To appear in the Proc. 4th IEEE Int. High Level Design Validation and Test Workshop, La Jolla CA, Nov. 1998.

Error Simulation with Conditional Error Models

DAVID VAN CAMPENHOUT, TREVOR MUDGE, AND JOHN P. HAYES

Department of Electrical Engineering and Computer Science
The University of Michigan, Ann Arbor, MI 48109-2122, USA
{davidvc, tnm, jhayes}@eecs.umich.edu

Abstract

Error-oriented functional design verification attempts to uncover functional bugs by applying test sequences to the design that are targeted at instances of synthetic design error models. Conditional error (CE) models have recently been shown to be powerful error models for such an approach. Practical use of higher order CE models has been impeded by the computational cost associated with test generation and error simulation as the number of error instances is superlinear in the size of the circuit.

This paper presents a new algorithm CESIM for efficient error simulation with CE's. Experiments using the ISCAS'89 benchmarks are presented that show the improved performance of CESIM compared with a state-of-the-art fault simulator.

Keywords: *High Level Design Validation, High Level Design Error Modeling, High Level ATPG/Fault, Validation of Microprocessors, Simulation-based Verification, Error models and verification test.*

1. Introduction

Design verification is considered one of the most serious bottlenecks for multimillion-gate microprocessor designs. There are two broad approaches to hardware design verification: formal and simulation-based. Formal methods try to verify the correctness of a system by using mathematical proofs. Simulation-based design verification tries to uncover design errors by detecting a circuit's faulty behavior when deterministic or pseudo-random tests (simulation vectors) are applied [Ahar91, Chan95, Tayl98]. The effectiveness of verification test suites is quantified by coverage metrics that include code coverage measures from software testing [Bez90], finite-state machine coverage [Ho96], architectural event coverage [Tayl98], and observability-based metrics [Fall98]. A shortcoming of all these metrics is that the relationship between the metric and the detection of classes of design errors is not well specified or understood.

An alternative verification approach draws on the similarity between hardware design verification and physical fault testing [Abad88, AA95, VC98]. In this approach, synthetic error models are derived from empirical design error data, and physical fault testing techniques are adapted to generate test sets for the synthetic errors.

The problem addressed by error simulation is as follows. Given a design, a set of (synthetic) design errors, and a sequence of test vectors, determine which errors are detected by the test sequence. Fault simulation addresses a similar problem in physical fault testing, but differs in the error/fault models. Whereas physical fault testing is concerned with SSL faults, bridging faults, open faults, our design verification methodology needs to consider other errors, such as the conditional errors introduced in [VC98]. In this section we address error simulation with conditional errors. First, we motivate error simulation.

Augmenting targeted test generation with error simulation can reduce overall run times. Test generators typically target one error at a time. A targeted test may detect errors other than just the targeted error. These errors can be identified by an error simulator so that they do not need to be considered by the test generator any more.

A stand-alone use of error simulation is the computation of design error coverage of a given test suite. This is useful in regression testing, where one might be interested in selecting a subset of a given set of test sequences that provides coverage of design errors similar to those of the complete test set. Error simulation can also reveal areas of the design that are not sufficiently tested by a given test suite, and hence spur further targeted test generation.

Error simulation needs to be efficient. Not only the length of test suites, which is extremely large for pseudo-random tests, but also the nature of the error models, and the number of error instances to be considered affect the size of the task. It is clear that better methods are required than simple serial error simulation, which simulates the erroneous designs for the complete test suite one by one.

Conditional error models. A conditional error (C, E) [VC98] consists of a condition C and a basic error E; its interpretation is that E is only active when C is satisfied. In general, C is a predicate over the signals in the circuit during some time period. To limit the number of error instances, we restrict C to a conjunction of terms ($y_i = w_i$), where y_i is a signal in the circuit that is not in the transitive combinational fanout of the basic error[1], and w_i is a constant of the same signal-width as y_i and whose value is either all-0's or all-1's. The number of terms (condition variables) appearing in C is said to be the order of (C, E). Specifically, we consider the following conditional error (CE) types:

- conditional single-stuck line error of order n (CSSLn)
- conditional bus order error of order n (CBOEn)
- conditional bus source error of order n (CBSEn)

Figure 1 gives an example of a CSSL1 error, ($x = 1$, y / 0). If the condition does not hold, $x \neq 1$, the erroneous circuit operates as the error-free. If the condition holds, $x = 1$, line y is stuck at 0.

The number of instances defined by a conditional error model (C,E) is given by the product of the number of basic errors and the number of conditions:

#CSSLn = $O(2^{n+1}N^{n+1})$
#CBOEn = $O(2^{n}N^{n+1})$
#CBSEn = $O(2^{n}N^{n+2})$

1. The requirement that condition signals are not to be part of the transitive combinational fanout of the basic error, eliminates problems of combinational feedback, and thus ensure that all conditional errors are well defined. This requirement also facilitates efficient error simulation, as we will see in Section 3.

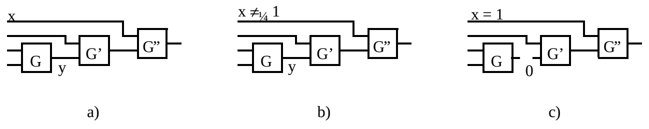

Figure 1. CSSL1 error ($x = 1, y$ / 0): a) error-free design; b) erroneous design with $x \neq 1$; c) erroneous design with $x = 1$.

For $n = 0$, a conditional error (C,E) reduces to the basic error E from which it is derived. Higher-order conditional errors enable the generation of more specific tests, but lead to a greater test generation cost due to the larger number of error instances. For example, the CSSL1 model defines a number of instances quadratic in the size of the circuit. Although the total set of all signals we consider for each term in the condition can possibly be reduced, CSSLn errors where $n > 1$ are probably not practical. Experimental work presented in [VC98, VC99] shows that test sets that are complete for CSSL1 errors provide a higher coverage for actual errors than test sets that are complete for CSSL0 errors only.

We review relevant previous work in Section 2. Our error simulation algorithm is presented in Section 3. We present experimental results in Section 4, and give some concluding remarks in Section 5.

2. Related Work

Representative approaches to fault simulation for sequential circuits [Abra90, Nier91a] are parallel, concurrent, deductive, and differential fault simulation. *Parallel* fault simulation takes advantage of the word-level parallelism of the computer used. On a 32-bit computer, 32 faulty machines can be simulated in parallel. This method lacks the ability to drop errors. The other methods are motivated by the observation that as long as a fault is not detected, the good and faulty circuit differ in only a fraction of the number of signals present. For this purpose, such methods process the complete set of faulty machines one vector at a time. Both *concurrent* and *deductive* fault simulation compute the node values of a faulty machine for the current vector, based on the good circuit's node values for the current vector, and the faulty machine's node values for the previous vector. A drawback of both methods is high memory requirement. *Differential* fault simulation, a variant of concurrent fault simulation, addresses the memory problem, but suffers from the inability to drop detected faults.

Niermann, Cheng and Patel [Nier90, Nier91a] described a fault simulator, called PROOFS, that combines ideas of concurrent, differential and parallel fault simulation. As our error simulation method for conditional errors derives from PROOFS, we briefly describe its main features, referring to Figure 2.

Given is a gate-level sequential circuit, a fault list, and a test vector sequence, PROOFS maintains two sets of signal values: one for the good, and one for a faulty machine. For each undetected fault, PROOFS also stores the difference in present state between the good machine and the corresponding faulty machine.

The outermost loop of PROOFS processes one test vector at a time. First, the good machine is simulated for the current vector. Next, faults that are active for the current test vector are identified. A fault is considered *active* if one or both of the following two conditions holds: 1) the present state of the faulty machine is different from that of the good machine; 2) the fault is excited by the current vector, and the faulty line is sensitized through the first two levels of logic. Checking condition 1 is straightforward since we have saved the faulty circuit's state while processing the previous vector. If condition 1 does not hold, that is, if the faulty circuit's present state is identical to that of the good circuit, checking condition 2 is inexpensive too, as it is very localized and requires only the good circuit's values.

```
PROOFS(circuit, faultList, testVectorSequence)

1.      while (vectors left) {
1.1         read next vector
1.2         simulate good circuit
1.3         determine which faults are active
1.4         for each active fault {
1.4.1           inject fault
1.4.2           add faulty node events
1.4.3           simulate faulty circuit
1.4.4           drop detected faults
1.4.5           store faulty next state
1.4.6           remove fault
1.5         }
2.      }
```

Figure 2. PROOFS' error simulation algorithm

Faults that are not active for the current vector have the property that they are not detected by the current vector *and* the next states of the corresponding faulty machines are identical to the next state of the good machine. Consequently, there is no need to simulate these faulty machines for the current vector.

Each active fault is processed as follows: First, the fault is injected into faulty circuit. The event list is initialized to reflect the fault injection and the present state lines whose values differ in the good and the faulty machine. The event-driven simulation of the faulty machine in PROOFS typically has a very low event activity, as in concurrent fault simulation. If the fault is detected by the current vector, it is dropped. Otherwise, the difference between the next state of the faulty machine and that of the good machine is saved.

The basic algorithm, as discussed above, can be augmented to take advantage of the word-level parallelism available on the computer executing the fault simulator. On a 32-bit machine, up to 32 iterations the simulation step 1.4.3 of loop 1.4 can be executed in parallel. This is done by assigning the values of different faulty machines to different bit positions within a word. The other steps of loop 1.4 still have to be executed serially. A more detailed description of one implementation is given in [Nier90, Nier91a].

3. CESIM

It is straightforward to modify PROOFS to handle conditional errors, such as CSSL1. For a given circuit and a given test sequence, the average run time per error for CSSL1 error simulation is very close to that for SSL error simulation. As the number of CSSL1 errors is quadratic in the size of the circuit, the cost of error simulation for CSSL1 may be prohibitively large. To address this, we develop an error simulation algorithm for conditional errors, called CESIM, that exploits the close relationship among CSSL1 errors derived from the same CSSL0 error. Its key features are processing of sets of conditional errors, and the injection of basic (instead of conditional) errors. We will demonstrate that this

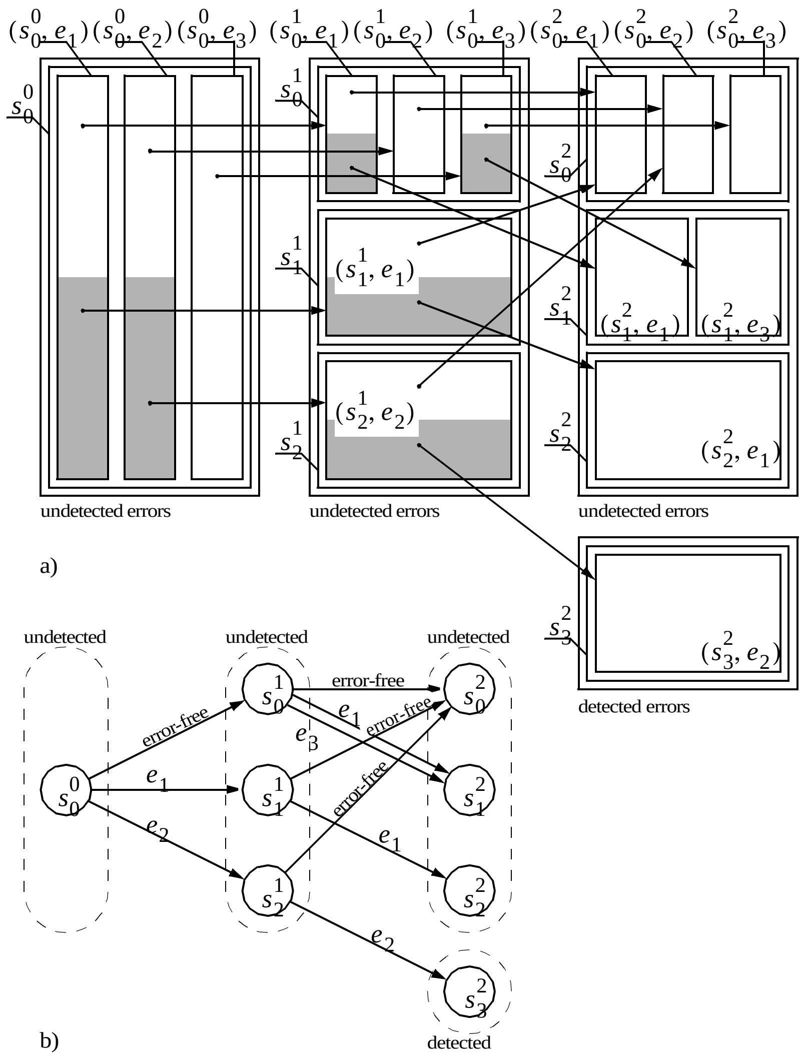

Figure 3. Example execution of CESIM for a 3-vector test sequence: a) PS- and PSBE-partitions of errors, b) corresponding state transition

leads to improved performance over the naive extension of PROOFS.

First, we define two equivalence relations on conditional errors. Two conditional errors are *PS-equivalent* with respect to the current vector iff the present states of the corresponding erroneous machines are identical. Two conditional errors are *PSBE-equivalent* with respect to the current vector, iff they are derived from the same basic error and the present states of the corresponding erroneous machines are identical. The two equivalence relations define a hierarchical partition on the set of conditional errors; PSBE-equivalence refines the partition defined by PS-equivalence. CESIM maintains the set of undetected errors in partitioned form.

We redefine the activity criterion of PROOFS as follows: A conditional error is *active* for the current vector iff 1) its condition holds in the erroneous circuit, and 2) the corresponding basic error is excited in the erroneous machine for the current vector, and 3) that basic error is sensitized through the first two levels of logic.

The inner loop 1.4 of PROOFS (Figure 2) that iterates over individual active faults is replaced in CESIM, outlined in Figure 4, by one that iterates over sets of PS-equivalent conditional errors:

Given is a set S_1 of undetected PS-equivalent conditional errors, we process this set of errors for the current vector as follows. First, we simulate the erroneous machine[1] with no errors

1. Note that CESIM uses a single copy of the circuit structure but associates two values with each signal, one corresponding to the error-free machine, the other to an erroneous machine. Hence by simulating the *erroneous machine* we mean simulating the circuit using the set of *erroneous values*.

```
CESIM( circuit, errorList, testVectorSequence)

1.          U = errorlist  /* hierarchically partitioned set of undetected errors */
2.          while (vectors left) {
2.1             read next vector
2.2             simulate good circuit
2.3             for each set $S_1$ of PS-equivalent condit. errs. in U {
2.3.1               add the erroneous present-state events
2.3.2               simulate erroneous machine (no errors injected)
2.3.3               partition $S_1$ into an active/inactive subsets, A/D.
2.3.4               if error effect is exposed {
2.3.4.1                 drop all errors in D
2.3.5               }
2.3.6               else {
2.3.6.1                 save next state for D
2.3.6.2                 insert D in nextU
2.3.7               }
2.3.8               for each set $S_2$ of PSBE-equiv. condit. errs. in A {
2.3.8.1                 inject the corresponding basic error
2.3.8.2                 add the erroneous node events
2.3.8.3                 simulate the erroneous circuit
2.3.8.4                 if basic error is detected {
2.3.8.4.1                   drop all errors in $S_2$
2.3.8.5                 }
2.3.8.6                 else {
2.3.8.6.1                   save the erroneous next state for $S_2$
2.3.8.6.2                   insert $S_2$ in nextU
2.3.8.7                 }
2.3.8.8                 remove the error
2.3.9               }
2.4             }
2.5             U = nextU
3.          }
```

Figure 4. CESIM error simulation algorithm for conditional errors

injected, starting from the present state associated with S_1 for the current vector (steps 2.3.1 and 2.3.2 of Figure 4).

For each conditional error in S_1, we check if it is active. Activation is determined by three conditions (see above). Conditions 2 and 3 only depend on the basic error, and hence are identical for all E-equivalent errors. We therefore check conditions 2 and 3 first (one check for each class of PSBE-equivalent errors). Only if both conditions hold do we have to check condition 1 (one check per individual conditional error). Note that lines appearing in the condition of a conditional error are not part of the transitive combinational fanout of the basic error. Therefore, the activation conditions can be evaluated using the values computed in step 2.3.2. This partitions S_1 into a subset *A* of the active conditional errors, and a subset *D* of dormant (not active) errors (step 2.3.3).

If any outputs computed in 2.3.2 differ from those of the good circuit (step 2.2) all errors in *D* are detected and can be dropped. Otherwise, we record the erroneous next state corresponding to *D*, and insert *D* into the *nextU*, the set of undetected errors for the next vector.

For each set S_2 of PSBE-equivalent errors in *A*, we inject the *basic* error corresponding to S_2, apply the erroneous present state corresponding to S_2, and simulate the erroneous circuit. If any outputs differ from those in the good circuit, all errors in S_2 are dropped. Otherwise, we record the erroneous next state for S_2, and insert S_2 into *nextU*.

Example. Figure 3 illustrates CESIM. Consider sets of conditional errors derived from three basic errors e_1, e_2, and e_3. Initially, the corresponding erroneous machines are all in the same present state, namely the unknown state s_0^0. The initial PS-partition has a single class, which is further partitioned with respect to PSBE-equivalence. First, the error-free machine is simulated for the first vector; the next state is s_0^1. This allows us to separate those conditional errors that are active (shaded in the figure) for the first vector from those that are not. For the dormant errors no further work is required: none of them is detected, and the next state of the corresponding erroneous machines is s_0^1. For each PSBE class that contains active conditional errors, the corresponding *basic* error is injected and the erroneous circuit is simulated for the current vector. In the example, none of these errors is detected, and the next states s_1^1 and s_2^1 are distinct. This process is repeated for the next vector. In the example, the active errors in PSBE class (s_2^1, e_2) are detected by the second vector; all other errors remain undetected. Note that there is a one-to-one correspondence between a single transition in the state transition diagram in Figure 3 and a circuit simulation step in the algorithm (steps 2.2, 2.3.2, or 2.3.8.3 in Figure 4).

Analysis. CESIM minimizes the overall computational cost by exploiting PS- and PSBE-equivalence of conditional errors. We now analyze the algorithm's complexity. The two major components of the cost of one iteration of the top-level loop (step 2) are the simulation cost of steps 2.2, 2.3.2, and 2.3.8.3, and the partition cost of step 2.3.3. The partition cost is proportional to the number of conditional errors for which we have to check activation condition 1, which is typically a small fraction of the total number of conditional errors. The event-driven simulator is called as many times as there are PSBE partition classes on all sets *A*; this is a fraction of the number of PSBE partition classes of *U*. In summary, the cost of one iteration has one component with complexity sublinear in the size of the error list (partition cost), and a second component proportional to the size of the circuit and the product of the number of basic errors and the number of distinct states (simulation cost). In our experiments, we observed that 90% of the execution time is due to partitioning, while only 10% is due to simulation. The algorithm requires maintaining both partitions (PS and PSBE) on the set of undetected errors. All partitions are implemented using hash tables, which allow for constant time insertions of error sets.

Initially, all errors are undetected and the corresponding erroneous machines all start from an unknown present state. Hence all errors are PS-equivalent initially, and all errors derived from the same basic error are PSBE-equivalent. In the partition step (2.3.3), the number of error sets (PSBE equivalence classes) may increase. The worst case occurs when 1) neither *A* nor *D* is empty, 2) neither of them is detected, and 3) the next states generated in steps 2.3.8.3 and 2.3.2 are all distinct. For this case, the number of error sets can double in a single iteration of step 2, leading to an exponential growth in the number of vectors. However, the total number of PSBE-equivalence classes can never exceed the total number of individual conditional errors we started with. Our experimental results (see below) show that, in practice, the number of error sets remains fairly constant.

Optimizations. As in PROOFS we take advantage of the word-level parallelism of the host computer; hence multiple iterations of

Table 1. Test generation and fault simulation of ISCAS’89 circuits using HITEC

Circuit	Detected faults	Redundant faults	Aborted faults	Vectors	Efficiency	Coverage	HITEC CPU[s]	PROOFS CPU[s]
s27	32	0	0	21	1.0000	1.0000	0.12	0.03
s208.1	18	146	53	12	0.7558	0.0829	164.07	0.07
s298	265	26	17	220	0.9448	0.8604	48.18	0.25
s344	321	9	12	89	0.9649	0.9386	32.38	0.15
s349	329	11	10	106	0.9714	0.9400	31.57	0.17
s382	281	2	116	881	0.7093	0.7043	267.70	1.45
s386	314	70	0	273	1.0000	0.8177	5.52	0.22
s400	331	9	86	1,228	0.7981	0.7770	215.72	1.20
s420.1	28	151	276	20	0.3934	0.0615	635.98	0.15
s444	254	16	204	316	0.5696	0.5359	777.90	0.88
s526	51	14	490	34	0.1171	0.0919	1,167.25	0.15
s526n	55	13	485	37	0.1230	0.0995	1,162.57	0.18
s641	404	63	0	203	1.0000	0.8651	3.97	0.35
s713	476	105	0	196	1.0000	0.8193	5.55	0.40
s820	811	29	10	940	0.9882	0.9541	96.17	1.52
s832	813	46	11	962	0.9874	0.9345	99.63	1.78
s838.1	48	515	368	28	0.6047	0.0516	849.52	0.40
s953	89	990	0	14	1.0000	0.0825	38.68	0.28
s1196	1,239	3	0	439	1.0000	0.9976	2.95	0.80
s1238	1,283	72	0	472	1.0000	0.9469	4.77	0.97
s1423	578	11	926	89	0.3888	0.3815	2,100.88	0.85
s1488	1,368	20	98	778	0.9341	0.9206	325.48	2.30
s1494	1,447	32	27	991	0.9821	0.9608	118.78	2.93
s5378	3,146	159	1,298	894	0.7180	0.6835	2,941.93	14.68
s9234	18	3,916	0	6	1.0000	0.0046	1.38	1.05
s9234.1	366	181	3,391	38	0.1389	0.0929	7,562.58	3.62
s13207	557	8,218	672	76	0.9289	0.0590	2,266.77	16.08
s13207.1	858	7,583	1,148	106	0.8803	0.0895	3,475.12	28.25
s15850	85	11,407	21	8	0.9982	0.0074	60.15	2.97
s15850.1	4,374	1,229	5,920	2,493	0.4862	0.3796	16,302.12	383.82
s35932	34,868	3,984	242	300	0.9938	0.8919	2,350.30	45.47
s38417	1,088	356	29,016	51	0.0474	0.0357	96,335.17	46.05
s38584	7,798	6,759	21,744	1,593	0.4010	0.2148	54,293.70	1,293.65
s38584.1	20,589	1,948	13,766	4,383	0.6208	0.5671	38,090.13	1,535.40

2.3.2 and of 2.3.8.3 are executed in parallel. To further reduce execution time, static dominators [Nier91a] could be used to identify redundant errors during a preprocessing step.

4. Experiments

We used the ISCAS’89 benchmarks to evaluate the performance of CESIM. We generated test sequences for SSL faults using HITEC [Nier90, Nier91a]. The parameters that determine when to abort a fault were set as follows: the backtrack limit was set to 10,000; the state backtrack limit was set to 10,000; the time limit per fault was set to 2 seconds. We separately fault simulated the obtained test sequences using PROOFS [Nier90, Nier91a]. Test generation and fault simulation were performed on a Fujitsu HAL-Station/300; the results are summarized in Table 1. We did not try to improve fault coverage further by increasing the abort limits.

We error-simulated the same test sequences using CESIM for CSSL0 and CSSL1 errors. The error list for CSSL0 errors is identical to the collapsed SSL fault list from before. The CSSL1 error list was constructed as follows. For each CSSL0 error, we considered a maximum of 500 lines to condition the error. The smaller circuits have less than 500 lines, so every line in the circuit is considered as the condition line. This leads to a maximum of 1000 CSSL1 errors per CSSL0 error. However, some CSSL1 errors are rejected because their condition is part of the transitive fanout of the error site. The results of error simulation using CESIM are summarized in Table 2. In the analysis that follows, we exclude benchmarks for which the test sequence achieves an SSL coverage less than 0.1.

Comparing the results of CSSL0 error simulation using CESIM in Table 2 with the results of SSL fault simulation using PROOFS in Table 1, we conclude the following. Fault/error coverages for

Table 2. Error simulation of ISCAS'89 circuits using CESIM

Circuit	Vectors	CSSL0 No.	CSSL0 Coverage	CSSL0 CPU [s]	CSSL1 No.	CSSL1 Coverage	CSSL1 CPU [s]	$\frac{\text{cover.}_1}{\text{cover.}_0}$	$\frac{\left(\frac{\text{CPU}}{\text{no.}}\right)_0}{\left(\frac{\text{CPU}}{\text{no.}}\right)_1}$
s27	21	32	1.0000	0.01	664	0.7289	0.10	0.7289	2.08
s208.1	12	217	0.0829	0.11	44,492	0.0228	0.67	0.2750	33.66
s298	220	308	0.8604	0.63	75,468	0.7103	7.80	0.8255	19.79
s344	89	342	0.9386	0.39	114,294	0.7598	5.09	0.8095	25.61
s349	106	350	0.9400	0.32	117,484	0.7796	5.20	0.8294	20.66
s382	881	399	0.7043	5.61	130,656	0.5830	60.35	0.8278	30.44
s386	273	384	0.8177	1.30	93,060	0.6040	12.60	0.7387	25.00
s400	1,228	426	0.7770	7.77	144,628	0.6275	74.47	0.8076	35.42
s420.1	20	455	0.0615	0.28	199,616	0.0180	3.60	0.2927	34.12
s444	316	474	0.5359	3.44	175,616	0.4185	42.15	0.7809	30.24
s526	34	555	0.0919	0.65	218,816	0.0656	7.15	0.7138	35.84
s526n	37	553	0.0995	0.69	219,166	0.0704	7.54	0.7075	36.27
s641	203	467	0.8651	1.31	349,832	0.7219	24.87	0.8345	39.46
s713	196	581	0.8193	1.39	452,256	0.6742	32.78	0.8229	33.01
s820	940	850	0.9541	6.30	442,758	0.6456	160.27	0.6767	20.48
s832	962	870	0.9345	6.72	445,274	0.6355	165.86	0.6800	20.74
s838.1	28	931	0.0516	1.02	821,774	0.0158	17.72	0.3062	50.81
s953	14	1,079	0.0825	0.63	879,540	0.0321	12.95	0.3891	39.66
s1196	439	1,242	0.9976	5.75	1,058,844	0.7483	151.78	0.7501	32.30
s1238	472	1,355	0.9469	7.49	1,140,402	0.6969	186.76	0.7360	33.75
s1423	89	1,515	0.3815	5.44	1,287,036	0.2695	72.08	0.7064	64.12
s1488	778	1,486	0.9206	10.18	1,142,374	0.6615	272.48	0.7186	28.72
s1494	991	1,506	0.9608	10.74	1,151,208	0.6993	327.06	0.7278	25.10
s5378	894	4,603	0.6835	94.80	4,517,276	0.5418	1,753.33	0.7927	53.06
s9234	6	3,934	0.0046	2.11	3,850,188	0.0004	43.25	0.0870	47.75
s9234.1	38	3,938	0.0929	21.70	3,854,738	0.0490	147.79	0.5274	143.73
s13207	76	9,447	0.0590	77.66	9,325,456	0.0132	638.82	0.2237	120.00
s13207.1	106	9,589	0.0895	115.96	9,453,928	0.0363	1,167.80	0.4056	97.90
s15850	8	11,513	0.0074	11.92	11,197,872	0.0021	151.72	0.2838	76.42
s15850.1	2,493	11,523	0.3796	2735.14	11,186,886	0.2574	33,994.54	0.6781	78.11
s35932	300	39,094	0.8919	170.70	38,708,548	0.8400	4,541.98	0.9418	37.21
s38417	51	30,460	0.0357	249.88	30,222,794	0.0109	1,981.77	0.3053	125.11
s38584	1,593	36,301	0.2148	6233.05	36,124,976	0.1244	122,142.85	0.5791	50.78
s38584.1	4,383	36,303	0.5671	8,656.32	36,124,758	0.4423	199,346.07	0.7799	43.21

corresponding circuits are identical, as expected. We can deduce that CESIM is on average 4.5 times slower than PROOFS. The actual slowdown varies between 0.3 and 7.7; for the largest benchmark it is 5.6. CESIM does not include any optimizations that are specific to CSSL0 errors.

The last two columns in Table 2 compare error simulation for CSSL1 errors to error simulation for CSSL0 errors. We observe that the ratio of coverage of CSSL1 errors to coverage of CSSL0 varies between 0.58 and 0.94; its average is 0.76. We also computed the ratio of CPU time per error for CSSL1 errors to CPU time per error for CSSL0 errors. This ratio attempts to measure the speedup of CESIM for CSSL1 errors compared to a naive approach. We observe an average speedup varies between 2 and 78, its average is 34, and for the largest benchmark a speedup of 43 was obtained.

The efficiency of CESIM can best be seen by plotting the CPU time per test vector versus the total number of errors simulated for each benchmark, as in Figure 5. There are two sets of data: the first concerns CSSL0 error simulation, the other CSSL1 error simulation. The plot also shows a least square fit (linear regression) for each data set. The execution time of CSSL0 error simulation is dominated by event-driven simulation of faulty circuits. However, when simulating the CSSL1 errors, checking whether the condition of each CSSL1 error holds dominates the execution time. Least-square analysis shows that the CPU time per test vector is proportional to the number of CSSL0 errors to the power 1.33. This superlinear behavior reflects the fact that those data with a larger number of CSSL0 errors correspond to larger circuits, and hence the execution time of each event-driven simulation increases. For the CSSL1 execution time we find that the CPU time per test vector is proportional to the number of CSSL1 errors to the power 1.13. This near-linear behavior is because checking if CSSL1 errors are active, which is independent of the size of the circuit, dominates the execution time. Further analysis is provided in [VC99].

Table 5 also allows us to compare CESIM with the straightforward extension of PROOFS for CSSL1 error simulation, which we will refer to here as CPROOFS. CPROOFS treats CSSL1 errors the same way CESIM treats CSSL0 errors. The increase in execution time of CPROOFS for CSSL1 errors compared to the execution time of PROOFS for CSSL0 errors is therefore proportional to the ratio of the number of CSSL1 errors to the number of CSSL0 errors. The figure shows the execution time of CPROOFS for only one benchmark (s1423); the speedup for that circuit is 64. We can see that the speedup is roughly equal to the vertical distance between the linear regression lines for CSSL0 and CSSL1 datasets. We conclude that CESIM outperforms the CPROOFS by a wide margin.

We further analyze the behavior of CESIM for a representative circuit, s1238. This circuit has 14 inputs, 14 outputs, 18 D-type flip-flops, 80 inverters and 428 gates. Figure 6 shows the error coverage as a function of the number of test vectors applied. The ratio of coverage of CSSL1 errors to coverage of CSSL0 errors varies between 0.49 and 0.72.

Figure 7 shows the number of distinct states as a function of the number of test vectors applied. For CSSL0 error simulation, the number of states rapidly drops; after vector 300 there are at most five distinct states among the present states of the remaining undetected erroneous machines. For CSSL1 error simulation, we observe that the number of states hovers around 20 but never becomes larger than 35 (about twice the number of flip-flops in the circuit).

Figure 8 details the number of error sets occurring during the execution of CESIM. We show both the total number of error sets, and the number of error sets in use. Both are normalized with respect to the total number of errors. The number of error sets in use is the number of PSBE-equivalence classes of the set of undetected errors *U* in loop 2.3 of Figure 4. The total number of error sets is the number of error sets in use plus the number of errors sets detected by previous vectors (those error sets are dropped in steps 2.3.4.1 and 2.3.8.4.1 of Figure 4). For CSSL0 simulation, the total number of error sets remains constant at the number of errors, whereas the number of error sets in use drops as coverage increases. For CSSL1 simulation we observe that the total number of error sets increases steadily, as coverage increases. However, the number of error sets in use remains fairly constant and hovers around the total number of basic errors, which is about 1000 times smaller than the total number of errors.

5. Conclusions

Higher-order conditional error models, such as the CSSL1 model, define a number of error instances superlinear in the size of the circuit. Although state-of-the-art fault simulation methods, such as PROOFS, can easily be extended for conditional error models, the large number of error instances may result in prohibitively large computational cost for error simulation with these error models. To address this problem, we have developed a new error simulation algorithm CESIM that exploits the nature of conditional errors.

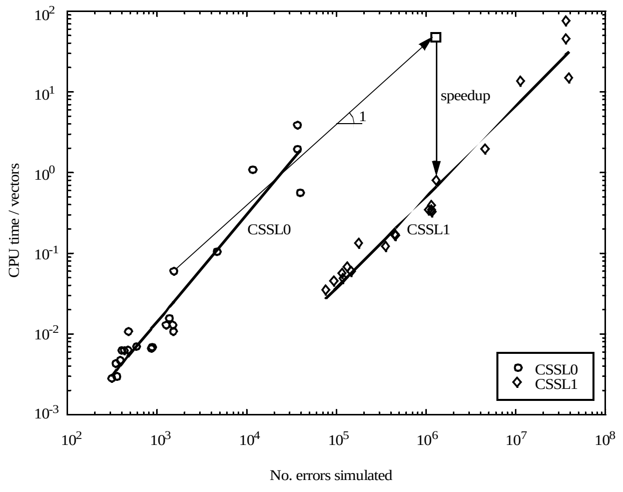

Figure 5. Run-time analysis of CESIM on the ISCAS’89 benchmarks

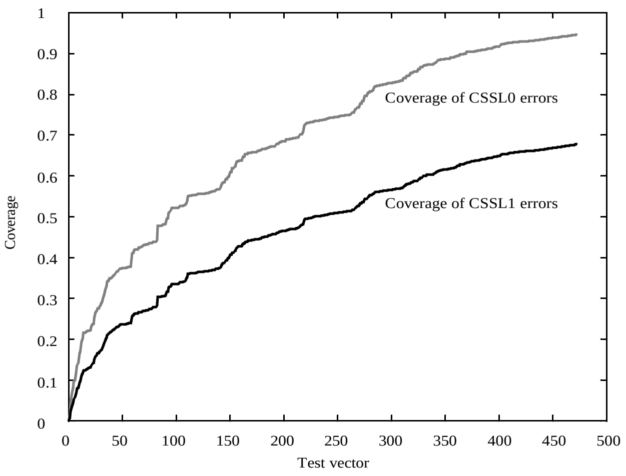

Figure 6. Coverage of CSSL0 and CSSL1 errors on s1238 by a CSSL0 test set generated by HITEC

We have conducted experiments using the ISCAS’89 benchmarks that demonstrate that error simulation with CSSL1 errors is practical for moderately sized circuits. On average CESIM gained a 34x improvement in execution time compared to CPROOFS, which is a straightforward extension of PROOFS for CSSL errors.

However, as the run time of our algorithm is linear in the number of errors, for very large circuits the quadratic number of CSSL1 errors becomes prohibitive. For those circuits restrictions on the general CSSL1 model may be appropriate. For example,

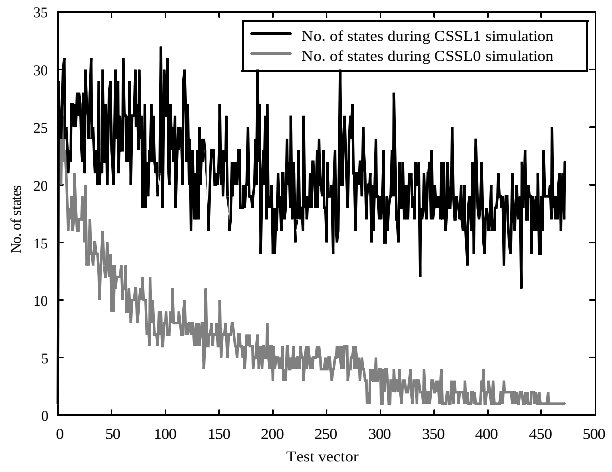

Figure 7. Error simulation on s1238 with CSSL0 and CSSL1: number of distinct states

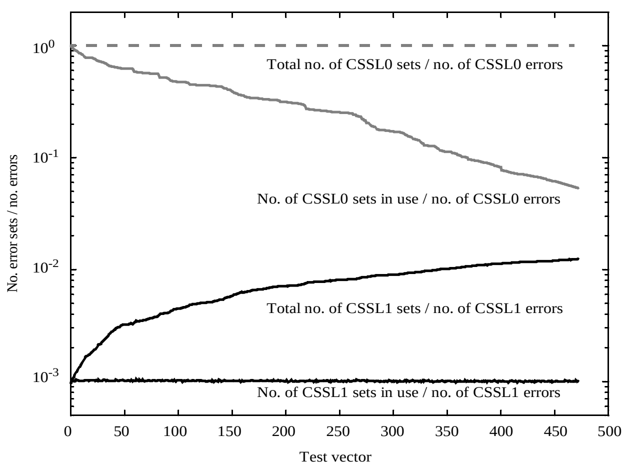

Figure 8. Number of error sets during error simulation on s1238 with CSSL0 and CSSL1 errors

when deriving CSSL1 errors from a given CSSL0 error, one could restrict the condition signals to those signals appearing in the same hierarchical module as the CSSL0 error.

Acknowledgment

This research is supported by DARPA under Contract No. DABT63-96-C-0074.

References

[AA95] H. Al-Asaad and J. P. Hayes. Design verification via simulation and automatic test pattern generation. In *Proc. Int. Conf. Computer-Aided Design*, pages 174–180, 1995.

[Abad88] M. S. Abadir, J. Ferguson, and T. E. Kirkland. Logic design verification via test generation. *IEEE Trans. Computer-Aided Design*, 7(1):138–148, January 1988.

[Abra90] M. Abramovici. *Digital systems testing and testable design*. Computer Science Press, New York, 1990.

[Ahar91] A. Aharon, A. Bar-David, B. Dorfman, E. Gofman, M. Leibowitz, and V. Schwartzburd. Verification of the IBM RISC System/6000 by dynamic biased pseudo-random test program generator. *IBM Systems Journal*, pages 527–538, 1991.

[Beiz90] B. Beizer. *Software testing techniques*. Van Nostrand Reinhold, New York, 2nd edition, 1990.

[Chan95] A.K. Chandra, V.S. Iyengar, D. Jameson, R. Jawalekar, I. Nair, B.K. Rosen, M.P. Mullen, J. Yoon, R. Armoni, D. Geist, and Y. Wolfsthal. AVPGEN - a test generator for architecture verification. *IEEE Trans. on VLSI*, pages 188–200, 1995.

[Fall98] F. Fallah, S. Devadas, and K. Keutzer. OCCOM: Efficient computation of observability-based code coverage metric for functional simulation. In *Proc. Design Automation Conf.*, pages 152–157, 1998.

[Ho96] R. C. Ho and M. A. Horowitz. Validation coverage analysis for complex digital designs. In *Proc. Int. Conf. Computer-Aided Design*, pages 146–151, 1996.

[Nier90] T. Niermann, W. T. Cheng, and J. H. Patel. PROOFS: A fast memorry efficient fault simulator for sequential circuits. In *Proc. Design Automation Conf.*, pages 190–196, 1990.

[Nier91a] T. Niermann. *Techniques for sequential circuit automatic test generation*. PhD thesis, University of Illinois, 1991.

[Nier91b] T. Niermann and J. H. Patel. HITEC: A test generation packaged for sequential circuits. In *Proc. European Design Automation Conf.*, pages 214–218, 1991.

[Tayl98] S. Taylor, M. Quinn, D. Brown, N. Dohm, S. Hildebrandt, J. Huggins, and C. Ramey. Functional verification of a multiple-issue, out-of-order, superscalar Alpha processor - the DEC Alpha 21264 microprocessor. In *Proc. Design Automation Conf.*, pages 638–643, 1998.

[VC98] D. Van Campenhout, H. Al-Asaad, J. P. Hayes, T. Mudge, and R. B. Brown. High-level design verification of microprocessors via error modeling. *ACM Trans. Design Automation of Electronic Systems*, 3(4):581–599, October 1998.

[VC99] D. Van Campenhout. Functional Design Verification for Microprocessors by Error Modeling. PhD thesis, University of Michigan, 1999.