Statistics on Concurrent Fault and Design Error Simulation

Brian Grayson, Saghir A. Shaikh, and Stephen A. Szygenda

TR ECE-PDS-1995-012 September 1995

Parallel & Distributed Systems group
Department of Electrical & Computer Engineering
University of Texas at Austin
Austin, Texas 78712

Statistics on Concurrent Fault and Design Error Simulation

Brian Grayson* Saghir A. Shaikh Stephen A. Szygenda
bgrayson@pine.ece.utexas.edu saghir@mail.utexas.edu szygenda@uts.cc.utexas.edu

Department of Electrical and Computer Engineering
The University of Texas at Austin
Austin, TX 78712-1084

October 2, 1995

Abstract

Basic data of the nature presented here on fault and design error simulation processes have not been previously reported. Experiments are performed on c_sim, a gate level concurrent simulator developed at the University of Texas at Austin. Three types of statistics are considered: event-based statistics, gate-evaluation statistics and memory requirements. These statistics are important for design verification researchers and engineers for numerous reasons. For example, they help simulator developers tune up or optimize their concurrent simulators. They also fulfill the increasing need for experimental data concerning design error simulation. Most importantly, these statistics provide guidance to hardware accelerator designers in evaluating and comparing various design options.

1 Introduction

Design verification of digital circuits includes logic, fault and design error simulation [1]. Logic simulation is performed to verify the functionality of the design, and fault and error simulations are done to test thedesign for anticipated manufacturing defects and design errors. The complexity of fault and error simulation is between G and G^3, where G is the number of primitives in the circuit. The cost of simulation for typical circuits is extremely large.

Fault simulation is a tool that defines how well the test vectors used to verify a finished product will detect a manufacturing defect. Fault simulation usually provides coverage for a limited set of faults. In general, it includes logical faults modeled by simplistic fault models i.e. considers signals being stuck at a fixed logical value of 0, 1, or X. Fault simulation also assumes that the components are fault free and only their interconnections are affected. Thus, simulation of classical stuck-at faults does not provide a true measure of coverage of all possible design faults and errors.

A new measure of design verification coverage, based on the number of design errors detected, was introduced by Kang and Szygenda [1]. A design error is defined as a"...design change that causes a different output value from the desired value for some input pattern." [1]. Some techniques to model design errors and use them for design validation have been presented in [1, 2, 3]. The design error models defined by Kang and Szygenda are: signal, gate, local, and include. Signal and gate errors model design changes in signals and gates respectively. Local errors represent a double-error situation: both a gate error and a signal error on one of the gate's signals. Include error considers a design error involving a change in the number of inputsto a gate.

Basic algorithms of error simulation are similar to those of fault simulation. The main difference is the use of new error models instead of classical fault models. Nevertheless, both fault simulation and design error simulation are vital but expensive tasks of the design cycle.

Concurrent fault simulation is one of the three popular techniques which exploit parallelism (the other two are Parallel and Deductive fault simulation). Concurrent fault simulation is popular because of its ef-

*Brian Grayson is supported by a National Science Foundation Graduate Fellowship.

ficiency, generality and accuracy. In various existing implementations of concurrent fault simulation [4, 5, 6, 7], the basic algorithm is the same; they usually incorporate various heuristics to increase the efficiency. The first concurrent design error simulator, named *c_sim*, was developed by Redlien [8] at the University of Texas at Austin. This simulator is capable of logic, fault, and error simulation. The program was developed in C++ and handles gate-level simulation. This simulator was used to collect the data presented here. In this paper we present statistics related to fault and design error simulation. While statistics on logic simulation for some specific (non-ISCAS) examples exist [9], the data presented here on fault and design error simulation of ISCAS circuits have not been previously reported. We use nine combinational circuits, collectively known as ISCAS-85 [10], and two additional circuits named *C95* and *C74ls181* as our examples. A summary of these circuits is presented in Table 1. These circuits are widely accepted as standard benchmarks for comparing various design automation tools. We present three types of statistics: event-based statistics, gate-evaluation statistics and memory requirements. The motivating factors for collection of this information and its analysis are:

- The performance of concurrent fault and error simulation can be tuned by using various heuristics. Information obtained from the statistics on these benchmark circuits may help in developing new software techniques or improving existing ones to accelerate simulation.

- Increased interest in design error simulation creates a need for this kind of experimental data.

- In order to reduce simulation time, hardware acceleration techniques have frequently been used for fault simulation. Some of these techniques include the development of special purpose hardware optimized for the concurrent simulation algorithm [11, 12, 13]. These statistics can be helpful for hardware accelerator designers in evaluating and comparing various design options.

This paper is organized in four sections. The introduction is followed by Section 2, which contains a brief overview of *c_sim* and describes the methodology for the collection of experimental data and statistics. Section 3 presents the results along withtheir analysis. Section 4 provides a summary, and is followed by the bibliography. An appendix contains optimization notes concerning *c_sim*, many of which are applicable to concurrent fault simulators, and fault simulators in general.

2 Methodology

The concurrent simulator used for this research is *c_sim*, developed at the University of Texas at Austin [8]. This simulator uses a time based, event driven, compiled model algorithm. It is capable of functional, fault and design error simulation. It accepts the circuit description in Standard Intermediate Format (SIF) and simulation patterns in SIM format [8]. The model library consists of C++ functions.

This simulator uses three time queues to schedule good, faulty and design error events (fault simulation and design error simulation are not performed concurrently, but separate queues are used for statistical purposes). The algorithm for design error simulation is the same as that for fault simulation except that erroneous events (events resulting from a design error) are scheduled into the design error time queue. A time queue consists of a collection of time slots; each time slot is a linked list that holds all the events scheduled to happen at one instance of simulation time. During the simulation the number of events in a time slot and time queue are variable.

Many of the desired statistics simply required the proper insertion of various counting mechanisms into the simulator's code. Initially, all the raw data was gathered at once and then placed in files for processing later. However, the extra overhead of saving all the raw data, and then re-processing it, led us to modify the code to compile much of the statistics on-the-fly. This resulted in the current version of the code, which calculates the maximums and averages, and gathers distributions into buckets, during simulation run-time, and produces a much-shorter, ready-to-use statistics file. Not only is the cost of this on-the-fly statistics collection significantly less than the original two-step (generation and processing) method, but it also has little effect on the running time of the simulator.

Nine ISCAS benchmark circuits and two example circuits were chosen for simulation. For a given circuit, the same set of random input patterns were used for both the fault and the design error simulations. Table 1 on the following page shows the summary of the example circuits used and Table 2 on page 4 shows the basic simulation statistics.

The statistics presented here can be divided into three different categories: event-based statistics, gate-evaluation statistics and memory requirements. Event-based statistics were the easiest to implement.

Table 1: Summary of the Benchmark Circuits

Name of Circuit	Function[a]	Number of Gates	Number of Primary Inputs	Number of Primary Outputs
C95	Example	27	15	7
74ls181	ALU	63	14	8
C432	Priority Decoder	160 (18 XOR)	36	7
C499[b]	ECAT	202 (104 XOR)	41	32
C880	ALU and Control	383	60	26
C1355[b]	ECAT	546	41	32
C1908	ECAT	880	33	25
C2670	ALU and Control	1193	233	140
C3540	ALU and Control	1669	50	22
C5315	ALU and Selector	2307	178	123
C6288[c]	16-bit Multiplier	2406	32	32
C7552	ALU and Control	3512	207	108

[a] ECAT – Error Correction and Translation

[b] Circuit *C499* and *C1355* are functionally equivalent. All XOR gates of *C499* have been expanded into their *NAND*-gate equivalents in *C1355*

[c] The simulaion on this circuit could not be performed because it required more virtual memory than available at the time of experimentation.

For each of the queues – good events, faulty events and error events – the statistics can be split into two categories. The investigated categories are:

- average, maximum, and distribution of the number of events in each timeslot

- average, maximum, and distribution of the number of events in the wholetime queue

Gate-evaluation statistics involved adding a procedure call before each gate evaluation to a helper function that kept track of which gate and which gate-type was being evaluated. This function also recorded whether the evaluation was a good or a faulty evaluation and at what time the evaluation occurred. The averages and maximum evaluations per time value per gate, along with their types, were saved in the statistics file.

Some UNIX resource-usage statistics were also gathered. The most important of these is the maximum amount of memory used. Many other details, such as the number of page-faults or I/O operations, are implementation-dependent, machine-dependent, and load-dependent, and thus are not presented.

3 Results and Analysis

This section contains all three types of statistics obtained along with an analysis of the results.

3.1 Event-based Statistics and Analysis

Event-based statistics are summarized in Table 3 on the following page. Good event statistics for fault simulation and design error simulation are the same due to the use of the same input patterns and thus are mentioned only once. Good events queue consists of two kinds of events; input vector-induced events and propagation events. The numbers presented in Table 3 in the good events section are for propagation events only. The values for input-vector induced events are known as they can be determined by multiplying the number of input patterns and number of primary inputs. Also, due to the use of unit delays, the only active time slots in the fault and design error queues are the current time slot (from which events are being taken) or the next slot (into which propagation events are inserted). Because of this, the overall queue statistics are essentially the same as the slot statistics. Thus Table 3 does not contain entries of average and maximum queue lengths for all queue because they are equal to the slot average and maximum sizes.

Table 2: Basic Simulation Data

Name of Circuit	No. of Modeled Faults	No. of Modeled Design Errors	No. of Input Patterns	Fault Coverage [%]	Design Error Coverage [%]	Fault Simul. Time [Sec.]	Design Error Simul. Time [Sec.]
C95	158	335	50	98.7	95.8	1.09	4.16
74ls181	344	814	1000	100	90.4	45.93	130.82
C432	818	1837	1000	100	96.6	128.67	266.21
C499	1206	2537	1000	100	100	181.53	274.89
C880	1756	4130	1000	98.7	90.2	271.96	681.05
C1355	2374	5953	1000	100	99.2	506.64	897.66
C1908	3813	8013	1000	98.8	97.3	1234.98	2197.21
C2670	5725	12071	1000	89.2	87.7	1933.04	3594.32
C3540	7136	15641	1000	97.4	92.8	1807.46	4342.81
C5315	10549	23228	1000	99.4	97.9	2827.52	3961.27
C7552	15231	33633	1000	96.9	95.5	5781.70	8617.20

Table 3: Fault/Design error simulation event statistics

Name of Circuit	Good (Propagation) Queue			Fault Queue			Design Error Queue		
	No. of Events	Avg. Slot Length	Max. Slot Length	No. of Events	Avg. Slot Length	Max. Slot Length	No. of Events	Avg. Slot Length	Max. Slot Length
C95	1969	7.10	37	3612	26.55	363	10571	42.97	1022
74ls181	109040	14.38	48	8586	49.34	905	240178	43.8	2884
C432	263907	22.25	81	62570	38.36	1255	415647	36.4	2610
C499	344203	36.72	112	131050	82.52	8739	342966	62.2	14818
C880	536491	35.43	222	117421	26.48	1765	1379777	95.2	5695
C1355	1038663	51.81	208	532049	193.40	13847	1710585	194.8	32025
C1908	1717177	66.53	214	2164323	117.69	8102	5461856	214.3	17720
C2670	2026712	98.97	400	2992050	157.38	6740	8789541	416.7	15915
C3540	2788624	86.25	380	2829237	93.14	9983	10913467	341.2	22137
C5315	4991918	137.45	868	1610339	93.98	12867	5325505	167.5	30058
C7552	7205929	195.0	1214	4351626	237.15	36068	14099894	396.2	72302

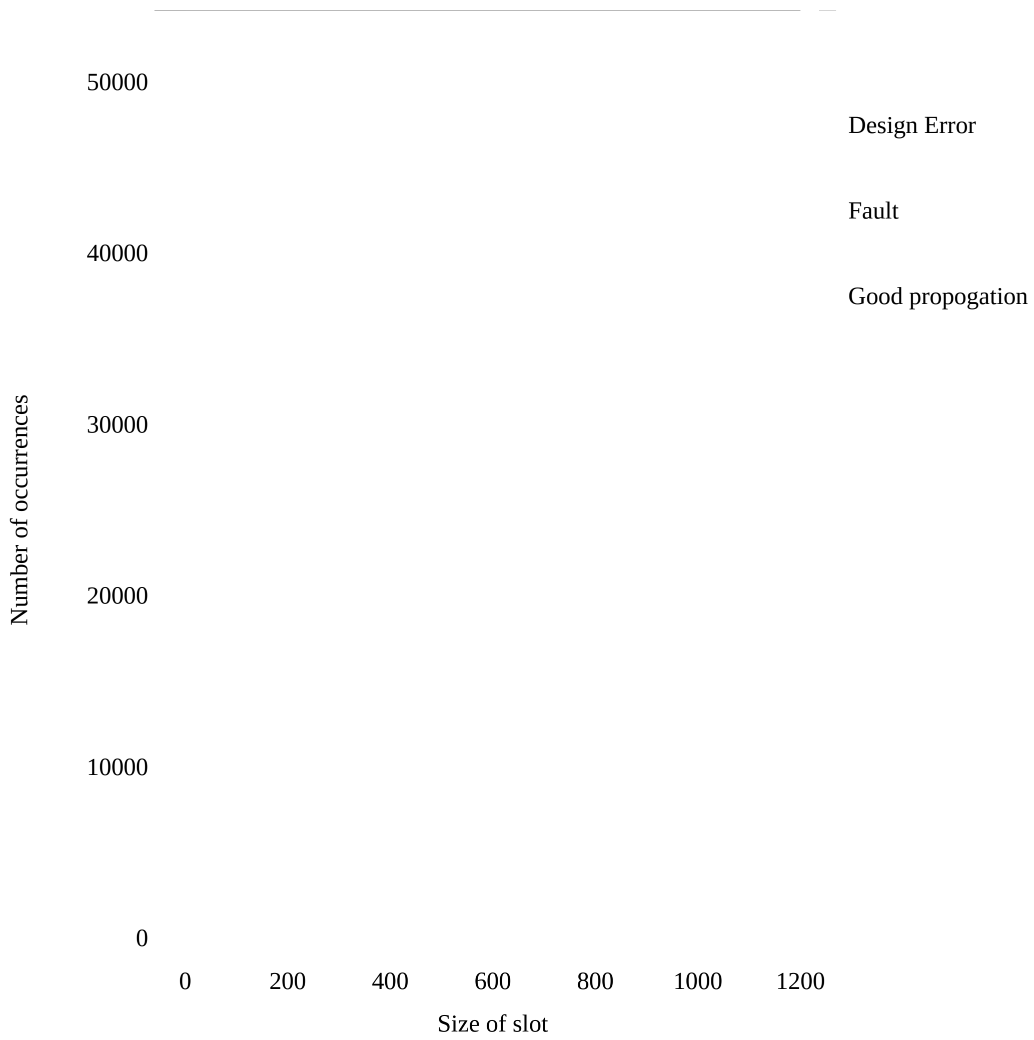
Note that for the design-error simulation, the number of error events typically outnumber the corresponding fault events by a factor between 3 and 12. This can be explained by noting that each gate in the circuit has more design errors than faults. Hence, the number of error events should be much larger than the number of faulty events. This also causes the average and maximum number of events per slot for error events to be larger than for faulty events.

One major difference in the event statistics between the good and faulty/error events is in the maximum number of events per slot. Values for faulty and error events are much larger than the corresponding values for good events. In the initial phase of the simulation many faults or errors are stimulated and propagated, resulting in larger time slots. Since the average slot lengths of good event queue and the faulty/error event queues are roughly comparable, these large maximum sizes in the faulty/error event queue near the beginning of the simulation must be offset by smallsizes in the later phase of simulation.

It is also important to know the distributions of the sizes of time slots and the overall queues. We have presented these distributions for the largest circuit, *C7552*, for good, faulty and error events in Figure 1.

In Figure 1, the input vector events have not been

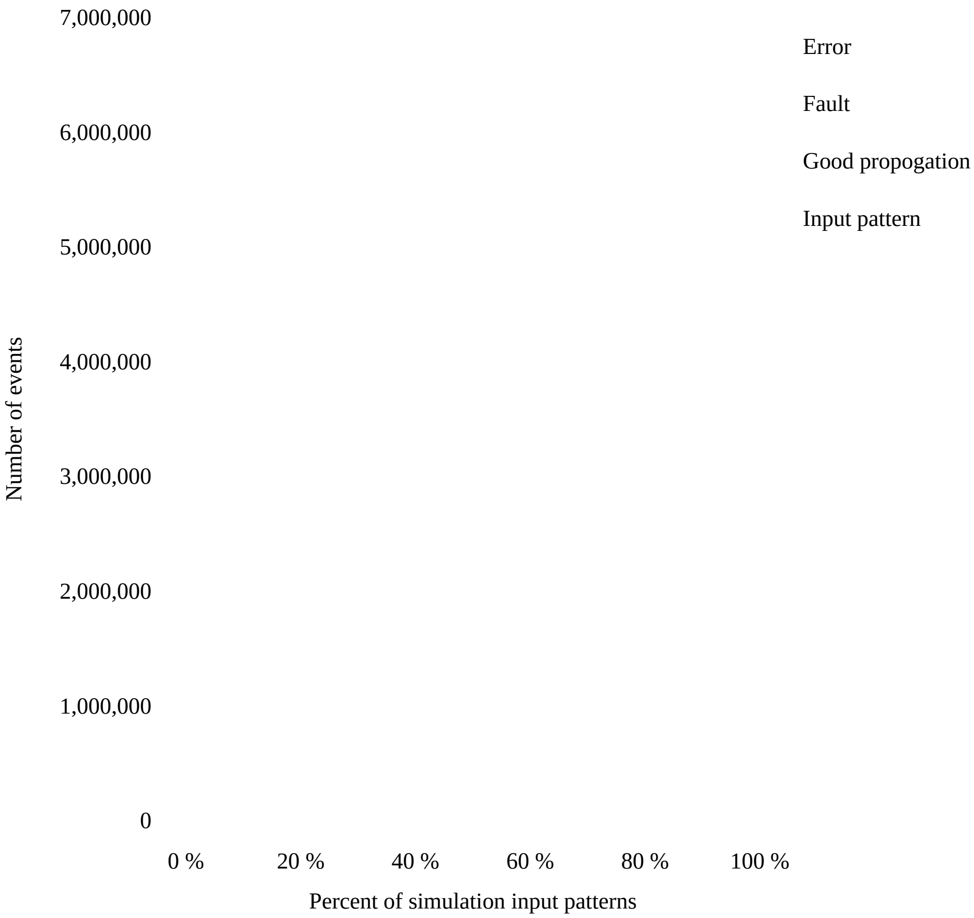
Figure 1: Good, Faulty, and Error Event Slot Distribution for a simulation of *C7552*



included – there is always one event for each PI, so all of the used slots have exactly the same size. The figure shows that even though the slot size can grow to be quite large (more than 60000 events in some slots), most of the time the slot size is quite small.

It is interesting to observe the distribution of events with the passage of time, i.e. through the course of the simulation. In one experiment on circuit *C7552* we have collected the number of good, faulty and erroneous events generated for each of the ten segments (the first 10% of all input patterns, the next 10% of the input patterns, etc.) of the simulation. These results are presented in Figure 2. The value for input pattern events is constant and quite small – on this figure, these bars produce a single horizontal line at the bottom of the graph. This time distribution of events shows that in the early phases of fault simulation, faulty error events greatly outnumber good events (because no faults/errors have been detected, so many faults and errors are being excited by the input vectors), whereas near the end of the simulation, good events greatly outnumber faulty events and slightly outnumber error events (because only a few undetected faults/errors remain to be simulated). This observation can be exploited in accelerating the simulation by running a fault simulation in two phases,

Figure 2: Good, Faulty, and Error Event distribution versus time for *C7552*



with the first phase exploiting the parallelism between faults, and the second exploiting the parallelism between patterns.

3.2 Evaluation-based Statistics and Analysis

In order to limit the length and content of this paper, we provide a summary of the evaluation statistics. The number of inputs to each gate was ignored for these statistics, that is, both a 3-input NAND and a 2-input NAND fall into the NAND category. The data includes the total number of evaluations, maximum number of evaluations for one time slot, and average evaluations per gate per time (for each type of gate). The value of average evaluations per gate per time was obtained as follows. In addition to the various counters incremented on each evaluation, for a given unit oftime, whenever a gate was evaluated, a global counter (the gate activation counter) was increased by one, irrespective of the number of evaluations ofthat gate performed in that unit time. Thus, the average evaluations per evaluated gate per unit time can be calculated by the number of evaluations of that type of gate divided by the total number of gate activations for that type.

Evaluation statistics for good events were the same in fault and design error simulations. Hence, they are presented only once in Table 4. Results for fault and error evaluation statistics are presented in Table 5 and

Table 4: Fault/Design error simulation good-evaluation statistics

Name of Circuit	No. of Eval	Overall Max.	Overall Avg.	PO Avg.	AND Avg.	OR Avg.	NAND Avg.	NOR Avg.	XOR Avg.	NOT Avg.	Buffer Avg.
C95	1972	4	1.197	1.075	-	-	1.345	-	-	1.0	-
74ls181	109041	4	1.33	1.102	1.476	-	1.261	1.316	1.132	1.0	-
C432	263911	7	1.08	1.125	1.753	-	1.071	1.006	1.032	1.0	-
C499	344206	4	1.257	1.022	1.310	1.242	-	-	1.393	1.0	-
C880	536494	4	1.201	1.049	1.250	1.287	1.324	1.058	-	1.0	1.0
C1355	1038664	5	1.184	1.066	1.358	1.288	1.164	-	-	1.0	1.0
C1908	1717178	8	1.089	1.122	1.195	-	1.107	1.040	-	1.0	1.0
C2670	2026715	5	1.095	1.070	1.168	1.138	1.110	1.038	-	1.0	1.0
C3540	2788626	5	1.092	1.178	1.111	1.135	1.147	1.090	-	1.0	1.0
C5315	4991920	9	1.075	1.146	1.132	1.053	1.063	1.057	-	1.0	1.0
C7552	7205930	5	1.084	1.163	1.187	1.081	1.066	1.054	-	1.0	1.0

Table 5: Fault simulation faulty-evaluation statistics

Name of Circuit	No. of Eval	Overall Max.	Overall Avg.	PO Avg.	AND Avg.	OR Avg.	NAND Avg.	NOR Avg.	XOR Avg.	NOT Avg.	Buffer Avg.
C95	2762	55	5.70	8.06	-	-	6.52	-	-	1.86	-
74ls181	6657	121	6.20	7.85	5.27	-	9.47	6.41	7.43	1.65	-
C432	52007	109	4.06	5.28	6.95	-	4.46	1.83	5.15	3.50	-
C499	113485	294	9.54	6.66	10.38	10.90	-	-	7.26	9.45	-
C880	77052	137	3.95	4.90	3.69	4.03	3.79	4.14	-	4.39	4.81
C1355	495700	595	11.23	11.37	15.29	18.73	8.19	-	-	11.27	11.28
C1908	1881497	236	7.23	6.19	10.25	-	7.14	9.41	-	4.51	4.79
C2670	2212682	252	3.60	6.74	4.40	2.84	3.44	5.65	-	3.28	2.61
C3540	2191098	804	5.38	13.46	5.18	5.86	4.73	5.61	-	6.84	7.06
C5315	1357406	306	5.05	6.30	4.96	5.04	5.35	4.27	-	4.69	4.20
C7552	3416514	376	5.28	8.57	6.35	4.50	4.90	5.84	-	4.34	3.95

6. In the case of good evaluations, the average number of evaluations per gate per time should be close to 1. The results show that this value is higher than unity. This phenomenon can be explained by one or more of the following reasons:

1. There are zero delays for some of the gates. However, these circuits had no zero-delay components, so this reason can be ignored.
2. This simulator is not collapsing events, i.e. if two events for one signal are scheduled for the same time, (due to a zero-delay device such as a PI or PO, or peculiarities of the simulation algorithm), the simulator performs both event calculations.
3. This simulator incorporates the one-pass technique in which elements are evaluated as soon as one of its input values is updated. Thus a gate with two inputs updated is evaluated twice. On the other hand, in the two-pass technique, all circuit elements for the current time are updated first, and then evaluated once only if one or more inputs have changed. Results show that the value for average good-evaluation per unit gate per unit time is around 1.16. This suggests that the two-pass technique could at best improve efficiency by about 16%.

In fault/error simulation the overall maximum and average evaluation values for both the faulty and the

Table 6: Design Error simulation error-evaluation statistics

Name of Circuit	No. of Eval	Overall Max.	Overall Avg.	PO Avg.	AND Avg.	OR Avg.	NAND Avg.	NOR Avg.	XOR Avg.	NOT Avg.	Buffer Avg.
C95	7716	142	6.03	9.93	-	-	9.85	-	-	1.60	-
74ls181	125104	357	4.32	12.45	3.37	-	13.33	4.65	11.98	1.07	-
C432	282804	232	3.06	7.94	11.33	-	4.76	2.49	4.92	2.44	-
C499	288377	437	8.68	10.24	11.65	16.05	-	-	5.34	8.66	-
C880	777531	337	2.94	8.37	2.50	2.76	2.76	3.20	-	2.65	8.27
C1355	1499743	1367	8.70	19.57	26.78	33.93	4.72	-	-	19.04	19.48
C1908	4530179	490	8.06	8.96	14.72	-	7.54	4.24	-	4.81	6.60
C2670	6289624	613	6.41	10.92	8.12	5.98	6.98	3.02	-	4.97	3.94
C3540	8093356	1642	6.67	21.10	6.39	4.20	8.92	5.59	-	5.67	6.38
C5315	4058764	541	5.01	9.43	7.47	4.19	5.26	2.99	-	4.00	6.02
C7552	10460456	630	6.71	13.13	10.74	6.31	6.24	3.23	-	4.53	6.00

Table 7: Memory Statistics

Name of Circuit	Fault simulation memory usage (MB)	Design error simulation memory usage (MB)
C95	0.270	0.324
74ls181	1.200	1.315
C432	2.683	2.822
C499	3.301	3.621
C880	4.415	4.739
C1355	6.820	8.196
C1908	3.084	3.768
C2670	16.228	17.039
C3540	4.674	5.767
C5315	13.386	14.987
C7552	16.527	19.300

error evaluations are much greater than good evaluation. As previously mentioned, the averages are much higher during the early phase of simulation, and decrease in the later phase. For example, in the first of the ten segments of the simulation for *C1908*, the average faulty evaluations per gate per time was over 10. During the last segment of the simulation, the average was only 2.2 faulty evaluations per gate per time. The large averages (the average is over 5 for most circuits, which implies averages around 10 or more for the first phase) indicate that an early-phase simulator could greatly increase the faulty/error evaluation performance across a relatively wide variety of circuits by being able to evaluate around ten faulty/error evaluations at once. This ability will not help as much in the later phase of the fault simulation. However, the average number of evaluations per gate per time does not fall as quickly as does the number of evaluations performed, so that, even though near the end of the simulation fewer evaluations are being performed, these evaluations tend to occur at the same gates more often than the evaluations during the earlier phase.

3.3 Memory Statistics

The most useful memory statistic is the maximum amount of memory needed at any one point. These statistics are provided in Table 7. Memory usage in general increases with the size of the circuit, although some circuits use more memory than a larger circuit. This is due to the fact that memory usage is a function of the number of gates, number of faults/errors simulated and the topology of the circuit, and not simply the number of gates.

It was observed that more than 90% of the memory for the simulation of *C7552* was used for storing data. This proves that data structures play a very important role in memory usage. The data structures used in *c_sim* are not always space-efficient, nor does *c_sim* do any internal memory-reuse optimizations through freelists. Significant reductions in memory usage are expected once *c_sim* has been further optimized. See the Appendix for more information on this topic.

4 Summary

Three types of statistics were gathered: event-based statistics, gate-evaluation statistics and memory requirements. Statistics related to both the fault simulation and the design error simulation are presented here. These statistics can provide guidance to simulator designers and hardware accelerator designers for evaluating and comparing various design options.

The average and maximum sizes of the time slots can be used by simulator developers to optimize their time-queue mechanisms for the common or average case. The time-distribution of the various kinds of events also show that a simulator could be optimized in two phases: first, optimize for the case where many faults are activated, when the processing time is dominated by the faulty activity. Second, optimize for the case when very few faults are propagated, and thus most of the processing time is spent handling good events.

The evaluation statistics showed that on average, most gates that are evaluated due to a fault or design error event are likely to be evaluated several more times in the save time step, with different faulty/erroneous input values. For example, the average parallel gate evaluation is typically more than 5 and less than 11. This demonstrates that a hardware accelerator that provided 8 to 16 evaluators (used to simulate in parallel all of the faulty events at a particular gate) would be used at around 50% to 100% of capacity, while an accelerator with 32 evaluators would on average be used at less than 20% to 40% of capacity for most simulations.

Acknowledgements

The authors would like to thank Dr. Jacob Abraham and Dr. Craig Chase for the use of their computers, and the anonymous reviewers for ICCD 95 for their valuable comments.

References

[1] Kang, S., and S. Szygenda, "Design Validation: Comparing Theoretical and Empirical Results of Design Error Modeling," *IEEE Design and Test of Computers*, pp. 18-26, March, 1994.

[2] Wang, Y., P. Chung, I. Hajj, "Logic Design Error Diagnosis and Correction," *IEEE Transactions on Very Large Scale Integration*, Vol. 2, No. 3, pp. 320-332, September 1994.

[3] Zhang, Q., and C. Trullemans, "Logic Verification of Incomplete Functions and Design Error Location," *Correct Hardware Design and Verification Methods, Lecture Notes in Computer Science 683*, Springer Verlag, pp. 68-79, May 1993.

[4] Schuster, M., and R. Bryant, "Concurrent Fault Simulation of MOS Digital Circuits," *Proc. of Conference on Advanced Research in VLSI*, M.I.T., pp. 129-138, January 1984.

[5] Davidson, S., and J. Lewandowski, "ESIM/AFS-A Concurrent ArchitecturalLevel Fault Simulator," *Proc. of IEEE Intl. Test Conf.*, pp. 375-383, 1986.

[6] Rogers, W., J. Guzolek, and J. Abraham, "Concurrent Hierarchical Fault Simulation: A Performance Model and Two Optimizations," *IEEE Transactions of Computer-Aided Design*, Vol. 6, No. 5, pp. 848-862, September 1987.

[7] Gai, S., and L. Montessoro, "Creator: New Advanced Concepts in Concurrent Simulation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and System*, Vol. 13, No. 6, June 1994.

[8] Redlien, P. "A Technique for Concurrent Design Error and Fault Simulation," Master's Thesis, The University of Texas at Austin, August 1993.

[9] Wong, K. F., M. A. Franklin, R. D. Chamberlain and B. L. Shing, "Statistics on Logic Simulation," *Proc. of 23rd IEEE/ACM Design Automation Conference*, pp. 13-19, 1986.

[10] Brglez, F., and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in FORTRAN," *Proc. of IEEE International Symposium on Circuits and Systems; Special Session on ATPG and Fault Simulation*, pp. 695-698, June 1985.

[11] Chan, T., and L. Edwin, "MegaFAULT: A Mixed-Mode, Hardware Accelerated Concurrent Fault Simulator," *Proc. of IEEE ICCAD.*, Washington, DC, pp. 394-397, 1986

[12] Agrawal, P., V. Agrawal, K. Cheng, R. Tutundjian, "Fault Simulation in a Pipelined Multiprocessor System," *Proc. of IEEE Int. Test Conf.*, pp. 727-734, September 1989.

[13] "Paradigm," *Zycad Product News*, Feb. 1994.

Appendix: Optimizations

Several of the algorithms in the original implementation of *c_sim* were coded in inefficient ways. Some of the improvements are mentioned below, partially to demonstrate the importance of good algorithm implementation.

- The original version of *c_sim* performed its insertion of events by starting at the head of the event queue, and iterating down the queue. Since initially *c_sim* loads in all of the input vectors, and since input vectors are typically ordered with the first time values first, the initialization phase of *c_sim* took $O(n^2)$ time to insert n events into the event queue. This can be fixed by either making a better `insert` function, which checks against the head value, and against the tail value, or by simply reversing the order of all events in the input file. The latter method was used for the experiments here.

- The hash tables, used by the circuit description parser, used a poor hashing function – only the last two characters of a string were actually used (most likely due to a misunderstanding of the hashing function). Because of this, strings with 'close' tailing characters hashed to 'close' entries. Since many circuits are machine-generated, they frequently contain names such as `NAND103` (where all trailing characters are numbers) or `103GATE` (where all trailing characters are identical), and thus similar trailing characters will be a common occurrence. For some circuits, this original hashing function led to a handful of slots out of 1024 being used. The hashing function was improved to make all of the bits of the string influence the final result.

- The event queue uses the "time wheel" technique to improve performance. This technique uses a fixed-length array of linked lists, used as a circular buffer, to hold all of the events occurring in the next t time units. Events scheduled for a time past t time units from the current time are placed on an "overflow" list. The original version of the program used an array of about 400 slots. Since all of the simulations in this paper used a unit delay, and since input patterns were frequently spaced 100 or even 1000 time units apart, very few of the 400 time slots were actually used. However, the simulator needs to look at each time slot in order to find the next event. Checking all of the empty time slots adds unnecessary work for the CPU. For the experiments presented here, the size of the array was changed to be 16[1].

- Most of the memory usage is used by the `net` and the `event` classes. Therefore, the size of a single object of these classes greatly impacts the total memory usage. Originally, all of the data members for these classes were 32-bit integers. By changing several of them to 16-bit integers, and allowing the compiler to pack the classes, the overall memory usage dropped by nearly 50%.

[1]Since the devices all used unit delay, an array of size 2 would have been sufficient!