2:

When the parallel sequence simulation becomes stable, the responses of every subsequence in the final pass are the correct responses.

Base on the above observation, we can derive the correct simulation results with a few simulation passes by terminating simulation when it is found to be stable.

3 Parallel Sequence Fault Simulation (PSF)

In this section, we extend the methods of the parallel sequence logic simulation to fault simulation. The parallel sequence fault simulation is a single-fault-propagation simulator [6,7], i.e., the faulty circuit associated with a fault is parallel-sequence simulated at a time. When a certain primary output of the faulty circuit produces a determinate value, 0 or 1, which is different from the determinate value of the good circuit, the fault is detected. Note that a detectable fault can be found once the above condition is met even when the simulation is not yet stable. This is because that, based on Observation 1, a determinate value is a correct value for both faulty circuit as well as good circuit. In our implementation, a detected fault is dropped immediately without further simulation. The undetectable faults can be certain only when the whole sequence is simulated and both fault and good parallel sequence simulations are stable. This is summarized in the next observation.

Observation 3:

A fault is detected if in parallel sequence fault simulation, there exists a primary output which has determinate but different values in good simulation and fault simulation regardless of the stable condition. And a fault F can not be detected by the whole test sequence if F is not detected in parallel sequence fault simulation when both fault and good simulations are stable.

It can be seen that conceptually, PSF is an extension of PPSFP to the sequential circuits. In fact, for the combinational circuits, PSF needs only one pass and behaves like PPSFP.

Some implementation techniques which contribute to the efficiency of PSF will be addressed in the following. The gate evaluation is based on bit level parallelism and uses three values (0,1,X). For a 3-value simulation, 2 bits for a logic value are required. In PSF, '0' is coded as (0,1), '1' as (1,0), and X as (0,0). These two bits are kept in separate words.

In the parallel sequence fault simulation, it is not necessary to simulate the first subsequence repeatedly after one simulation pass has been complete, since the result for this subsequence is always correct. To make the fault simulation more efficient, at the beginning of subsequent simulation passes, PSF removes the first subsequence of the preceding simulation pass. This can be accomplished by simply shifting the vector one bit on each primary input. This shift operation also achieves the task of final state transporting.

In PSF, we adopt a technique proposed in differential fault simulation[2]. Namely, to reduce the number of events only the differences between current faulty circuit and the preceding circuit are simulated. However, for the purpose of identifying stable condition in PSF, the state differences between the current and the good circuit are stored as a list.

For the sake of saving memory requirement and enhancing the performance, state lists are not retained for previous simulation pass. Instead, two counters, Cd and Cx, for each fault are used to record the necessary information for all subsequences, where Cd is the total number of FFs with deterministic values which are different from the good circuit and Cx is the number of FFs with indeterminate values which are different from the good circuit.

Since it is not necessary to check if a faulty circuit is stable before the good circuit is stable, Cd and Cx are set to be negative values during this period. When the good circuit simulation becomes stable, for each faulty circuit, new Cd and Cx are computed and compared with one for the preceding pass. If these values remain the same then this fault simulation is stable. Since

the checking begins when good simulation is stable, if a faulty circuit simulation is not stable it must have some indeterminate values change to determinate values. Thus, either Cd or Cx or both will change. This guarantees to identify a stable fault simulation when it has been stable. Note that the counts for the first subsequence should be discarded when new Cx and Cd are stored, because in the next pass, the first subsequence in current pass is removed. A further improvement can be achieved by assigning Cd and Cx both to be zero when the state list is empty even though the good simulation is not yet stable. With this implementation, at the pass when good simulation just becomes stable, faults with both Cd and Cx remain zero are undetectable and can be dropped because fault simulation has been stable and produced no difference from the good one. Otherwise, faults have to wait till the fault simulation is stable to decide the detectability.

4 Results and Discussions

The PSF algorithm was implemented in C and runs on SUN4/390 workstation. To compare existent fault simulator for sequential circuits, a differential fault simulator (MDSIM) modified from differential fault simulation had also been implemented. Differential fault simulation has been reported to achieve 3-12 times faster than an existing concurrent fault simulation[4,5]. In MDFS, the FF's which have different values from those of good circuit are stored instead of those different from the previous faulty circuit as in differential fault simulator.

To evaluate the performance of parallel sequence fault simulator for a variety of circuits and test sequences, our experiment has been conducted on the sequential benchmark circuits[3] each with three different sets of test sequences. These three sets are randomly generated test sequences, sequences from a simulation-based ATPG, sequences from a propagation-then-justify ATPG. The comparison results of PSF and MDFS for these test sets are shown in Tables 1, 2, and 3, respectively. Also note that, in the experiment, each test sequence T is made possible to be partitioned into equal-length subsequences by inserting appropriate number of X's at the beginning of T . These inserted X's do not affect the results of simulation, since the circuits have indeterminate initial states.

In each table, the column N1 is the minimum number of simulation pass at which the good simulation becomes stable, and N2 is the average number of simulation passes needed to identify a undetected fault. Since PSF begins to drop the undetectable faults after the good simulation becomes stable and as indicated in the tables, most undetected faults are identified within the first two simulation passes after N1. Therefore, the number of

simulation passes needed to make the good simulation stable is the major factor for the performance of PSF. The circuit s5378 has slower performance of PSF than MDSIM in all these tables because it has particular high N1 due to its oscillatory subcircuits.

Table 1 shows the result for random sequence. For all but one (s5378) circuits PSF is faster than the MDSIM. The average speedup (excluding s510) of PSF over DSIM is 4.90. The particularly high speedup of s510 is because the circuit is not initializable, thus no fault can be detected. Note that, for s510, the parallel sequence logic simulation becomes stable at the end of pass 1 because the circuit is not initializable.

The test sequences used in Tables 2 and 3 are generated by a simulation-based ATPG and a propagate-then-justify ATPG, respectively. The average speedup ratios of PSF over MDFS are 2.28 and 2.63. Clearly, the performance of PSF for test sequences generated by ATPGs is not as good as for random sequences. One obvious reason is that the test sequences from ATPGs for comparable fault coverage tend to be shorter. As a result, only part of full parallelism, 32 subsequences, can be employed. However, the test length is only one the factors for the PSF performance. We found that PSF encounter degraded performance with circuits especially for circuits which are difficult to initialize. For example, s1494, there is a primary input used to reset the FFs in the circuit. The worst-case scenarios of PSF occur when this reset pin is enabled only at the first few vectors in the entire test sequence. Then, during PSF simulation, states for most of the subsequences remain indeterminate, consequently, the number of simulation passes to become stable is almost equal to the number of subsequences. This situation is worst with the simulation-based ATPG. The two basic phases of a simulation-based ATPG are initialization and forward propagation. The initialization is done only once and then faults are detected by appending more test vectors. Therefore, with such test sequences, PSF generally needs more simulation passes to become stable. This is why PSF results a better performance in Table 3 than in Table 2.

In summary, PSF delivers better performance than MDFS in most cases, although the speedup ratios vary with circuits and test sequence construction. To alleviate the worst-case scenarios, a simple strategy might be employed by performing the parallel sequence logic simulation before the fault simulation. If the number of simulation passes needed to make the good simulation stable is more than a certain ratio, say 1/2, of the number of subsequences then a serial fault simulator might be preferred. The results of parallel sequence logic simulation are also shown in Table 4. It can be seen that little overhead is needed to perform

the parallel sequence logic simulation.

5 Conclusion

A novel parallel sequence fault simulation algorithm (PSF) had been described. Conceptually, an extension of PPSFP to sequential circuits, PSF partitions a given test sequence into subsequences of equal length and then performs fault simulation for these subsequence in parallel. To overcome the initial-state dependency of sequential circuits, a multiple-pass method is developed and a detection mechanism is devised to use minimal simulation passes.

The performance of the parallel sequence fault simulator had been extensively evaluated on the sequential benchmark circuits with three different sets of test sequences. These three sets are randomly generated test sequences, sequences from a simulation-based ATPG, sequences from a propagation-then-justify ATPG. The results had been compared with a modified version of differential fault simulator. The comparison shows that although the speedup ratios vary with circuits and test sequence construction, PSF delivers better performance than MDFS in most cases.

References

- [1] J. A. Waicukauski, E. B. Eichelberger, D. O. Forlenza, E. Lindbloom and T. McCarthy. "Fault Simulation for Structured VLSI," *VLSI System Design*, December 1985, pp. 20-32.
- [2] Wu-Tung Cheng and Meng-Lin Yu. "Differential Fault Simulation - A Fast method Using Minimal Memory," *26th Design Automation Conference*, June 1989, pp. 424-428.
- [3] Franc Brglez, David Bryan, and Krzysztof Kozminski. "Combinational Profiles of Sequential Circuits," *International Symposium of Circuits and System*, May 1989, pp. 1929-1934.
- [4] E. G. Ulrich, and T. Baker. "The Concurrent Simulation of Nearly Identical Digital Networks," *10th Design Automation Workshop*, Vol. 6, June 1973, pp.145-150.
- [5] P. Goel, H. Lichaa, T. E. Rosser, T. J. Stroh and E. B. Eichelberger, "LSSD Fault Simulation Using Conjunctive Combinational and Sequential Methods," *International Test Conference*, 1980, pp.371-376.
- [6] J. P. Roth, W. G. Bouricius, and P. R. Schneider, "Programmed Algorithms to Compute Tests to Detect and Distinguish Between Failures in Logic Circuits," *IEEE Trans. Electron. Comput*, Vol. EC-16, No. 5, October 1967, pp. 567-580.
- [7] F. Ozguner, et al., "On Fault Simulation Techniques," *Journal of Design Automation and Fault Tolerant Computing*, Vol.3, No.2, 1979, pp. 83-92.

Table 1. Results for random test sequence

circuit	vectors	S_i	fault cov. (%)	CPU time (sec.)		N1	N2	speedup ratio
				MDSIM	PSF			
s208	640	20	34.88	5.17	0.78	2	2.21	6.63
s298	640	20	73.05	6.50	1.03	2	2.23	6.31
s344	640	20	94.73	5.50	0.90	2	2.39	6.11
s349	640	20	93.71	5.55	0.93	2	2.27	5.96
s382	640	20	12.28	13.53	3.47	2	2.49	3.54
s386	640	20	54.17	2.92	0.77	2	2	3.79
s400	640	20	12.03	13.53	3.20	2	2.48	4.23
s420	640	20	26.28	14.83	2.07	2	2.02	7.16
s444	640	20	11.18	16.08	3.75	2	2.49	4.29
s510	320	10	0.0	13.23	0.58	1	1.05	22.81
s526	640	20	8.65	19.82	4.60	2	2.34	4.31
s641	640	20	81.56	7.58	1.72	2	2.08	4.41
s713	640	20	76.94	8.52	1.88	2	2.06	4.53
s820	640	20	37.29	11.80	1.93	2	2.01	6.11
s832	640	20	33.22	12.62	2.23	2	2.01	5.66
s838	640	20	22.17	43.23	6.80	2	2.13	6.35
s953	640	20	8.34	154.15	42.42	2	2.64	3.63
s1196	640	20	83.01	10.05	1.77	2	2.16	5.68
s1238	640	20	77.27	11.97	2.05	2	2.15	5.84
s1423	640	20	36.17	69.38	19.77	2	2.64	3.51
s1488	640	20	55.52	22.63	4.13	2	2.01	5.48
s1494	640	20	57.37	23.08	4.10	2	2.01	5.59
s5378	640	20	61.74	160	195	29	29.11	0.82
s9234	320	10	0.26	593	124	4	2.20	4.75
s13207	640	20	5.96	1852	416	5	5.32	4.44
s15850	320	10	0.75	1072	234	3	2.32	4.58
s35932	640	20	68.65	1323	392	2	2.13	3.37
s38417	640	20	3.53	5709	1107	2	2.42	5.16

Table 4. Results for logic simulation on random sequence

circuit	vectors	S_i	fault cov. (%)	CPU time (sec.)		N1	N2	speedup ratio
				MDSIM	PSF			
s208	640	20	0.10	0.02	2			
s298	640	20	0.25	0.05	2			
s344	640	20	0.32	0.05	2			
s349	640	20	0.27	0.03	2			
s382	640	20	0.18	0.03	2			
s386	640	20	0.27	0.03	2			
s400	640	20	0.18	0.02	2			
s420	640	20	0.18	0.03	2			
s444	640	20	0.18	0.02	2			
s510	320	10	0.07	0.005	1			
s526	640	20	0.22	0.05	2			
s641	640	20	0.55	0.12	2			
s713	640	20	0.60	0.08	2			
s820	640	20	0.58	0.08	2			
s832	640	20	0.58	0.10	2			
s838	640	20	0.35	0.07	2			
s953	640	20	0.63	0.08	2			
s1196	640	20	1.08	0.15	2			
s1238	640	20	1.02	0.15	2			
s1423	640	20	1.12	0.20	2			
s1488	640	20	0.80	0.20	2			
s1494	640	20	0.92	0.20	2			
s5378	640	20	6.67	13.68	29			
s9234	320	10	1.23	0.57	4			
s13207	640	20	7.97	3.18	5			
s15850	320	10	7.15	1.73	3			
s35932	640	20	73.90	10.68	2			
s38417	640	20	45.32	5.70	2			

MDSIM : Modified Differential Fault Simulation

PSF : Parallel Sequence Fault simulation

N1 : minimum number of simulation passes when the good simulation becomes stable

N2 : average number of simulation passes needed to identify an undetectable fault

 S_i : subsequence length

Table 2. Results for test sequence generated by a simulation-based ATPG

circuit	vectors	S_i	fault cov. (%)	CPU time (sec.)		N1	N2	speedup ratio
				MDSIM	PSF			
s208	400	13	63.25	3.05	1.10	4	4.26	2.77
s298	350	11	85.71	3.93	1.50	5	5.20	2.62
s344	300	10	95.03	3.10	0.68	2	2.35	4.55
s349	99	10	95.43	1.12	0.73	2	2.37	1.53
s382	1014	32	88.97	22.42	11.44	10	10.36	1.96
s386	300	10	81.77	1.40	1.28	8	8	1.09
s400	1512	48	87.5	36.50	17.55	9	9.28	2.08
s420	200	10	40.70	5.27	2.15	3	3.16	2.45
s444	1692	54	85.86	47.17	23.47	10	11	2.01
s526	2682	86	79.28	95.92	38.78	8	8.25	2.47
s641	211	10	84.37	3.67	3.15	5	5.10	1.16
s713	350	11	81.76	5.22	2.50	4	4.06	2.08
s820	900	29	93.88	10.57	3.67	3	3.02	2.88
s832	703	22	91.61	11.85	5.10	5	5.03	2.32
s838	150	10	29.05	13.03	17.37	6	6.11	0.75
s1196	650	21	97.26	10.62	1.32	2	2.26	8.04
s1238	500	16	91.43	9.08	1.68	2	2.03	5.40
s1423	1501	48	82.97	440.43	409.67	9	9.59	1.08
s1488	300	10	94.01	15.60	17.90	29	29.04	0.87
s1494	850	27	96.02	29.13	39.58	31	31.06	0.74
s5378	501	16	69.15	98.73	155.10	26	26.11	0.64
s13207	149	10	5.81	422.37	211.25	5	5.35	2.0
s35932	313	10	88.27	605.58	588.43	4	4.03	1.03

Table 3 Results for test sequence generated by a forward-then-justify ATPG

circuit	vectors	S_i	fault cov. (%)	CPU time (sec.)		N1	N2	speedup ratio
				MDSIM	PSF			
s208	218	10	63.72	1.72	0.67	3	3.20	2.57
s298	354	12	83.12	2.32	0.73	4	4.29	3.17
s344	147	10	95.91	1.05	0.38	2	2.43	2.76
s349	189	10	94.00	2.98	0.65	2	2.29	4.58
s382	361	12	86.22	10.13	11.58	18	18.36	0.87
s386	244	10	81.77	0.92	0.80	3	3	1.15
s400	162	10	51.65	6.97	5.50	6	6.32	1.27
s420	128	10	41.63	3.45	2.55	4	4.13	1.35
s444	107	10	45.78	6.37	5.57	5	5.40	1.14
s526	296	10	82.97	16.03	14.12	7	7.41	1.14
s641	141	10	86.30	1.88	1.50	4	4.13	1.25
s713	600	19	81.41	9.68	2.83	3	3.06	3.42
s820	1005	32	95.76	14.93	3.63	3	3.03	4.11
s832	1026	33	94.02	15.40	3.67	3	3.02	4.20
s838	198	10	29.40	16.15	10.53	4	4.02	1.44
s1196	439	14	99.76	4.73	1.00	2	2.67	4.73
s1238	315	10	91.00	5.73	1.08	2	2.06	5.30
s1423	139	10	40.85	20.67	15.72	3	3.74	1.31
s1488	1233	39	97.04	41.50	10.17	4	4.04	4.08
s1494	1721	65	96.28	51.25	11.23	3	3.07	4.56
s5378	188	10	64.82	45.87	50.87	9	9.12	0.90
s13207	108	10	6.14	297.80	257.07	7	7.22	1.16