

PROOFS Fault Simulation Algorithm

Pratap S.Prasad
Dept. of Electrical and Computer Engineering
Auburn University, Auburn, AL
prasaps@auburn.edu

Term Paper for ELEC 7250 (Spring 2005)

Abstract

This paper discusses the prominent features of PROOFS, a super-fast sequential circuit simulator used in VLSI fault simulations. It touches upon the salient aspects such as the PROOFS algorithm, comparison with other contemporary simulators and experimental results. Also, PROOFS uses the advantageous features of differential, concurrent and parallel fault simulation techniques and uses the single fault propagation model. Moreover, it also reduces the software complexity, memory usage and the time required for simulation. It is demonstrated that PROOFS needs only 20% of the memory required for concurrent fault simulation but runs 6 to 67 times faster on certain ISCAS benchmark circuits.

Keywords : VLSI Testing, PROOFS, Fault Simulators.

1 Introduction

Gordon Moore predicted in 1965 that the number of transistors per square inch on integrated circuits would double every year since the integrated circuit was invented. With tremendous strides in VLSI technology, Moore's law is valid even today. Moore predicted that this trend would continue for the foreseeable future. However, in subsequent years, the pace did slow down a bit, but the data density has doubled approximately every 18 months, and this is the current definition of Moore's Law. The amount of logic circuitry and the complexity has also grown proportionally.

To test such a large number of components using only a few hundred pins on the chip is an extremely difficult task. This calls for design tools and other softwares and one amongst them is the *fault simulator*. A fault simulator [1, 2] is used to develop manufacturing

tests, i.e., it can simulate the circuit output response even in the presence of faults.

The fault list supplied(or generated, as the case may be) determines the coverage of a given set of input vectors for a given fault list. A good fault simulator is one that has a very high fault coverage for a very few input vectors. Also, because of the complexities of the circuits involved in present day VLSI chips, it is of paramount importance that the simulator also complete the job as quickly as possible. This can be done in various ways by inferring the data generated from the fault coverage. The simulation time can be reduced by not simulating faults that have already been detected while detecting another fault. Hence, fault simulators are used to find unproductive(vectors which can be eliminated without reducing fault coverage) test vectors; dropping of such vectors can hasten the test generation process.

To perform all these operations for complex VLSI circuits, we require fault simulators that are fast, and use memory efficiently.

1.1 Strategy for Fault Simulation

All fault simulators are based on a common philosophy. The essential tasks of a fault simulator are described in Table 1.

The table is a description of m faulty machines and n test vectors [3, 4]. Each column corresponds to one of the test vectors and each row corresponds to a fault-free or faulty machine. The primary task of the fault simulator is to generate the primary output values for each one of the $(m+1)n$ machine status and thereby determine which faulty machines have output vectors different from the good machine.

The usual strategy is to develop generate every machine status using a different test vector of the same machine. This is illustrated in Table 1. For instance,

Table 1: Tasks of a Fault Simulator

	V_1	...	V_i	V_{i+1}	...	V_n
Good	G_1	...	G_i	G_{i+1}	...	G_n
Bad ₁	$B_{1,1}$...	$B_{1,i}$	$B_{1,i+1}$...	$B_{1,n}$
Bad ₂	$B_{2,1}$...	$B_{2,i}$	$B_{2,i+1}$...	$B_{2,n}$
.
Bad _k	$B_{k,1}$...	$B_{k,i}$	$B_{k,i+1}$...	$B_{k,n}$
Bad _{k+1}	$B_{k+1,1}$...	$B_{k+1,i}$	$B_{k+1,i+1}$...	$B_{k+1,n}$
.
Bad _m	$B_{m,1}$...	$B_{m,i}$	$B_{m,i+1}$...	$B_{m,n}$

we can generate $B_{1,i+1}$ from $B_{1,i}$ by simulating the differences of their test vectors as the initial event sources and events inside their circuits.

Serial fault simulation, Parallel fault simulation [5, 6], deductive fault simulation [7] and concurrent fault simulation methods [8, 9] generate each machine status as described above. Concurrent fault simulator is the fastest but every line in the circuit is associated with a list, which requires a large amount of memory. Deductive fault simulation requires a special set of operations which have a large time overhead. Parallel simulation, simulates N (for a N bit machine) faulty machines per pass but does not have the ability to drop detected faults. Differential simulation [10] determines $B_{i,j}$ from the state $B_{i-1,j}$. This simulation requires less memory but suffers from the inability to drop detected faults easily because subsequent faulty machines rely on differences from previous faulty machines. Single fault propagation generates each machine state from its reference machine state in the same column, however restoring the state of the good machine before every faulty machine simulations results in a performance overhead. Single fault propagation with word level parallelism is parallel pattern single fault propagation technique. Efficient heuristics have been proposed to trace the fault effects in combinational circuits, which will not be useful for sequential circuits. All the drawbacks discussed above are overcome by PROOFS.

An alternative and more efficient approach is to generate each machine status from its reference machine status, i.e., using the same test vector but from a different machine. Some of the other fault simulators are *differential fault simulator* (DSIM), *SSIM* (a software levelized compiled-code simulator) and *ROOFS* (a forerunner of PROOFS).

DSIM can record value differences at state elements with minimal overhead. However, in sequential cir-

cuits, a fault can be detected across several time-frames and it is difficult to trace it. Hence, every faulty machine has to be simulated explicitly and the overhead to restore the fault-free machine status back might become difficult. Hence, what is done is that the reference machine in DSIM is not the fault-free machine but the previous machine simulated. However, SSIM uses the fault-free machine as the reference machine. Hence any particular kind of ordering of the fault has no effect in SSIM.

ROOFS stands for Restorative Order-independent Fault Simulator. This technique sets the fault-free machine status before performing a faulty machine simulation. The salient feature of ROOFS is the restoration of the fault-free machine status every time and this has to be efficient lest it penalize the simulation time. Hence the restoration procedure is discussed in some detail in the ensuing subsection.

1.2 Restoration of fault-free machine status

There are three copies of the circuit status at all instants. The first copy holds the fault-free machine status which is a result of true-value simulation. Further, every faulty machine status is assigned a simulation ID by which we can identify a particular fault in a machine. So, when a deviation from a fault-free machine status is detected, the faulty value is recorded in the second copy and the corresponding simulation ID in the third copy. There is a possibility to generate 2^N simulation IDs in a N -bit word processor. Hence, this does not cause any problems in the simulation time. Also, since the first copy contains the fault-free machine status, there is no time penalty in restoring it.

The restoration procedure is shown below and is self-explanatory.

```

For every test vector
{
Do true-value simulation;
for every undetected faulty machine
{
Give a unique simulation ID;
Inject current fault;
recover current states;
do event-driven simulation;
if the fault is detected, drop the fault;
}
}

```

2 PROOFS

PROOFS is an acronym for Parallel ROOFS. This is because of the parallel nature of execution of ROOFS. N-bit word length can be exploited by simulating N simultaneous logic gate operations. This is the technique of parallel simulation. A combination of parallel simulation and ROOFS is used in the PROOFS fault simulator.

2.1 Terminology

2.1.1 Fault Dropping

A certain fault is said to be dropped if that fault is detected and removed from current list of faults. Fault dropping reduces gate evaluations and speeds up simulation. If a fault is not detected, then all the state nodes with values different from the good machine values are stored in the linked list associated with the fault for the next vector.

2.1.2 Fault Grouping

The dynamic fault grouping strategy [11] regroups the remaining faulty machines into groups of N machines (because of the N-bit word length) for each vector that is applied to the circuit. In order to reduce the overall processing time, faults that cause the same events should be grouped into the same packet. In PROOFS, faults are ordered by a depth first search from primary outputs towards primary inputs during processing time.

2.1.3 Inactive Fault

If the stuck-at value of a single event fault f and the good value of the faulty line are identical, the fault f is not activated. Hence, it does not need to simulate the fault. These kinds of faults, called inactive faults, are not injected for parallel fault simulation. The concept of inactive faults has been further extended, in the later version of PROOFS. If a single event fault f is active, the fault f is attempted to propagate one (or two) level further. If the fault does not propagate to any gate in the next one (or two) level, the fault f is not injected.

2.1.4 Fault Injection

Faults are injected into the circuit by the fault simulator. In PROOFS, s-a-1 fault is injected by using an extra OR gate in the circuit and s-a-0 is injected by using an extra AND gate in the circuit. As shown

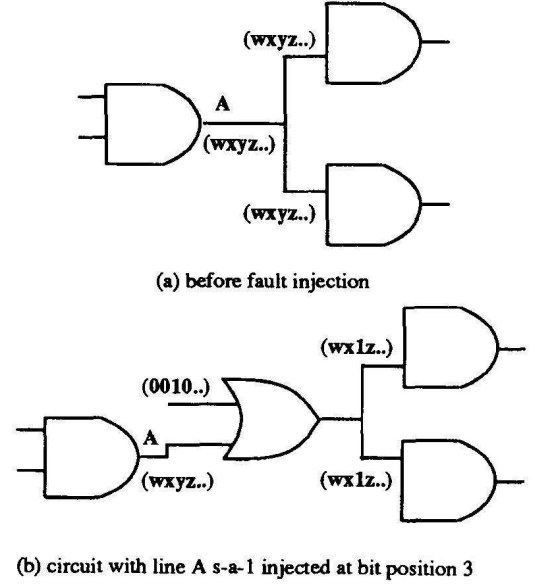


Figure 1: Fault Injection

	V0	V1
AND	$A_0 \& B_0$	$A_1 \& B_1$
OR	$A_0 \& B_0$	$A_1 \& B_1$
INV	A_1	A_0
XOR	$(A_0 \& B_0) \vee (A_1 \& B_1)$	$(A_0 \& B_1) \vee (A_1 \& B_0)$
TRIG	$E_0 \vee (A_0 \& E_1)$	$E_0 \vee (A_1 \& E_1)$
BUS	$(A_0 \& B_0 \& E_1) \vee (A_0 \& A_1 \& B_0)$	$(A_1 \& B_0 \& B_1) \vee (A_0 \& A_1 \& B_1)$

Figure 2: Gate Evaluations

in the Figure s-a-1 fault at A in 2^{nd} bit position is introduced by inserting OR gate at line A with other input of OR gate connected to a dummy gate whose input is all zeros except 2^{nd} bit position.

2.2 Data Format

Four valued $(0,1,X,Z)$ algebra is used in PROOFS. Z is used to depict bus conflicts in BUS nodes. Two bits are used to code the four values. 0 is coded as $(1,0)$, 1 as $(0,1)$, X as $(0,0)$, and Z as $(1,1)$. Figure 2 shows the logic used to evaluate different gates, 32 faulty machines at a time.

Fault-free and faulty machine values consists of two 32-bit words, V0 and V1, where each bit is used to store a different faulty machines value. Group id is stored in a 32 bit word. This allows more than 2^{32} faulty machine simulations without requiring to clear the older values.

2.3 The Algorithm

A synchronous sequential circuit can be decomposed into flip-flops and combinational logic blocks. The outputs of FFs are called pseudo-primary inputs (PPIs) and inputs of FFs pseudo-primary outputs (PPOs). Under the zero gate delay model, the sequential behavior of the circuit can be simulated by repeating the simulation of the combinational logic blocks in each time frame. After simulation of one group of faults, the PPOs which have different values as compared to good circuit are stored. The states of PPOs become PPIs in next time frame. The complete algorithm of PROOFS is shown in the Fig.3. The main loop which reads the next input vector, does the good machine evaluation. Group Id is incremented and a packet of N faults are injected together with previous vector state node events and simulated using bit parallelism of a computer word for that time frame. Faults detected at POs are dropped, and state node events are stored for next input vector. This is repeated until all the faults are simulated. Then the next vector is taken, group id is incremented and above steps are repeated. Good fault grouping strategy simulates all the faults in minimum number of passes thus reducing the iteration count in the inner loop and speeds up the simulation.

3 Results

A cross-section of the results of the PROOFS fault simulator are shown here. The PROOFS algorithm was implemented and run on many of the ISCAS sequential benchmark circuits. Figure 4 shows a summary of the circuits and test vectors used to evaluate the fault simulator. The vectors were generated using the sequential circuit test generator.

Figure 5 shows a comparison of PROOFS versus Concurrent. As we notice, the speed-up ratio varies greatly and can vary as less as 7 to as high as 67 for different benchmark circuits.

4 Conclusion

Parallel ROOFS was found to be very efficient in reducing events by 12% to 67% and time by 13% to 65%.

When compared with earlier fault simulators like differential fault simulator and concurrent fault simulator, PROOFS is definitely a better fault simulator in terms of speed and memory usage. Although new fault simulators based on parallel pattern single

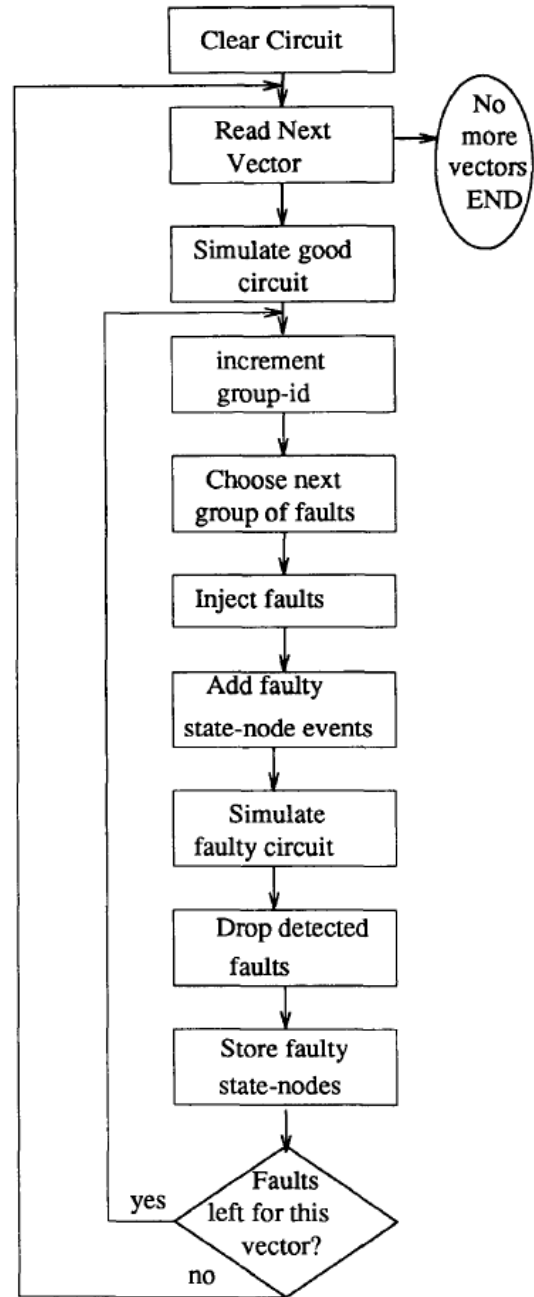


Figure 3: Flowchart depicting PROOFS algorithm

	PI	PO	FF	gates	faults	vectors	coverage
c432	36	7	0	160	524	75	99.23
c499	41	32	0	202	758	71	98.94
c880	60	26	0	383	942	106	100.00
c1355	41	32	0	546	1574	145	99.49
c1908	33	25	0	880	1879	186	99.52
c2670	233	140	0	1269	2747	214	95.74
c3540	50	22	0	1669	3428	199	96.00
c5315	178	123	0	2307	5350	308	98.90
c6288	32	32	0	2416	7744	35	99.24
c7552	207	108	0	3513	7550	398	98.25
s298	3	6	14	119	308	162	85.71
s349	9	11	15	161	350	91	95.71
s526	3	6	21	193	555	754	75.32
s713	35	23	19	393	581	107	80.90
s832	18	19	5	287	870	377	81.38
s838	35	2	32	390	857	137	29.64
s1238	14	14	18	508	1355	349	94.69
s1494	8	19	6	647	1506	469	91.10
s5378	35	49	179	2779	4603	408	74.02
s35932	35	320	1728	16065	39094	86	87.99

Figure 4: Descriptions of various Benchmark Circuits and vectors

Circuit	PROOFS		Concurrent		Speedup Ratio	Memory Reduction Ratio
	Run Time (sec)	Maximum Memory(Kb)	Run Time (sec)	Maximum Memory(Kb)		
s208	2.3	80	24.3	552	11	7
s298	4.1	96	32.5	608	8	6
s344	2.9	104	33.5	656	12	6
s349	2.9	112	33.7	656	12	6
s382	67.2	112	445.1	728	7	7
s386	3.0	104	32.2	664	11	6
s400	37.1	112	347.2	720	9	6
s420	9.8	120	69.3	632	7	5
s444	56.2	120	757.5	808	13	7
s526	40.2	120	349.5	896	9	8
s526n	35.5	120	235.2	992	7	8
s641	5.0	208	38.5	600	8	3
s713	5.5	216	54.5	752	10	3
s820	23.4	192	182.0	744	8	4
s832	23.2	176	175.7	816	8	5
s838	19.3	224	145.8	888	8	4
s953	3.3	176	32.0	720	10	4
s1196	11.9	216	111.9	720	9	3
s1238	16.7	216	133.2	728	8	3
s1423	9.1	344	254.1	1288	28	4
s1488	53.7	272	544.6	1392	11	5
s1494	44.2	272	516.8	1184	12	4
s5378	174.0	752	1367.8	1544	8	2
s35932	358.3	5872	24148.6	5576	67	0.95*

Figure 5: A comparison of PROOFS with Concurrent

fault propagation PROOFS is faster for large circuits. Fault simulator, HOPE, uses the idea of screening out of faults with short propagation paths through single fault propagation. Effective fault injection methods, better dynamic fault grouping strategies and taking advantages of PARIS at the expense of memory are some directions for the improvement of PROOFS fault simulator.

References

- [1] M.L.Bushnell and V.D.Agrawal, "Essentials of Electronic Testing for Digital, Memory and Mixed-Signal VLSI circuits" Kluwer Academic Publishers., 2000
- [2] M.Abramovici, M.A.Breuer, A.D.Friedman, "Digital Systems Testing and Testable Design", IEEE Press, 1995.
- [3] W.-T.Cheng and J.H.Patel, PROOFS: A super fast simulator for sequential circuits, Design Automation Conference, Mar. 1990.
- [4] T.M.Niermann, Wu-Tung Cheng and J.H. Patel "Proofs: a fast, memory efficient sequential circuit fault simulator", Proceedings of the 27th ACM/IEEE conference on Design automation, pp. 535 - 540, 1991
- [5] S.Seshu, "On an Improved Diagnosis Problem", IEEE Transactions on Electronic Computers., vol EC-14, no 1, pp 76-79, Feb 1965.
- [6] S.Seshu and D.N.Freeman, "The Diagnosis of asynchronous Sequential Switching Systems", IRE Transactions on Electronic Computers, vol EC-11, Aug 1962.
- [7] D.B.Armstrong, "A Deductive Method for Simulating Faults in Logic Circuits", IEEE Transactions on Computers, vol. C-21, no.5, pp. 464-471, May. 1972.
- [8] E.G.Ulrich, V.D.Agrawal and J.H.Arabian, "Concurrent and Comparative Discrete Event Simulation". Boston: Kluwer Academic Publishers, 1994.
- [9] E.G.Ulrich and T.Baker, "Concurrent Simulation of nearly Identical Digital Networks", Computer, vol 7, pp. 39-44, Apr 1974.
- [10] W.-T.Cheng and M.-L.Yu, "Differential Fault Simulation for Sequential Circuits", Journal of

Electronic Testing: Theory and Applications, vol. 1, no. 1, pp. 7-13, Feb. 1990.

- [11] C.R. Graham, E.M. Rudnick, J.H. Patel, "Dynamic Fault Grouping for PROOFS: A Win for Large Sequential Circuits", International Conference on VLSI Design: VLSI in Multimedia Applications, 1997