Efficient Concurrent Simulation of Large Networks Using Various Fault Models

Evan Weststrate and Karen Panetta
Tufts University
Department of Electrical Engineering and Computer Science
161 College Avenue
Medford, MA 02155
eweststr, karen@eecs.tufts.edu

Abstract

An improved fault simulation environment is described in this paper, and details of its faulting capability are presented and tested. The versatility of our fault simulator in handling different fault models by adding new activity functions to our modeling structure such as n-terminal bridge faults is shown. We use our TUFTsim simulator, which is based on concurrent simulation algorithms to efficiently fault simulate large networks, and the Multiple List Traversal [9] mechanism handles the propagation of concurrent elements through the topology. Results on some of the ITC '99 Benchmarks [1] are shown based on the stuck-at and logical bridge fault models. The use of the bridge fault in combination with the stuck-at model does not depreciate the simulation efficiency.

1. Introduction

Within a fault simulation environment, there is a need for efficient algorithms and supporting data structures to enable fast pattern grading for ever increasing circuit complexities. Our current simulation tool, TUFTsim, has the capability to handle complex fault models in addition to the traditional stuck-at model to verify modeled defect accuracy, improve simulation resource usage and provide fault coverage comparisons.

1.1. Motivation

There always exists a need for more efficient, multi-model diagnostic tools for fault simulation. Lavo et al [2] discusses the issues of using many differing fault models and algorithms. The question of how closely the models can represent actual defects, let alone if defects will be found using patterns that have been graded using these basic models, still remains.

Many simulators that have been developed [3,4,5] target single or specific fault models. DeLong, Johnson, and Profeta [6] also report on a method of fault injection for VHDL behavioral models, for fault tolerance evaluation early in the design cycle. The focus here will be production test validation. The versatility of our algorithms allow for simple insertion of any fault model at any level.

2. Overview

Concurrent algorithms are the basis of our simulation environment [7]. This method can efficiently handle timing models, asynchronous circuits, and functional models. The goal of concurrent simulation (CS) is to produce simulation results equivalent to simulating each faulty circuit separately. Scenarios of experiments are created for the reference experiment and for each fault. The main idea is to evaluate these scenarios and propagate only those experiments that create state differences from the reference output.

In CS, a primary experiment, called the reference experiment, exists during the entire simulation and exhibits the defect free or good behavior of a circuit. The behavior of these experiments can be observed and contrasted, since each experiment propagated leaves a signature, namely its experiment identifier. In CS, the simulation of identical behaviors, reference and faulty, is performed only once. Additional simulation time is dedicated to faulty experiments that differ from the reference experiment.

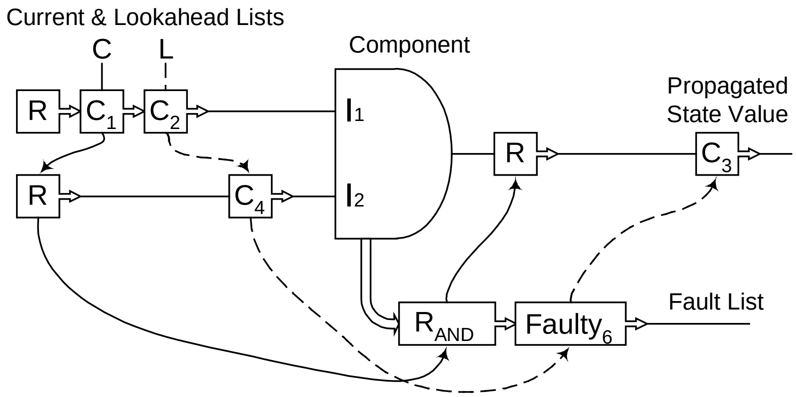

Figure 1. Conceptual diagram of the Multiple List Traversal. The current list assembles the scenario to be simulated, while the look-ahead list, denoted by the dashed lines, determines the next experiment to be processed after the current one is completed.

3. Function Lists / Multiple List Traversal (MLT)

Every element simulated maintains lists of concurrent identifiers at each terminal along with what is called a *function list* [8]. The function list allows the simulator to assume that components are black boxes, eliminating the need to know its behavior. The simulator only requires a handle or pointer to the evaluation routine of a component. The function list also stores fault sources, which are to be processed during the MLT along with all input and output lists. By not storing fault sources on the input and output lists, the simulator is able to reduce list traversal times and efficiently insert faults into the network topology. The initial item on the function list is the evaluation function for the fault free reference experiment. Other items on the function list specify the behavior for a fault source. The function list provides a level of abstraction that can enable the simulator to perform parameter (stuck-at) faulting, delay faulting, or functional faulting.

There are many methods to implement concurrent simulation, but the most efficient single CPU algorithm is the Multiple List Traversal (MLT) algorithm [9]. Each gate or component in a network topology is actually represented in the simulator by data structures. The component data structure stores information pertinent to an element's function and delay information, as well as links to its inputs and outputs. The inputs and outputs of a component are linked lists that store all experiments identifier information. By traversing all inputs and outputs simultaneously, the MLT determines the presence of a specific experiment identifier (reference or faulty) on any input or output. This provides a reliable snapshot of all state information, including whether a fault effect is present on multiple lists. This reduces the number of evaluation routine calls necessary during simulation. The output lists are also included in the MLT algorithm to quickly insert and propagate experiments or update states.

The MLT creates the simulation scenarios by using two positioning lists [8] called the current list and the look-ahead list (figure 1). A current list contains pointers to the state information to be used in evaluating an experiment scenario. If needed, the current experiment is evaluated, using the function on the current list, and is checked to determine if the concurrent state value needs to be explicitly represented on each output of the element. The look-ahead list contains pointers that keep track of which experiment should be processed next. By doing this, the MLT needs not continually traverse the entire list of state values on a given terminal during an evaluation cycle to find each incremental concurrent ID.

4. Implementing Bridge Fault Models

In order to demonstrate the flexibility in the architecture, different fault models were added to the simulator's functionality. A stuck-at fault requires replacing the input of a gate with faulty value. The component still obeys its characteristic behavior, calling the same function as the reference, but the fault source uses a stuck at value in place of the input.

The goal of implementing the bridge fault model was to demonstrate that *functions* or behaviors could also be inserted and run concurrently in our framework. To add a fault model requires that the functional behavior of the fault be included in the fault library with respect to the inputs and the outputs. For these examples, the fault functions were written in the C language. The OR bridging fault inserted at the input of AND gate 1 in figure 2 [10,11] will propagate to any fanout connected to it.

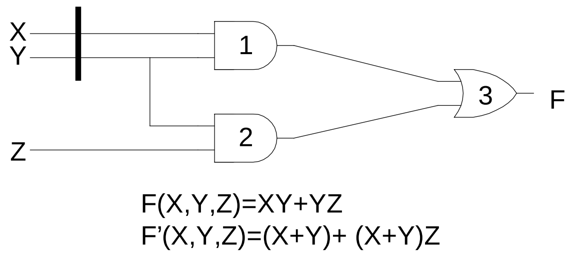

Figure 2. Circuit with an OR-bridge fault inserted between inputs X and Y. Function F is the fault free function and function F' is the function realized by the bridge fault.

In the simulator, no new gates are inserted to emulate the bridging fault. Instead, another fault source is included on the topological element's function list. The fault source and any effects propagated are handled seamlessly through the MLT processing.

The insertion of bridging faults requires that all elements affected by the bridge include a pointer to the faulty bridge behavior. In figure 2, the network contains a bridge fault on inputs X and Y. A bridge fault source is inserted onto the function list for AND gate 1. Since input Y also fans out to the AND gate 2, we insert what we will call a "*fault source alias*", on the function list of that element. The fault source alias maintains the fact that an input of gate 2 is dependent on a function to determine its input value rather than an assigned value, like a stuck-at fault. Therefore, the first input of gate 2 will see the (X+Y) opposed to the correct signal Y.

Figure 3 depicts the same network as figure 2 from a topological data structure point of view. The Function List is shown contained within each topological element in addition to the element's input and output lists.

5. Experimental Results

Simulations were run on the ITC '99 Benchmarks [1] from Politecnico di Torino. The circuits used were the synthesized netlists in bench (ISCAS89) [12] format. A reset signal with AND gates connected to the data inputs was inserted on all flip-flops instead of direct reset inputs for controllability. Test patterns were generated using the GATEST genetic algorithm based test pattern generator [13]. In order to exercise the simulator, an exhaustive set of stuck-at 0 and 1 faults on every possible element input were first run with the GATEST waveforms. Fault coverage results in all stuck-at experiments from our TUFTsim simulations closely matched those reported by GATEST.

A set of bridge faults was then simulated on the same benchmarks. Every OR and AND gate was given all possible wired-AND and wired-OR bridge faults, respectively, on the input terminals. The same test patterns were used.

Demonstrating the ability of the simulator to handle multiple fault types within one simulation, the simulations were run again, with the addition of the bridge faults. Every combination of n-terminal bridge faults was used, once again. For example, a four input AND gate would be assigned eleven wired-OR bridge faults and eight stuck-at faults on the inputs.

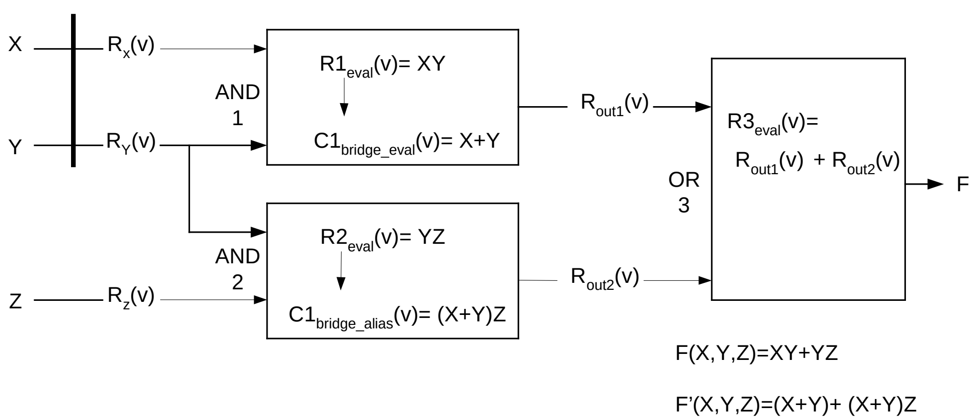

Figure 3. Topological view of a network showing bridge fault insertion between inputs X and Y of element 1 and the resulting alias fault source insertion. The function list on each element is shown containing both the reference evaluation function and the bridge fault evaluation function.

The simulations were run on a 450Mhz Sun Ultra 80 with 1Gb of memory. Table 1 represents the simulation results.

Since the test patterns are not being evaluated here, the actual coverages are not significant. What is important to note in figure 4 are the detection differences between fault sets. On all but one benchmark circuit

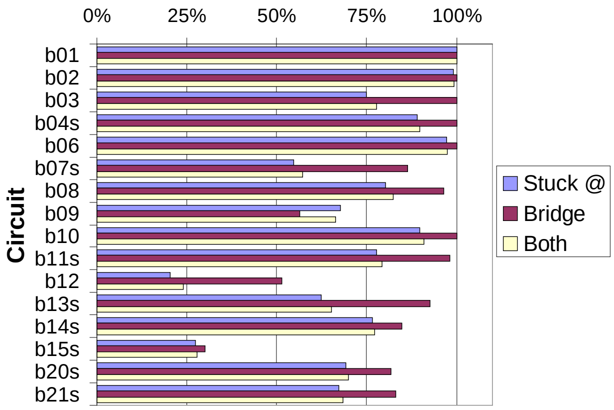

Figure 4. Fault coverages for stuck-at and bridge fault sets

tested, the bridge fault coverage was higher than that which the patterns were generated for – the stuck-at fault set. Comparable numbers of the two different fault types were easily run in the same simulation environment without costly CPU times.

Propagated events are simply the pin state values that have been propagated through the circuit representing either the reference case or a concurrent faulty state not represented by the reference state. Plotted against this in figure 5 is the number of component evaluations. This corresponds to the number of times that the MLT algorithm requires an evaluation or re-evaluation of a

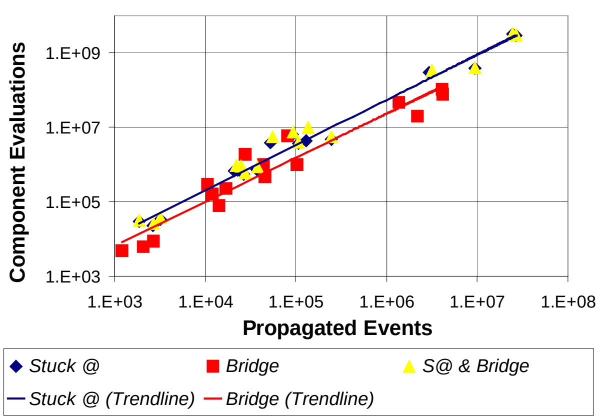

Figure 5. Component evaluations vs. propagated events for stuck-at and bridge fault sets

Table 1. ITC '99 Torino Benchmark fault simulation results using TUFTsim

Network Name	# of Gates /FF	# of Test Vectors	Fault Set	# of Inserted Faults (bridge fault aliases)	# of Events	# of Component Evaluations	Simulation CPU Time (s)
b01	52 /5	80	S-@	208	3,200	33,631	0.037
			Brdg	22 (148)	2,710	8,819	0.010
			Both	230 (148)	3,220	38,459	0.044
b02	33 /4	104	S-@	122	2,660	23,340	0.025
			Brdg	16 (118)	2,084	6,177	0.007
			Both	138 (118)	2,672	26,161	0.029
b03	180 /30	270	S-@	720	37,056	732,131	0.633
			Brdg	104 (1,479)	12,063	157,680	0.165
			Both	824 (1,479)	37,755	852,641	0.794
b04s	664 /66	442	S-@	2,596	244,060	4.71E+6	5.421
			Brdg	173 (4,819)	102,601	998,204	1.097
			Both	2,769 (4,819)	245,610	5.49E+6	6.775
b06	70 /9	66	S-@	252	1,855	29,266	0.027
			Brdg	27 (169)	1,208	4,721	0.006
			Both	279 (169)	1,863	31,440	0.030
b07s	470 /49	286	S-@	1,876	91,790	6.47E+6	11.101
			Brdg	160 (2,881)	44,150	973,386	1.246
			Both	2,036 (2,881)	93,529	7.38E+6	13.125
b08	189 /21	245	S-@	736	22,841	708,380	0.916
			Brdg	113 (659)	17,037	227,290	0.270
			Both	849 (659)	24,968	911,375	1.250
b09	188 /28	205	S-@	754	20,996	682,857	0.765
			Brdg	101 (949)	10,658	293,479	0.288
			Both	855 (949)	21,919	954,592	1.149
b10	207 /17	244	S-@	812	26,631	543,143	0.588
			Brdg	111 (1,036)	14,353	78,609	0.100
			Both	923 (1,036)	27,429	594,872	0.692
b11s	513 /31	267	S-@	2,118	106,249	3.63E+6	5.151
			Brdg	169 (1,931)	45,576	451,829	0.633
			Both	2,287 (1,931)	107,485	4.01E+6	5.865
b12	1168 /131	327	S-@	4,752	129,983	4.39E+6	10.428
			Brdg	643 (17,823)	81,406	5.81E+6	11.913
			Both	5,395 (17,823)	137,037	1.01E+7	16.752
b13s	393 /53	444	S-@	1,498	52,050	3.88E+6	7.353
			Brdg	164 (3,459)	27,666	1.87E+6	2.662
			Both	1,662 (3,459)	55,327	5.68E+6	6.593
b14s	5021 /245	2272	S-@	21,082	9.47E+6	3.86E+8	967.23
			Brdg	1,822 (69,674)	2.20E+6	2.03E+7	71.85
			Both	22,904 (69,674)	9.48E+6	4.01E+8	1108.00
b15s	9343 /449	515	S-@	40,688	2.97E+6	2.98E+8	1165.75
			Brdg	5,361 (217,797)	1.36E+6	4.58E+7	142.27
			Both	46,049 (217,797)	3.18E+6	3.43E+8	1479.77
b20s	9910 /490	3033	S-@	42,004	2.45E+7	3.18E+9	12909.62
			Brdg	3,159 (259,423)	4.11E+6	1.06E+8	366.20
			Both	45,163 (259,423)	2.46E+7	3.27E+9	15593.22

circuit element. The component evaluations increase on the order of $n^{1.2}$, where n is the functional evaluations. Trend lines indicated on figure 5 are shown to highlight the fact that bridge fault test runs actually have less fault effects to propagate. This comes from the fact that during the simulation, many fault aliases with the same concurrent ID exist after fault insertion, and fault effects with the same ID and state activation time can re-converge on a given element, requiring now only one evaluation. This is referred to as fraternal processing [9].

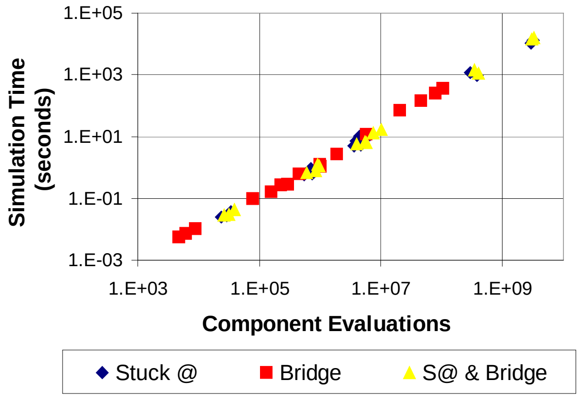

Figure 6. Simulation time vs. component evaluations for stuck-at and bridge fault sets

Another interesting observation is that simulation time very closely correlates to the number of component evaluations (figure 6). Fundamental to concurrent simulation is the principal that by only propagating state differences the number of element evaluations is reduced. Regardless of fault source type, the effects are propagated in the same way. This is why all of the faulting scenarios follow the same trend.

6. Conclusions

This paper has presented a method using concurrent simulation to efficiently fault simulate large networks using a variety of fault models. The most significant advantage of multiple functional faulting is the ability to efficiently represent physical faults in various ways and at different levels in a single environment. The efficiency of the algorithms used in our simulation environment has been demonstrated using some of the latest real world benchmarks.

We have also found that the bridge fault models used have similar detectability as the exhaustive stuck-at faults. Patterns generated for stuck-at faults produced the same, if not increased coverage for these n-way bridge faults.

Utilizing the function list within a concurrent simulation environment, the fundamental challenge of simultaneously employing multiple fault models has been addressed.

Acknoledgements

We would like to thank Mr. Jamie Heller of ASIC-Alliance and Dr. Elizabeth Rudnick of the University of Illinois for their valuable assistance. Also Paul Olson and Michael Mattioli helped with the experimental process, and we thank them.

References

[1] F. Corno, M. Sonza Reorda, and G. Squillero, “RT-Level ITC 99 Benchmarks and First ATPG Results”, *IEEE Design & Test of Computers*, July-August 2000. http://www.cad.polito.it/tools/itc99.html

[2] D. Lavo, B. Chess, T. Larrabee, I. Hartanto,“Probabilistic Mixed-Model Fault Diagnosis”, *Proceedings of the International Test Conference*, 1998, pp. 1084-1093.

[3] S. Venkataraman, I. Hartanto, W.K. Fuchs, E.M. Rudnick, S. Charkravarty, and J.H. Patel, “Rapid Diagnostic Fault Simulation of Stuck-at Faults in Sequential Circuits using Compact List”, *Proceedings of the ACM/IEEE Design Automation Conference*, June 1995, pp. 133-138.

[4] J. Rearick and J.H. Patel, “Fast and Accurate Bridging Fault Simulation,” *International Test Conference*, 1993, pp. 54-62.

[5] S. Bose, V.D. Agrawal and T.G. Szymanski, “Algorithms for Switch Level Delay Fault Simulation", *International Test Conference*, 1997, pp. 982-991.

[6] T. A. DeLong, B.W. Johnson and J. A. Profeta, “A Fault Injection Technique for VHDL Behavioral-Level Models,” IEEE Design and Test of Computers, pp. 24-33, Winter 1996.

[7] Ernst Ulrich and Thomas Baker, “The Concurrent Simulation of Nearly Identical Networks”, *Proceedings Design Automation Conference*, June 1973, Vol. 6, pp. 145-150.

[8] Pierluca Montessorro and Silvano Gai, “CREATOR, General and Efficient Multilevel Concurrent Simulation”. *First Great Lakes Symposium on VLSI*, March 1991, Kalamazoo, MI, pp. 43-48.

[9] D. Machlin, D. Gross, S. Kadkade, and E. Ulrich, “Switch-Level Concurrent Fault Simulation Based on a General Purpose List Traversal Mechanism”, *Proceedings, IEEE International Test Conference (ITC)*, 1988, pp. 574-581.

[10] R. Byrant, “Algorithmic Aspects of Symbolic Switch Network Analysis”, *IEEE Transactions on Computer Aided Design*, July 1987, Vol. 6, pp. 618-633.

[11] Abramovici, R., M.A. Breuer, and A.D. Friedman, *Digital Systems Testing and Testable Design*, Computer Science Press, Oxford, 1990, p. 96.

[12] F. Brglez, D. Bryan, and K. Koziminksi, “Combination Profiles in Sequential Benchmarks for Sequential Test Generation”, *Proceedings IEEE International Symposium on Circuits and Systems*, May 1989, pp. 1929-1934.

[13] D. Krishnaswamy, M.S. Hsiao, V. Saxena, E.M. Rudnick, J.H. Patel, and P. Banerjee, “Parallel Genetic Algorithms for Simulation-Based Sequential Circuit Test Generation”, *Proceedings of the International Conference on VLSI Design*, January 1997, pp. 475-481.