AGFC: AN APPROXIMATE SIMULATION-BASED GLOBAL FAULT COLLAPSING TOOL FOR COMBINATIONAL CIRCUITS

Hussain Al-Asaad
Department of Electrical & Computer Engineering
University of California
One Shields Avenue, Davis, CA 95616-5294
E-mail: halasaad@ece.ucdavis.edu

ABSTRACT

Exact global fault collapsing can be easily applied locally at the logic gates, however, it is often ignored for large circuits due to its high demand of execution time and/or memory. In this paper, we present AGFC, an approximate global fault collapsing tool for combinational circuits. Experimental results show that (i) AGFC reduces the number of faults drastically with feasible resources and (ii) AGFC produces significantly better results than existing approaches.

KEY WORDS

Global fault collapsing, fault simulation, physical fault testing, combinational circuits.

1 Introduction

To test a digital circuit, an automatic test pattern generation (ATPG) tool generates a test set that targets possible physical faults. As the complexity of the digital circuit increases, the possible number of physical faults increases that consequently leads to a significant slow down of the test generation process using the ATPG tool. One approach for considerably reducing the length of the testing process as well as producing compact test sets is fault collapsing. Fault collapsing [1] is the process of reducing the number of faults by using redundance, equivalence, and dominance relationships among faults. Exact fault collapsing can be easily applied locally at the logic gates; however, it is often not feasible to apply it globally for large circuits.

Several researchers have worked on fault collapsing. An algorithm was presented in [2] that collapse all the structurally equivalent faults in a circuit, plus many of the functionally equivalent faults. Application of the algorithm to the ISCAS-85 benchmark circuits [3] establishes that identification of functionally equivalent faults is feasible, and in some cases, they are a large fraction of the faults in a circuit. However, the overall produced collapsed fault list is still large in comparison to the global collapsed fault list.

A graph-theoretic hierarchical fault collapsing method was presented in [4][5] that can collapse faults in any large cell-based circuit. Since functional analysis (equivalence and dominance) is computationally expensive, it is only applied to standard cells. As an example, consider the size of the collapsed fault list for an exclusive-OR cell. Using the method of [5], the collapsed fault list reduces to just four faults when functional fault collapsing is considered. With the traditional method of structural collapsing this set contains 13 faults. When the exclusive-OR cell is used to build an 8-bit adder circuit, the size of the collapsed fault list produced by [5] reduces to 112 faults from a total of 466 faults. Traditional structural fault collapsing would have given a set of 226 faults. Although a significant reduction is achieved here, the method assumes a hierarchical design with a good use of standard cells. Moreover, the size of the produced collapsed fault list is still large in comparison to the exact global collapsed fault list.

Recently, a new diagnostic and detection fault collapsing method was introduced for multiple-output circuits [6]. Using this method, a significant reduction in the fault list was achieved, however, the method again assumes a hierarchical design (such as adders and ALUs) with a good use of small standard cells.

Efficient techniques for identifying functionally equivalent faults in combinational circuits were presented in [7]. The techniques are based on implication of faulty values, and evaluation of faulty functions in cones of dominator gates of fault pairs. Experimental results show that most of the equivalent fault pairs are identified. However, this work does not aim at producing a small collapsed fault list.

In our previous work [8], we have presented a preliminary method that produces a compact fault list—an approximation of the global collapsed fault list. Our approximate global fault collapsing technique is based on the simulation of random vectors. Experimental results show that our method produced significant reduction in the size of the collapsed fault list. However, our preliminary approximate global fault collapsing tool (a set of scripts manipulating several academic CAD tools) turned out to be resource intensive and memory hungry process. Even with only 1,000 test vectors, many of the smaller benchmark circuits

required several hours to simulate.

In addition to the above, we have developed an exact global fault collapsing tool (EGFC) for combinational circuits [9]. EGFC uses binary decision diagrams to compute the tests for faults and consequently achieve the global fault collapsing. EGFC reduces the number of faults drastically in small circuits (such as a 4-bit ALU) but it cannot be scaled to handle large circuits.

In this paper, we describe AGFC, an efficient tool for approximate global fault collapsing. We review fault collapsing in Section 2 and present our efficient technique for approximate fault collapsing for medium size circuits (comparable to the ISACS-85 benchmarks [3]) and the AGFC tool in Section 3. We present some experimental results in Section 4 and finish with a discussion about extending our approximate global fault collapsing method to large combinational circuits in Section 5.

2 Fault Collapsing

In physical fault testing, physical defects are abstracted into a logical fault model. The most widely-used logical fault model is the Single Stuck-Line (SSL) model [1]. Under this model, every single signal line can become permanently fixed (stuck) at a logical 1 or 0 value. The model is simple and technology-independent. It represents a large number of different physical faults, and tests derived for SSL faults detect many design errors/faults [10]. In this paper, we only consider SSL faults; however, our method is applicable to several other fault models.

Fault collapsing first removes redundant faults from the fault list. A fault is redundant if there is no test that can detect it. In other words, a fault is redundant if the faulty function is the same as the correct function. Fault collapsing then reduces the number of faults using two relationships among faults: fault equivalence and fault dominance. Two faults are considered equivalent if the faulty functions produced by the two faults are equal. Alternatively, the two faults are equivalent if they can be detected by the same tests. In this case, there is no way to distinguish between the two faults. For example, the SSL fault *a* stuck-at 0 represented by *a*/0 in Figure 1 is equivalent to the fault *z*/0. If two faults are equivalent then one of the faults can be dropped from the fault list since the detection of the other fault guarantees the detection of the dropped fault.

Inputs		Output z	Faulty functions					
a	*b*		*a*/0	*a*/1	*b*/0	*b*/1	*z*/0	*z*/1
0	0	0	0	0	0	0	0	1
0	1	0	0	1	0	0	0	1
1	0	0	0	0	0	1	0	1
1	1	1	0	1	0	1	0	1

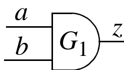


Figure 1 Fault collapsing for a 2-input AND gate.

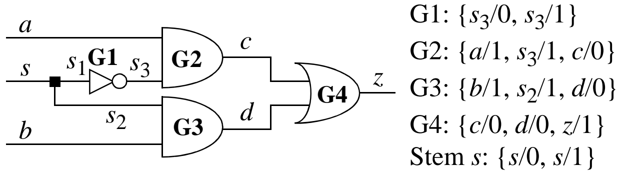
A fault *f* is considered to dominate another fault *g* when every test for *g* is also a test for *f*. For example, the fault *z*/1 dominates the fault *a*/1 in Figure 1 since the only test vector 01 for *a*/1 (shaded in the figure) is also a test for *z*/1. If a fault *f* dominates a fault *g*, then the fault *f* can be dropped from the fault list since the detection of *g* guarantees the detection of *f*.

By applying fault collapsing to the AND gate in Figure 1, we can reduce the number of faults from six to three. First, there are no redundant faults on the AND gate that should be dropped. Second, the faults *a*/0 and *b*/0 are dropped since they are equivalent to *z*/0. Finally, the fault *z*/1 is dropped since it dominates both *a*/1 and *b*/1. The collapsed fault list is thus {*a*/1, *b*/1, *z*/0}. A test set that detects the faults in the collapsed list can be derived from the table in Figure 1 as {01, 10, 11}. This test detects all faults in the collapsed fault list and consequently all six faults in the AND gate.

There are two approaches to fault collapsing: local and global. The local fault collapsing method computes the collapsed fault list for individual gates and then collects the collapsed fault lists for the gates to form the overall collapsed fault list for circuit. For example, by using fault collapsing over the gates in the circuit shown in Figure 2, we get the results shown in the figure. Both stuck at faults on line *s* (called a stem since it branches to other lines) need to be considered because *s* is not an input or output of any gate. Using local fault collapsing, we combine the faults on the gates to form the collapsed fault list for the circuit as {*s*/0, *s*/1, s_3/0, s_3/1, *a*/1, *b*/1, s_2/1, *c*/0, *d*/0, *z*/1}. Therefore, by using local fault collapsing we were able to reduce the fault list from 18 to 10.

Global fault collapsing is similar to local fault collapsing, except that we perform the same process of fault collapsing on the entire circuit as opposed to individual gates. In other words, we look for equivalent and dominance relationships among all faults in the circuit. For example, to perform global fault collapsing for the circuit in Figure 2, we compute a table for all faulty functions (called a fault table) as shown in Figure 2. It is simpler to start with the local collapsed fault list since it has less faults than the original fault list for the circuit. We then drop faults from the local collapsed fault list using redundance, equivalence and dominance relationships. It is obvious that there are no redundant faults that should be dropped. The faults *s*/0, *b*/1, *z*/1 are dropped since they dominate s_3/1. Also, the faults *s*/1, *a*/1 can are dropped since they dominate s_2/1. The fault s_3/0 is dropped since it is equivalent to *c*/0. The global collapsed fault list for the circuit is thus {s_2/1, s_3/1, *c*/0, *d*/0}. Hence, by using global fault collapsing we were able to reduce the number of faults from 18 to 4. This is in effect a 77.78% reduction from the original fault list.

Local fault collapsing can be easily scaled to large circuits. However, global fault collapsing is often avoided due to



Inputs			Correct	Faulty functions									
s	a	b	output *z*	*s*/0	*s*/1	s_3/0	s_3/1	*a*/1	*b*/1	s_2/1	*c*/0	*d*/0	*z*/1
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	1	0	0	1	0	0	1	0	1	0	0	1
0	1	0	1	1	0	0	1	1	1	1	0	1	1
0	1	1	1	1	1	0	1	1	1	1	0	1	1
1	0	0	0	0	0	0	0	0	1	0	0	0	1
1	0	1	1	0	1	1	1	1	1	1	1	0	1
1	1	0	0	1	0	0	1	0	1	0	0	0	1
1	1	1	1	1	1	1	1	1	1	1	1	0	1

Figure 2 A simple multiplexer circuit with a list of its gate faults and the resulting fault table.

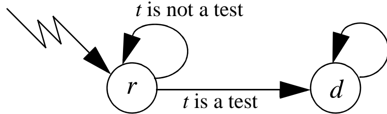


Figure 3 State diagram for tracking types of faults.

the lack of resources including the expensive computations and memory needed to determine redundance, equivalence and dominance relationships among the faults in the overall circuit. In the next section, we describe our technique for approximate global fault collapsing and the AGFC tool.

3 Approximate Global Fault Collapsing

In this section, we describe our method of approximate global fault collapsing. In this method, a large set of random vectors is used to reduce the number of faults instead of using the complete vector set for the circuit. The idea behind approximate collapsing is that the resulting faults after the simulation is an approximation of the faults from exact global fault collapsing of the circuit. As more and more vectors are applied for the simulation, the results appear more and more similar to those of exact global fault collapsing.

In order to identify redundant faults, our approximate global fault collapsing method works as follows. We first label all the faults in the circuit as redundant (*r*) faults since we have no information about the detectability of the faults. We then apply a random test vector *t* and determine the faults detected by *t* and then update the type of every fault. Figure 3 shows a state diagram for tracking the types of all faults in the circuit. Note that all faults are initially in the redundant (*r*) state and once a fault becomes detectable (*d*) due to detection by a random test *t*, it remains in that state forever.

To speed up the fault collapsing in our implementation, we apply a packet of 32 random vectors and determine the faults detected by the packet and then update the type of every fault. The type of a fault changes to detectable (*d*) if the fault is detected by at least one vector from the packet. The process is repeated for several iterations until no change is reported in the types of faults for a constant number of random packets (we use the number 100 in our simulation experiments).

Once the simulation of random vectors is completed, all the likely redundant faults (which include the random-resistant hard-to-detect faults) are removed from the overall fault list of the circuit and stored in a separate fault list. Since all of the faults remaining in the circuit's fault list are detectable, the goal of our approximate global fault technique becomes to eliminate faults from the list using equivalence and dominance relationships. We introduce the notation of a fault pair $<f_i, f_j>$ to identify the relationship between faults f_i and f_j. It is obvious that if we have n faults in the fault list, then we have $n(n-1)/2$ fault pairs.

The type of the fault pair $<f_i, f_j>$ can be any of the following:

- Equivalent (*e*) if f_i is equivalent to f_j.
- First dominating (*f*) if f_i dominates f_j.
- Second dominating (*s*) if f_i is dominated by f_j.
- Independent (*i*) if there is no relationship between f_i and f_j.

Initially, all fault pairs are of the equivalent type (*e*). As random test vectors are applied, the types of fault pairs are updated. The process is repeated for several iterations until no change is reported in the fault-pair types for a constant number of random packets (we use the number 100 in our simulation experiments). Figure 4 shows that state diagram describing the possible changes of fault-pair types. After applying a random vector *t* to a fault pair of type equivalent (*e*), the type of the fault pair does not change if (1) *t* does not detect any of the two faults in the fault pair

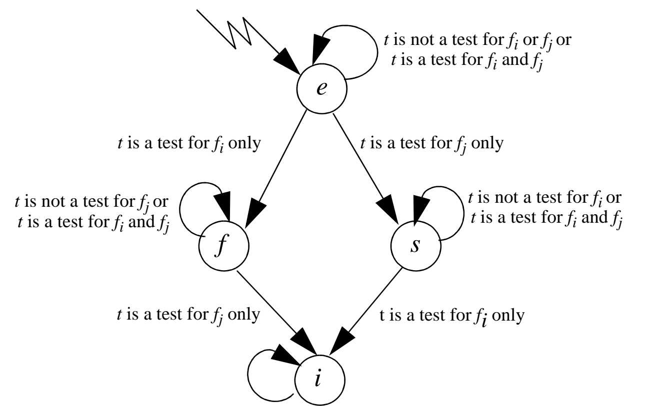


Figure 4 State diagram for tracking types of fault pairs.

or (2) t detects both faults in the fault pair. However, if t detects f_i but not f_j, then the type of the fault pair becomes first dominating (*f*). Finally, if t detects f_j but not f_i, then the type of the fault pair becomes second dominating (*s*). Similarly, the transitions from other fault-pair types in Figure 4 can be easily explained. It is interesting to notice that once the type of a fault pair becomes independent (*i*), it remains in that type forever.

The number of fault pairs with type (*e*) decreases as more random vectors are applied. On the other hand, the number of fault pairs with type (*i*) increases as more random vectors are applied. The number of fault pairs with types (*f*) or (*s*) often increases at the beginning but later decreases as more vectors are applied.

Once the simulation of random vectors is completed, the types of fault pairs are used to eliminate faults from the fault list according to the following rules: (1) If the type of the fault pair $<f_i, f_j>$ is equivalent (*e*), then eliminate either f_i or f_j; (2) If the type of the fault pair is first dominating (*f*), then eliminate f_i; (3) If the type of the fault pair is second dominating (*s*), then eliminate f_j. Note that all fault pairs with independent (*i*) type are discarded since they do not help in eliminating faults from the fault list.

After the elimination of equivalent and dominating faults from the fault list as described above, we obtain an approximate global collapsed fault list. Now, we have to choose what to do with the (likely) redundant faults that we have extracted earlier. We have two possible alternatives:

- Option 1: Discard the likely redundant faults. This will speed up the test generation since no time is wasted in targeting the redundant faults. Moreover, if the likely redundant faults are in fact redundant, the fault coverage is not affected. On the other hand, if the likely redundant faults are in fact random-resistant hard-to-detect faults, then the coverage will be reduced by a little percentage (that is often negligible).
- Option 2: Add the likely redundant faults to the approximate global collapsed fault list. This will slow down the test generation but the fault coverage will not be affected. If the majority of the likely redundant faults are in fact redundant, then plenty of test generation time is often wasted.

Since the goal is to produce an approximate global collapsed fault list that will simplify the test generation and since the loss in fault coverage is often negligible if we discard the likely redundant faults, we adopt option 1 in this paper. However, option 2 can be used if the need arises.

Since the number of fault pairs is proportional to the square of the number of faults, then the storage of the fault pairs in memory become prohibitive for the largest ISCAS-85 benchmark circuits. To solve this problem, we first simulate a random packet and then decide on the types of fault pairs. This allow us to skip storing fault pairs of type (*i*) in memory since they are not useful in fault collapsing. Moreover, during the simulation, fault pairs that change their type to (*i*) are automatically removed from memory. It should be noted that most fault pairs will ultimately be identified as of type (*i*). For example, out of 29,718,195 possible fault pairs in the ISCAS-85 circuit c6288 [3], there are at least 29,658,213 fault pairs of type (*i*). So, only 0.2% of the fault pairs are useful in determining the collapsed fault list and this is the same set of fault pairs that remain in memory at the end of the simulation.

```
/* C is the circuit*/
procedure AGFC(C);
begin
  Form the fault/error list L
  /* Redundancy Identification */
  repeat
    Select a random packet P of 32 tests
    Fault simulate using P on L
    Update fault types (redundant or detectable)
  until no change is reported in 100 random packets or
      all faults are detectable
  Remove faults with redundant type
  /* Equivalence and dominance relationship identification
  Build fault pair list
  repeat
    Select a random packet P of 32 tests
    Fault simulate using P on L
    Update types of fault pairs
  until no change is reported in 100 random packets
  Drop faults from the fault list according to the rules
  Output the results;
end;
```

Figure 5 AGFC's simulation algorithm.

In addition to the above, we can further limit the memory needed for the storage of fault pairs by enforcing a maximum value on the number of fault pairs that can be stored in memory. Once the fault pairs are initiated, they are stored in memory until the maximum value is reached. We then simulate a random packet to identify fault pairs of type (*i*) that are residing in memory. Consequently, these fault pairs are removed from memory and a new batch of fault pairs are initiated that can be stored in memory. The process is repeated until all fault pairs are processed.

The above techniques enabled us to handle the largest ISCAS-85 benchmark circuits. In order to handle even larger circuits, other techniques (to be described in Section 5) can be used.

Our AGFC tool implements the techniques presented in this section for the elimination of faults from the fault list. AGFC is written using C++ in approximately 5600 lines of code. Its simulation algorithm (a simplified version) is shown in Figure 5. The fault simulation engine used within AGFC is the same as the error simulator tool ESIM [12]. The fault simulation engine is a novel combination of parallel-pattern evaluation, multiple fault/error activation,

single fault propagation, and critical path tracing.

4 Experimental results

We now describe experimental results that illustrate the capabilities of AGFC. The circuits used in the experiments are the ISCAS-85 benchmark circuits [3] as well as few circuits from the 74X TTL IC series [11].

The approximate global fault collapsing results for the above considered circuits are shown in Table 1. The first column in the table reports the circuit name. The next column reports the possible number of SSL faults (twice the number of lines in the circuit). The next two columns report the number of inputs and outputs in the circuit, respectively. The next column reports the number of SSL faults reported by the netlist which is often the local collapsed fault list of the circuit. The next column reports the number of random packets used by AGFC. It should be clear that the number of packets should be at least 100 since this is the condition needed to stop the simulation. The next column reports the number of (likely) redundant faults reported by AGFC and the actual number of redundant faults. It should be noted that AGFC likely redundant faults are in most cases the same as the actual redundant faults. For the c2670 circuit, there is a big difference due to the large number of inputs to the circuit. The next four columns in the table report the number of fault pairs at the end of the simulation in each of the four fault pair types. It is interesting to note that most of the fault pairs are proved to be independent. The next two columns in the table report the approximate global collapsed fault list size (and percentage from total faults) produced by AGFC. It is clear that the size of the AGFC fault list can be as low as 11.54% of the total faults which corresponds to 88.46% reduction in the fault list. The AGFC fault list size actually produced the exact global collapsed fault list for the circuits with a shaded AGFC fault list size. This is established using EGFC [9], a tool that can produce the exact global collapsed fault list for small-size circuits. For the other circuits, we cannot determine at this time how far our AGFC fault list size from the exact global collapsed fault list. The next column in the table shows the execution time in seconds for a sample run on a Dell computer (2.2 GHz Pentium II, 512MB of RAM) running Windows XP. It should be noted that the time varies from one run to another since it depends on the quality of the generated random vectors. The last column in the table shows the best known collapsed fault list size from other researchers. It should be clear that our method produced way better results in comparison to prior work.

Our AGFC tool was not able to compute the approximate global collapsed fault list for the circuit c7552. AGFC terminated (after running for a long time) once a sequence of 100 random vectors did not produce any further changes to the fault pairs. The generated random vectors were not sufficient to identify a lot of the fault pairs of type (*i*) and the remaining fault pairs were not capable of collapsing the fault list. However, if more time is allowed and a stop limit greater than 100 is used, we are certain that AGFC can produce the collapsed fault list.

It is obvious from Table 1 that the execution time of AGFC for the large ISCAS-85 circuits is high. This is justified since global fault collapsing has exponential complexity. Moreover, our AGFC tool is a preliminary tool that can be further optimized to speed up the fault collapsing.

It may be the case that the execution time of AGFC is longer than the ATPG applied on a non collapsed fault list. So, where is the advantage of using AGFC? The collapsed fault list is computed once for every design module and it can be stored with the module in a design library. For large designs with multiple modules from the design library, the

Table 1 Approximate global fault collapsing results for the considered ISACS-85 and TTL circuits.

Circuit	Possible SSL faults	Circuit characteristics		Netlist SSL faults	AGFC no. of random packets	AGFC redundant faults (actual)	No. of fault pairs at end of simulation				AGFC fault list		AGFC execution time	Collapsed fault list in [4]
		Inputs	Outputs				(*e*)	(*f*)	(*s*)	(*i*)	Size	%		
c17	34	5	2	22	103	0 (0)	1	18	25	187	11	32.36	0.04	16
c432	864	36	7	524	555	4 (4)	69	1962	4508	128401	152	17.60	19.62	449
c499	998	41	32	758	475	8 (8)	28	1892	7164	271791	438	43.89	18.19	706
c880	1760	60	26	942	2905	1 (0)	63	1645	1466	439096	498	28.30	321.28	N/A
c1355	2710	41	32	1574	700	8 (8)	756	11812	11972	1200855	462	17.05	259.89	1210
c1908	3816	33	25	1879	2597	9 (9)	581	12316	19461	1715157	524	13.73	1769.14	1566
c2670	5340	233	140	2747	1971	429(117)	1266	12693	35407	2636037	802	15.02	3211.66	2317
c3540	7080	50	22	3428	5848	142 (137)	925	21563	22110	5352657	1234	17.43	10802.78	2786
c5315	10630	178	123	5350	5031	59 (59)	937	25175	21357	13947226	1781	16.75	24844.24	4492
c6288	12576	32	32	7744	329	34 (34)	4491	23101	32390	29658213	2202	17.51	2689.49	5824
c7552	15104	207	108	7550	8693	132 (131)	7102	393024	288017	26821510	Fail	N/A	106984.50	6132
7485	234	11	3	137	118	0 (0)	12	362	406	8536	48	20.51	0.48	N/A
74181	398	14	8	237	158	0 (0)	26	709	1096	26135	81	20.35	1.19	N/A
74283	208	9	5	128	141	0 (0)	22	373	524	7209	24	11.54	0.46	N/A

faults of the modules can be added together to form the fault list of the overall design. ATPG execution time for large designs with multiple modules will eventually surpass the execution time of fault collapsing for the individual modules.

5 Conclusions

AGFC is a approximate global fault collapsing tool for combinational circuits. It is based on a novel combination of parallel-pattern evaluation, multiple fault activation, single fault propagation, critical path tracing, and random simulation. The experiments reported here show that AGFC is relatively fast. They also confirm a number of interesting observations such as: (1) AGFC produced a small collapsed fault list that matches the exact global collapsed fault list for small circuits, (2) the ratio of the AGFC fault list to the total number of circuit faults can be as low as 11.54%, and (3) our method produced better results in comparison to prior work.

When circuits do become huge in size, the process of approximate global collapsing of faults eventually becomes tedious and time consuming. A method for expediting the process of global fault collapsing is to take the middle ground between global and local fault collapsing. In this hybrid process, we take a complex circuit and partition it into smaller modular components. We then perform approximate global fault collapsing for each of the components using AGFC so that we end up with a list of the remaining faults that characterize each of the components. Once this is accomplished, we recombine the entire circuit and target all the collapsed fault lists from each of the components. The premise is that the combined collapsed fault lists of the components produces an approximation of the globally collapsed fault list.

To illustrate the advantage of the above hybrid process, assume that an arbitrary large design is partitioned into k modules each with n/k faults, where n is the number faults in the large design. Instead of analyzing $n(n-1)/2$ fault pairs, AGFC needs to analyze $(n/k)((n/k)-1)/2$ fault pairs for each module. So, the overall number of fault pairs that need to be analyzed by AGFC is approximately the total number of fault pairs divided by k. So, a significant speedup can be achieved by AGFC as a result of partitioning the design.

To further speed up the fault collapsing, components can be simulated in parallel on different computers. When the simulation of all the components are completed and their collapsed fault lists are determined, the overall collapsed fault list of the circuit can then be constructed. Moreover, a library of components with their collapsed fault lists can be constructed so that it can be used for other designs.

In summary, there are two methods for collapsing the faults in a circuit. Local fault collapsing is simple to implement, but does not reduce the faults as efficiently as exact global fault collapsing. Global fault collapsing is highly desirable, but it requires extensive resources in terms of time and memory. In this paper, we have presented an approximate global fault collapsing tool which produces a more compact fault list—an approximation of the global collapsed fault list. Experimental results comparing our method to prior research show that our method achieves significantly better results.

Acknowledgment

This material is based upon work supported by the National Science Foundation under Grant No. 0092867.

References

[1] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, New Jersey, 1994.

[2] A. Lioy, "Advanced fault collapsing", *IEEE Design and Test of Computers*, Vol. 9, N0. 1, pp. 64-71, March 1992.

[3] F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in fortran", *Proc. International Symposium on Circuits and Systems,* 1985, pp. 695-698.

[4] A. V. S. S. Prasad, V. D. Agrawal, and M. V. Atre, "A new algorithm for global fault collapsing into equivalence and dominance sets", *Proc. International Test Conference*, 2002, pp. 391-397.

[5] V. D. Agrawal, A. V. S. S. Prasad, and M. V. Atre, "Fault collapsing via functional dominance", *Proc. International Test Conference*, 2003, pp. 274-280.

[6] R. Sandireddy and V. D. Agrawal, "Diagnostic and detection fault collapsing for multiple output circuits", *Proc. Design, Automation and Test in Europe*, 2005, pp. 1014-1019.

[7] M. E. Amyeen et al., "Fault equivalence identification in combinational circuits using implication and evaluation techniques", *IEEE Transactions on CAD*, Vol. 22, pp. 922-936, July 2003.

[8] H. Al-Asaad and R. Lee, "Simulation-Based Approximate Global Fault Collapsing", *Proc. International Conference on VLSI*, 2002, pp. 72-77.

[9] H. Al-Asaad, "EGFC: An exact global fault collapsing tool for combinational circuits", *Proc. IASTED conference on Circuits, Signals, and Systems*, 2005, pp. 56-61.

[10] H. Al-Asaad and J. P. Hayes, "Logic design verification via simulation and automatic test pattern generation", *Journal of Electronic Testing: Theory and Applications*, Vol. 16, No. 6, pp. 575-589, December 2000.

[11] Texas Instruments, *The TTL Logic Data Book*, Dallas, 1988.

[12] H. Al-Asaad and J. P. Hayes, "ESIM: A multimodel design error and fault simulator for logic circuits", *Proc. IEEE VLSI Test Symposium*, 2000, pp. 221-228.