Iterative [Simulation-Based Genetics + Deterministic Techniques] = Complete ATPG †

Daniel G. Saab
Coordinated Science Laboratory
University of Illinois
Urbana, IL 61801

Youssef G. Saab
Computer Science Department
University of Missouri
Columbia, MO 65211

Jacob A. Abraham
Computer Engineering Research Center
University of Texas at Austin
Austin, TX 78758

Abstract

Simulation-based test vector generators require much less computer time than deterministic ATPG but they generate longer test sequences and sometimes achieve lower fault coverage. This is due to the divergence in the search process. In this paper, we propose a correction technique for simulation-based ATPG. This technique is based on identifying the diverging state and on computing a fault cluster (faults close to each other). A set of candidate faults from the cluster is targeted with a deterministic ATPG and the resulting test sequence is used to restart the search process of the simulation-based technique. This above process is repeated until all faults are detected or proven to be redundant/untestable. The program implementing this approach has been used to generate tests with very high fault coverage, and runs about 10 times faster than traditional deterministic techniques with very good test quality in terms of test length and fault coverage.

1. Introduction

Deterministic test vector generation algorithms[1] deal very effectively with large combinational circuits at the expense of large amounts of CPU time and memory. This requirement is due to the large number of faults that needs to be considered, especially when dealing with Very Large Scale Integrated (VLSI) circuits with device counts in the order of millions today. This test generation problem has also been shown to be NP-Complete [2]. Several hueristics have been developed to speed up this process including: identifying redundancies [3,4], aborting searches that do not lead to a test [4,5], and delaying long searches [6]. On the other hand, for sequential circuits, test generation techniques at the gate-level are based on constructing an iterative array model for the circuit [1,7-11]. These techniques cannot generate high quality tests for stuck-at faults and cannot handle realistic faults, such as transistor-level and delay faults. In addition, they require tremendous amounts of CPU time and memory, which prevents their use on large practical circuits. In addition, for sequential circuits, the gate-level model does not consider any faults internal to the sequential elements (flip-flops). Therefore, the internal faults of a sequential element may or may not be detected by the derived test. Thus, the fault coverage reported by a gate-level test generator may be quite optimistic, and the actual quality of the tested product may not be as high as expected.

Ideas for exploiting the power of a fault simulator to derive tests to diagnose faults in asynchronous sequential circuits were proposed almost 30 years ago by Seshu [12]. Candidate tests were derived from previous tests by changing one input line at a time, and were evaluated using a fault simulator to determine their ability to distinguish between different faulty machines. A test vector generator that uses time frame expansion for sequential circuit with fault simulation to select test vectors to form a test set was reported in [13]. A test vector generator that uses time frame expansion in conjunction with logic simulation to compute a cost function to compute test sequences is reported in [14,15]; however, this method requires a large amount of cpu time. Cheng and Agrawal[16] developed a test generator called CONTEST, which derives a new test vector by changing a bit in a current vector based on dynamic controllability and observability cost functions; a concurrent fault simulator is used to evaluate the derived test. Recently, an approach for test cultivation of multi-level of VLSI circuits based on ideas from genetic algorithms[17] and on the use of logic and fault simulation was proposed in [18]. In this approach, logic simulation is used to rank candidate test vectors and fault simulation is used to evaluate the fault coverage of a derived test set. This approach generated test sets with high fault coverage and with lower CPU time compared with deterministic technique. This is due to the extensive use of logic simulation and the limited use of the fault simulation procedure. A technique that uses genetic algorithms with fault simulation during the ranking of test vectors is reported in [19,20]. This technique generates very compact test sets but runs slower than deterministic[9] techniques, which greatly limits its application.

None of the above ATPG tools and techniques combines an efficient simulation-based technique with a deterministic technique. In this paper, we propose an approach for test vector generation which combines the best features of deterministic and simulation-based techniques. Here, simulation is used extensively and deterministic techniques are used only in the cases when they are needed. These cases includes the identification of untestable and redundant faults, and the correction in the direction of simulation search during divergence. Divergence occurs when all vectors that are being explored using simulation techniques do not detect the remaining faults. This case happens quite often during the search process in a simulation-based ATPG, resulting in a longer search time and longer test sequences. Thus, detecting and correcting for this case increases the efficiency of the ATPG process.

2. Approach

Since conventional test generation algorithms perform poorly on large circuits, and since simulation-based techniques fail to identify untestable faults and generate very long test sets, we feel that a new approach needs to be considered for this problem. The proposed approach is based on the two facts: (1) Deterministic ATPG identifies untestable and redundant faults in a given circuit, and can generate tests for testable faults. However, it requires a large amount of CPU time and, for this reason, needs to be used only when needed and no more, (2) Simulation-based ATPG requires much smaller amounts of CPU time when compared with a deterministic ATPG, but tends to generate longer test sets and cannot find any untestable/redundant faults; in some cases, it places the circuit in an undesirable sink state which tends to stop the search process from proceeding. Therefore, any effective test generation technique that combines both techniques should be able to exploit the best capabilities of each technique and use those capabilities only as needed. Therefore, the approach to deriving high quality tests for large sequential circuits is based on exploiting the power of a Genetic-based test vector generator[18] on identifying the minimum number of times a deterministic based ATPG should be used, and on using those techniques only when needed, namely, to identify redundant/untestable faults and to correct simulation based techniques when they reach undesirable states.

3. Simulation Based Technique

The simulation based approach we used[18] to derive high quality tests for large sequential circuits is based on exploiting the power of a switch-level logic and fault simulator[21] and on using genetic algorithms to guide the test generation procedure [17,22,23]. The algorithms implicitly preserve the useful history during the search process by generating successive populations of possible solutions and by passing the best characteristics in a given population to the next.

4. Genetic algorithms

Genetic algorithms use a directed random search to locate an optimal solution for a specific problem. The directed random search simulates the natural process of evolutionary mating-induced survival used in genetics. Genetic algorithms were first introduced by Holland [24], and later used in solving NP-complete[17] and CAD problems, such as cell placement [23]. Figure 1 shows the genetic evolutionary process. A set of objects is selected from the given population and then mated. This is followed by a recombination phase which combines characteristics of the selected object to produce two or more objects through a process of separation. The newly produced objects would either survive or simply vanish depending on the survival mechanism.

† This work was supported in part by the Semiconductor Research Corporation Contract 93-DP-109 at the University of Illinois, in part by the National Science Foundation under Grant MIP-9208293, and in part by the National Science Foundation under Grant No. MIP-9222481.

©1994 ACM 0-89791-690-5/94/0011/0040 $3.50

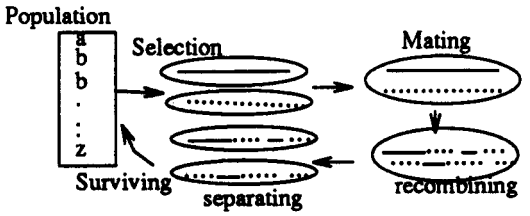

Figure 1. Evolutionary process

The genetic operators that are usually used are:

(1) Splicing/Crossover: in which individuals are mated to combine the desirable characteristics.

(2) Mutation: in which individual bits are flipped at random.

(3) Reproduction: in which the most promising individuals are forwarded to the next generation.

4.1. Objective function

In order to apply genetic techniques to the problem of test vector generation a novel objective function that ranks population members is proposed. This objective function is based on balanced circuit activity as recorded by a logic simulator, and the extent to which errors, due to faults, have been propagated to primary outputs. The first part of this objective function creates randomness that is evenly spread throughout the circuit, thus giving equal chance for every fault to be activated. The second part models and accounts for the cost of propagating a fault. This cost function is chosen because it can be computed very easily by a logic and fault simulator. If speed is the objective, then only the first part should be used which allows the processing of very large circuits. In our implementation, the first part is used extensively. The second part is used when only a few undetected faults are left.

To compute the first part of the proposed objective function, we need to monitor changes in the logical values of circuit nodes. These consist of both a change from logic low to logic high and from logic high to logic low. It is also assumed that the circuit is partitioned into a number of moderate size subcircuits. In partitioning the circuit, the following need to be considered:

(1) Dependency: the worst case is to group all primary inputs into one part.

(2) Locality: the local relationship between elements in a subcircuit. Do not put two elements in a subcircuit that are not related.

(3) Size: partitions should be of a moderate size, so changes in a part could have some effect on the part.

It was found that depth-first search (DFS)[25] based partitioning is adequate.

To formulate the objective function the following are needed: (1) We refer to a node n of subcircuit i as n_i; (2) We denote the logical state of a node n_i by $s(n_i)$; (3) The number of changes in $s(n_i)$ after the application of an input sequence of length K is denoted $|N_i^K|$.

Definition: Given a circuit partitioned into N parts (subcircuits) and a sequence of vectors R, we say that R improves the objective function if for all i,j pairs of subcircuits the following holds:

$$\mathbf{abs}(|N_i^R| - |N_j^R|) < \alpha \qquad (1)$$

where α is a small integer, and **abs** is the absolute value.

Note that the above definition ensures that activities can be balanced over all subcircuits by controlling R, α, and the size of subcircuit i. For example, if α is very large, then any sequence will improve the objective function. On the other hand, if α is small then very few sequences improve the objective. In the implementation, sequences are measured on how much they improve the overall quality of α. The size and the shape of the partitions play a very crucial role in the ranking of potential solutions. In the case where the size of the partition is one, finding a sequence that activates a certain partition is difficult. On the other hand, if the size of the partition is large, then activating that partition is easy. Redundancy and certain uninitializable circuits can also make the realization of balanced profile hard and, in some cases, impossible.

The second part of the objective function is defined as the sum of distances from a fault site to the maximum level where the corresponding error has propagated for every fault. This part of the function is computed by the fault simulator and is used only when the number of faults is very small.

5. Fault Detection in a Simulation Based ATPG

Fault detection in the proposed simulation based ATPG approach depends on the initial population, on the genetic operators, and on the fitness function. Each plays a very crucial role in the process. Given an initial population, the above simulation-based ATPG detects closely related faults that belongs to a cluster with respect to the initial population. When all faults in a cluster are detected, a mechanism needs to be devised to detect this case because fault that do not belong to the cluster, most likely, would not be detected. In order to detect these faults, a new population of test vectors needs to be generated so that tests for these faults can be derived by applying a fixed number of genetic operators to the newly generated population. This is illustrated in Figure 2. shown below. In this figure, faults are represented by small circles and are grouped into clusters (large sets). A solid arrow indicates the application of an input vector or sequence of vectors and the head of the arrow shows how close that input is from being a test for some faults. Note that, after detecting all faults in cluster 1, a series of inputs are applied and the result is for the search to move away from faults in the second cluster. In order to correct the search, a set of tests which are generated deterministically needs to be loaded into the population. These tests should detect or be very close to detecting faults that are in the center of another cluster. This is shown by a dashed arrow. This process is repeated until all faults are either detected or determined to be untestable. Note that during the activation of the deterministic technique, redundant/untestable faults are identified and removed from further consideration.

The test generator that implements this correction scheme can be in either one of two states as shown in Figure 3. The first state is a converging state in which the program should keep applying the genetic operators as long as the resulting vectors are very close to being a test for some faults. In this state, the simulation based part should keep track of faults that might be in the center of a cluster by monitoring their activity as recorded by the fault simulator. The second state is a diverging state in which the program recognizes that results from genetic process are not leading to any test or to a set of vectors close to a test. In this state, a deterministic phase is applied to generate tests for a given set of faults that are computed to be in the center of a cluster by the simulation-based phase.

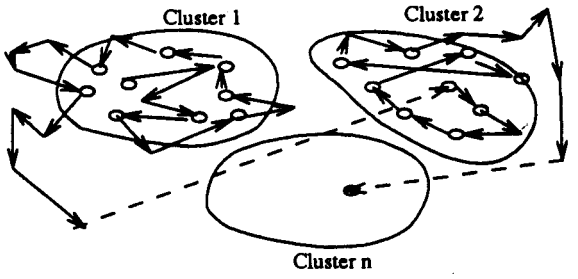

Figure 2. Fault Clusters

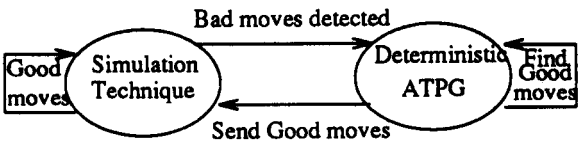

Figure 3. Search States of a Simulation Based ATPG

6. Deterministic Technique

The techniques used to implement the deterministic search procedure are based on the the two phases; (1) the forward time processing phase and (2) the state justification phase. In the first phase, an undetected fault is activated and propagated to a primary output based on a PODEM-like search and on single time frame expansion. During this phase, a state may need to be justified in order to activate the fault. In this case, the second phase is activated to justify the state. The procedure uses a 9-valued logic algebra for completeness and the split model [26]. This process consumes a large amount of CPU time and should be used only when needed. Therefore, in our implementation, this process is called from the simulation-based part with an objective of calling it as few times as possible, and only when needed and no more.

7. Algorithm Description

In order to implement the above algorithm, the circuit is partitioned by a DFS traversal into a set of subcircuits. New vectors have to be generated from old ones to produce new activity in subcircuits with low levels of activity. The main algorithm would select either one or two objects from the current population. In one case, one object is selected, a mutation is performed on the object and the resulting mutated object is either saved in the case where α of equation (1) is below a certain threshold or discarded. During mutation, bits that cause good activity and some other randomly selected bits are flipped. In the other case, two objects from the current population and either *splicing* or *mutation* are performed, depending on whether the objects consist of a single vector or a sequence of vectors. The resulting object is either saved in the case where α of equation (1) is below a certain threshold, or discarded. The objects are saved at the end of the current population. Thus, when objects from the current population are exhausted, objects from the next population are selected and a marker is placed at the end of the current population to be used as a starting point for saving objects in the next generation. This process of selection, modification, and saving is repeated until a desired fault coverage is achieved, a cpu time limit is reached, or a prespecified number of modifications is reached. During the process, if a fixed number of test vectors were added to the test set without improving the fault coverage, then a deterministic test vector generation procedure is applied to get a set of vectors to be added to both the test set and the current population. In our implementation, a very small initial test set is generated randomly. In addition, the fault simulation procedure returns a set of faults that are the center of a cluster for the case where targeting is needed.

8. Results

In order to demonstrate the power of the system on combinational circuits, we give a summary which contains the results of test generation procedure on the ISCAS 85 benchmarks[27]

shown in Table I. For example, the procedure ran in 15.13 CPU seconds requiring a total of 101 blocks of memory on a SPARC SLC detecting 520 faults and generating 81 tests for the circuit C432. Notice that in all cases, the procedure required a small amount of CPU time and memory, resulted in a high fault efficiency, and generated test sets that are not too large.

To evaluate the effectiveness of the algorithm, results for the ISCAS89 [28] benchmark circuits are also presented. In this experiment, the initial test sequences are generated randomly. The key steps in producing the high quality tests are the mutation and splicing of the vectors, the analysis of the effectiveness of the resulting candidate sequences, and the minimum number of times a deterministic procedure is called. Table II contains the result of applying the adaptive test generator on the ISCAS89 sequential circuits [28]. For example, the test generator generated a test sequence of 297 vectors for circuit s208. The deterministic test vector generation procedure was called a total of 67 times resulting in the identification of 53 untestable faults. The fault coverage resulting from the application of this test set is 63.25, the fault efficiency 87.90, and the test generator required a total of 130.12 CPU seconds.

The fault efficiency of 87.90 (compared to the 79.7 fault efficiency produced by the test generator HITEC [9].). It should be noted that both fault coverage and the number of tests generated were very close to those

Table I. Results on the ISCAS 85 combinational circuits

Circuit	# of tries	# of tests	# of faults detected	Fault Covrg Covrg	Fault eff.	CPU in sec.
c432	6864	81	520	99.2	100	15
c499	4293	89	750	98.9	100	25
c880	6842	105	942	100	100	22
c1355	6528	100	1566	99.4	100	53
c1908	8000	232	1869	99.2	99.78	110
c2670	8000	212	2618	95.3	99.56	247
c3540	8000	221	3285	95.8	99.82	153
c5315	8000	245	5281	98.7	99.64	239
c6288	8000	80	7710	99.5	100	256
c7552	8000	431	7410	98.14	99.88	522

Table II. Results on the ISCAS 89 sequential circuits

Ckt	# of ATPG Calls	# of tests	Fault Covrg	Fault Eff.	Fault Eff. HITEC	CPU time sec
s208	67	297	63.25	87.90	79.7	130
s298	58	331	86.36	96.75	94.1	121
s344	15	308	96.19	99.70	98.5	70
s349	21	367	95.42	99.71	98.5	128
s382	48	443	88.47	93.23	70.4	265
s386	85	636	80.72	83.33	90.8	151
s400	63	1461	87.41	92.16	62.2	651
s420	300	327	41.39	49.30	69.3	855
s444	78	646	87.76	94.93	76.1	127
s526	120	1191	78.01	86.48	53.2	143
s526n	160	756	73.77	82.27	N/A	160
s641	75	990	86.50	90.57	87.1	317
s713	110	918	81.75	85.54	87.6	743
s1196	25	2467	99.75	100	100	118
s1238	70	2317	94.64	99.70	99.9	177
s1423	220	1773	85.54	89.43	N/A	2471
s1488	44	3677	97.10	98.65	100	920
s1494	130	2856	92.23	94.35	100	1117
s5378	1205	3227	76.10	76.31	N/A	2369
s35932	4020	1204	89.27	99.25	99.5	11553

of HITEC, and were obtained in a very small run time. The total run time to generate all results in Table II is less than 6.4 hours of CPU time on an SUN SLC workstation with 16 Meg. of memory compared to 108 hours required for HITEC [9], to get the results on the same machine.

Table III contains characteristics of other circuits on which the procedure was applied. Results of this experiment are shown in table IV. Note that the adaptive procedure resulted in considerably higher fault efficiencies in all cases in which HITEC figures were available. This procedure clearly outperforms HITEC on speed, with comparable fault coverage and efficiencies. In fact, a higher efficiency was obtained on twelve circuits, for eight circuits a lower but comparable efficiency to HITEC was obtained, and two circuits resulted in efficiencies that are similar to the ones obtained by HITEC.

Table III. Characteristics of Other Sequential Circuits

Circuit	# of Gates	# of Inputs	# of Outputs	# of FF	# of Flts
amd2910	913	87	20	16	2573
div16	819	50	33	34	2147
mp116	626	55	18	33	1708
div32	1787	98	65	66	4544
mp132	1306	104	34	64	3228
piir8	9863	56	9	8	29689
pcont	3783	24	9	8	11272

Table IV. Results on the Other Sequential Circuits

Ckt	# of ATPG Calls	Flt Eff. CRIS	Flt Eff. HITEC	CPU time sec CRIS	CPU time sec HITEC
amd2910	300	99.33	92.42	3894.50	4796
div16	230	81.92	N/A	36460.30	N/A
mp116	98	99.64	N/A	681.57	N/A
div32	200	73.28	73.52	3800.64	142011
mp132	350	91.44	82.30	6559.57	135076
piir8	12300	98.79	72.00	133831.15	254531

9. Conclusion

In this paper, we presented a new approach for generating test vectors by combining the best features of deterministic and simulation based techniques. The simulation-based procedure is used extensively and deterministic techniques are used only in the cases when they are needed. These cases includes the identification of untestable and redundant faults and the correction in direction of simulation search during divergence, which happens when all vectors that are being explored by the simulation techniques do not detect the reamining faults. This divergence happens quite often during the search process in a simulation-based ATPG, which results in a longer search time and longer test sequences. We implemented this technique and demonstrated its usefulness on the ISCAS85 and the ISCAS89 benchmark circuits. Furthermore, we showed that the proposed techniques run, on the average, 10 times faster than traditional deterministic techniques with very competitive test length and fault coverage. We believe that this novel approach can be used to generate high quality tests for the extremely large circuits being designed today with very little scan.

10. References

[1] Miron Abramovici, Melvin A. Breuer, and Arthur D. Friedman, *Digital Systems Testing and Testable Design*. NewYork: Computer Science Press, 1990.

[2] O. H. Ibarra and S. Sahni, "Polynomially complete fault detection problems," *IEEE Trans. Comput.*, vol. C-24, pp. 242-249, March 1975.

[3] M. A. Iyer and M. Abramovici, "Low-Cost Redundancy Identification for Combinational Circuits," *Proc. 7th Int. Conf. on VLSI Design India*, pp. 315-318, 994.

[4] M. H. Schulz, E. Trischler, and T. M. Sarfert, "SOCRATES: A highly efficient automatic test pattern generation system," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, pp. 126-137, Jan. 1988.

[5] H. Fujiwara and T. Shimono, "On the Acceleration of Test Generation Algorithms," *IEEE Trans. Computers*, pp. 1137-1144, Dec. 1983..

[6] J. A. Waicukauski, P. A. Shupe, D. J. Giramma, and A. Matin, "ATPG for Ultra-Large Structrued Designs," *Proceedings of IEEE International Test Conference*, pp. 44-51, 1990.

[7] A. Ghosh, S. Devadas , and A. R. Newton, "Sequential Test Generation at the Register-Trandfer and Logic Levels," *Design Automation Conference*, pp. 580-586, 1990.

[8] H-K. T. Ma, S. Devadas, A. R. Newton, and A. Sangiovani-Vicentelli, "Test Generation for Sequential Circuits," *IEEE, Transactions on Computer Aided-Design.*, vol. CAD-7, pp. 1081-1093, Oct. 1988.

[9] T. M. Niermann and J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *European Design Automation Conference*, pp. 214-218, 1991.

[10] W.-T. Cheng, "The BACK Algorithm for Sequential Test Generation," *International Conference on Computer Aided Design*, pp. 214-218, 1991.

[11] S. Patil and P. Banerjee, "A Parallel Branch and Bound Algorithm for Sequential Test Generation ," *IEEE Transactions on Computer Aided-Design.*, vol. CAD-9, pp. 313-322, March, 1990.

[12] S. Seshu and D. N. Freeman, "The diagnosis of asynchronous sequential switching systems," *IRE Transactions on Electronic Computing*, vol. EC-11, pp. 459-465, August 1962.

[13] M. A. Breuer, "A Random and an Algorithmic Technique for Fault Detection Test Generation for Sequential Circuits," *IEEE Transactions on Computers*, vol. C-20, pp. 459-465., Nov. 1971.

[14] T. Ono and M. Yoshida, "A test Generation Method for Sequential Circuits Based on Maximum Utilization of Internal States," *ITC91.*

[15] K. Hatayama and K. Hikone, "Sequential Test Generation Based on Real-Valued Logic Simulation," *ITC92.*

[16] K. T. Cheng and V. D. Agrawal, *Unified Methods for VLSI Simulation and Test Generation.* Boston, MA: Kluwer Academic Publishers, 1989.

[17] D. E. Goldberg, *Genetic Algorithms in Search, Optimization and Machine Learning*. Massachusetts: Addison-Wesley, 1989.

[18] Daniel G. Saab, Youssef G. Saab, and Jacob Abraham, "CRIS : A Test Cultivation Program for Sequential VLSI Circuits," *IEEE International Conference on Computer Computer Aided Design*, 1992.

[19] E.M. Rudnick, J. G. Holm, D. G. Saab, and J. H. Patel, "Application of Simple Genetic Algorithms to Sequential Circuit Tesd generation," *Proc. European Design and Test Conference*, 1994.

[20] E.M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, "Sequential Circuit Tesd generation in a Genetic Algorithm Framework," *Proc. ACM/IEEE Design Automation Conference*, 1994.

[21] D. G. Saab, Robert B. Mueller-Thuns, David Blaauw, Jacob A. Abraham, and Joseph T. Rahmeh, "Hierarchical Multi-level Fault Simulation of Large Systems," *JETTA Journal of Electric Testing: Theory and Applications*, vol. 1, pp. 139-149, March, 1990.

[22] Kenneth DeJong, "Solving NP-Complete Problems with Genetic Algorithms," *International Conference on Genetic Algorithms*, 1987.

[23] J. P. Cohoon and W. D. Paris, "Genetic Placement," in *Proc. of the IEEE International Conference on Computer-Aided Design*, Santa Clara, CA, pp. 422-425, November 1986.

[24] J. H. Holland, "Adaptation in Natural and Artificial Systems," *Ann Arbor: University of Michigan Press*, 1975.

[25] A.H. Aho, J.E. Hopcroft, and J.D. Ullman, *The Design and Analysis of Computer Algorithms*. Reading, MA: Addison-Wesley, 1974.

[26] W.T. Cheng, "Split Circuit Model for Test Generation," *Proc. 25th IEEE Design Automation Conference*, pp. 96-101, June, 1988.

[27] F. Brglez and H. Fujiwara, "A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortran," *Proc. of the Int. Test Conf.*, pp. 785-794, 1985.

[28] F. Brglez, D. Bryan, and K. Kozminski, "Combinational Profiles of Sequential Benchmark Circuits," in *Proceedings of the 1989 Int. Symp. on Circuits and Systems*, Portland, Oregon, May 1989.