CONTEST : A CONCURRENT TEST GENERATOR FOR SEQUENTIAL CIRCUITS

Vishwani D. Agrawal
AT&T Bell Laboratories, Murray Hill, NJ 07974

Kwang-Ting Cheng
University of California, Berkeley, CA 94720

Prathima Agrawal
AT&T Bell Laboratories, Murray Hill, NJ 07974

ABSTRACT - This paper describes the application of a concurrent fault simulator to automatic test vector generation. As faults are simulated in the fault simulator a cost function is simultaneously computed. A simple cost function is the distance (in terms of the number of gates and flip-flops) of a fault effect from a primary output. The input vector is then modified to reduce the cost function until a test is found. The paper presents experimental results showing the effectiveness of this method in generating tests for combinational and sequential circuits. By defining suitable cost functions, we have been able to generate: 1) initialization sequences, 2) tests for a group of faults, and 3) a test for a given fault. Even asynchronous sequential circuits can be handled by this approach.

1. INTRODUCTION

Recent work [1,2] has demonstrated the feasibility of a simulation based test generation approach. The basic concept behind this approach is as follows: We start with any vector and simulate the circuit. From the simulation result, we compute a cost function. This cost is so defined that it is below a threshold only if the simulated vector is a test. If the vector is not a test, i.e., the cost is high, cost reduction by gradual changes in the vector leads to a test.

This method has proven successful for generating tests for combinational and sequential (synchronous and asynchronous) circuits. It takes circuit delays into account in the same manner as an event-driven simulator. However, a disadvantage of our earlier work is that the computation of the cost function required a new type of logic model called the threshold-value model. In the present work we have removed that modeling restriction and have used the cost function approach with a conventional fault simulator. Additionally, the approach allows us to develop several new applications of fault simulator.

We discuss three specific applications each of which is a phase in test vector generation. All functions are accomplished through a simulator using suitable cost functions.

A. Generation of Initialization Sequence. The purpose of this vector sequence is to bring flip-flops in the circuit to known states. We define the cost function as the number of flip-flops that are in the "unknown" state. The input vector is modified to reduce this cost. When the cost becomes zero, all flip-flops have been initialized.

B. Test Generation for a Group of Faults. In the initial stages of test generation, the fault list is usually long and this method is very effective for vector generation. Cost is defined as the minimum distance between a fault effect and any observation point. Observation points are the primary outputs and the flip-flop inputs. Primary outputs, however, are lower cost points than the flip-flop inputs. As the input vector is modified by the fault simulator to minimize this cost, tests are generated.

C. Test Generation for a Single Fault. A weighted sum of costs for two objectives is used as a cost function. The first objective is to sensitize the fault. The cost for this objective is defined as a dynamic controllability measure computed from the circuit states determined by the simulator. Minimization of this cost leads to activation of the fault. For the second objective, i.e., observation of fault, the cost function is the observability of the fault through all paths from the fault site to primary outputs. Observabilities are also computed dynamically.

All three cost functions have been incorporated into a concurrent fault simulator and results on actual VLSI circuits are presented.

2. SEQUENTIAL CIRCUIT TEST GENERATION

Most test generators for sequential circuits have been developed on a common principle. A combinational model of the circuit is constructed by regenerating the feedback signals from *previous-time* copies of the circuit. Thus, the timing behavior of the circuit is approximated by combinational levels. Topological analysis algorithms that activate faults and sensitize paths through these multiple copies of the combinational circuit are used to generate tests [3,4,5]. Innovations [6] and clever implementations [7,8] have produced practical test generation programs. However, there are two basic problems with this approach. First, the expansion of a sequential circuit into a combinational circuit results in increased complexity and memory requirement. Since each copy of the circuit represents one time frame (or vector), limiting the complexity restricts the test sequence length. This restriction can make many testable faults out-of-bounds for the test generator. The second problem arises due to the inability of test generation algorithms to consider the timing behavior of gates. Thus, the tests may cause races and hazards in the circuit.

CH2540-3/88/0000/0084$01.00 © 1988 IEEE

Because of these two problems, most test generators using the above principle can perform reasonably well only on circuits with up to about 1,000 gates. The test quality is often questionable.

Other approaches to sequential test generation are functional, expert-system, and simulation-based methods. The functional method [9] and the expert-system method [10, 11] require intensive user interaction, higher-level modeling libraries, and often restricted architectures. For these reasons, it has been difficult to effectively integrate them with existing design methodologies.

Simulation-Based Methods. It is a normal procedure to verify tests through a fault simulator which checks the timing behavior (races, hazards, etc.), computes the circuit response, and verifies detectability of the modeled faults. Use of simulators to enhance combinational test generation algorithms has also been proposed. For example, the effect of given values on primary inputs on internal signals (known as *implication*) can be determined by a fault simulator [12]. In a similar manner, several faults can be simultaneously considered by using a concurrent fault simulator [13]. The advantage of the information on propagation of several faults can be significant. While generating a test for a given fault, if the test generator finds that some other fault is closer to being detected, then it can switch targets to produce a test for the other fault.

Two simulation-based methods have been reported for sequential circuits. In one method, tests are primarily constructed by tracing backward through the circuit in much the same way as a combinational test generator, but all forward signal propagation is carried out by an event-driven simulator [14]. The method is effective for circuits of moderate size (few hundred gates) but the complexity of backtracking restricts its use for larger circuits.

More recently, a modified form of simulation was proposed for test generation. In this method, known as the threshold-value simulation, logic gates are modeled as modified threshold functions [1, 2]. The output signal of a gate, thus modeled, not only gives its logic value but also contains information about its controllability. A cost function is defined such that it depends on the outputs of the good and the faulty circuits. Fault detection is indicated by the cost dropping below a certain value. The input vector is progressively modified to reduce the cost. Suitable schemes for modifying the input vector provide an effective search of the input space while the cost function makes the search *directed*. There is no explicit backtracking involved and the event-driven simulation takes the timing into consideration. Effectiveness of this method for combinational and sequential circuits has been demonstrated.

Present Work. The objective of the present work is to develop a generalized directed search methodology. A logic level concurrent fault simulator is used. By defining various cost functions, vectors can be obtained for initialization, for simultaneously testing several target faults, or for testing a single target fault.

3. COST FUNCTIONS

We define three different cost functions for use by a concurrent fault simulator.

Cost Function For Initialization. This cost is defined simply as the number of flip-flops that are in the unknown state. Initially, the cost may be equal to the number of flip-flops in the circuit. The goal in the initialization phase is to reduce this cost to 0. Once the circuit is initialized, the test generator may switch to the test generation phase. It is worth noting that the cost function for initialization is derived only from good circuit simulation results and is not related to the faulty circuit behavior. If the circuit is very hard to initialize, we can relax the criterion for switching to the next phase by allowing a small number of flip-flops, say 10%, to remain uninitialized.

Cost Function For Concurrent Test Generation. Test generation begins with a circuit that is initialized and some faults are already activated but not detected, i.e., some fault effects are present in the circuit that have not been propagated to primary outputs. We define a cost function for each fault as the shortest distance from any fault effect of that fault to any primary output. The distance here is simply the number of logic gates on the path. The smaller the cost, the closer the fault is to being detected. When a fault is detected, its cost will be zero. In test generation, we reduce the cost by propagating the fault effect forward, gate by gate, until finally it reaches a primary output.

We compute the cost C_i for each fault i for the current input vector and internal state. We then compute the cost C'_i for a trial candidate vector. We can not simply compare C_i and C'_i to decide whether to accept the candidate vector or reject it. This is because we are dealing with two *lists* of cost functions instead of two numbers. That is, the search for tests is guided by a group of faults instead of a single target fault. We can devise simple rules to determine the acceptance of a vector. For example, if the combined cost for 10% of the lowest cost undetected faults is found to reduce, then the new vector may be accepted. Using this cost function, a concurrent fault simulator can generate tests.

If flip-flops are modeled as functional primitives, we treat them differently from primitive gates like AND and OR. Propagating a fault through a gate only needs setting appropriate values at the inputs of the gate. In contrast, propagation through a flip-flop requires first setting the appropriate value at its data input and then activating the clock signal. In cost computation, therefore, a large constant, say 100, is assigned to a flip-flop as the distance.

This cost function can not provide any discrimination if the fault is not activated; the cost will be infinity and will be quite useless in the search process. Also, if the circuit under test is very shallow, i.e., the number of levels of gates from primary inputs to primary outputs is small, this cost function will be less effective. Once the test generator finds that the fault coverage can not be improved further by this cost function, it automatically switches to a third cost function defined below. A strategy for switching cost functions is discussed later.

Cost Function For Single Fault Test Generation. This cost function is based on a SCOAP-like testability measure [15], but instead of computing the measure statically, it is dynamically computed depending on the current state. Also, the dynamic testability measure is used only to compare the suitability of two vectors for detecting a target fault, i.e., the testability measure is not used in an absolute sense to assess the testability of the circuit.

We define $DC1(i)$ and $DC0(i)$ as the dynamic 1 and 0 controllabilities, respectively, for node i. These are related to the *minimum* number of primary inputs that must be *changed* and the *minimum* number of additional vectors needed to control the value of the node i to 1 or 0. We call the number of inputs to be changed as the dynamic combinational controllability (DCC) and the number of vectors as the dynamic sequential controllability (DSC). These measures are dynamic because they depend upon the current state of the circuit. Since we want to keep the test sequence short, DSC is weighted heavier than DCC. For example, $DC1(i)$ and $DC0(i)$ could be the weighted sums of DCC and DSC, say $100 \times DSC + DCC$.

Dynamic Controllability Measures. If the current logic value on node i is 1, then $DC1(i)$ is defined as:

$$DC1(i)\ |_{V(i)=1} = 0$$

where $V(i)$ is the logic value on node i. Similarly, if the current logic value on node i is zero, then

$$DC0(i)\ |_{V(i)=0} = 0$$

This definition follows from the fact that no input change is needed to justify a 1(0) on node i if the value is already 1(0). Under other conditions, $DC1(i)$ and $DC0(i)$ will assume non-zero values. For example, for the output line i of an m-input AND gate, we have

$$DC1(i)\ |_{V(i)=0 \text{ or } X} = \sum_{j=1}^{m} DC1(k_j)$$

$$DC0(i)\ |_{V(i)=1 \text{ or } X} = \min_{1 \leqslant j \leqslant m} DC0(k_j)$$

where k_j is the j-th input line of the gate. Primary input controllabilities are set to 1. As explained, the controllability of sequential elements is weighted heavier. Dynamic controllabilities for a flip-flop output i is defined as:

$$DC1(i)\ |_{V(i)=0 \text{ or } X} = DC1(d) + K$$

$$DC0(i)\ |_{V(i)=1 \text{ or } X} = DC0(d) + K$$

where d is the input data signal of the flip-flop and K is a large constant, say, 100.

Activation Cost & Propagation Cost. In order to detect a stuck fault, the test generator must first find a sequence of vectors to activate the fault, i.e., set the appropriate value (opposite to the faulty state) at the fault site and then find another sequence to sensitize a path to propagate the fault effect to a primary output. Thus, the cost function should reflect the effort needed for activating and propagating the fault. The activation cost of node i stuck-at-j fault is defined as:

$$AC(i_{s-a-j}) = DCv(i)$$

where v is 0 if node i is stuck-at-1 and v is 1 if node i is stuck-at-0. This follows from the consideration that the cost of activating a stuck-at-0 (stuck-at-1) fault is the cost of setting up a 1 (0) at the fault site.

The propagation cost is a dynamic observability measure. It is 0 for primary outputs. For a fanout stem i with n fanout branches, the propagation cost is:

$$PC(i) = \min_{1 \leqslant j \leqslant n} PC(i_j)$$

where i_j is the j-th fanout branch of i. For an input signal i_a of an m-input AND gate with output signal i, we have

$$PC(i_a) = PC(i) + \sum_{\substack{1 \leqslant k \leqslant m \\ k \neq a}} DC1(i_k)$$

where i_k is the kth input line of the gate. Similar formulas are obtained for other gates.

The cost function for test generation for a single target fault combines the costs of activation and propagation. For an undetected fault F, i stuck-at-v, that is not activated, the cost is defined as:

$$Cost(F)\ |_{F \text{ not activated}} = K1 \times AC(i_{s-a-v}) + PC(i)$$

where $K1$ is a large constant for higher weighting of the activation cost. For an activated fault with N_F as the set of the nodes having fault effects

$$Cost(F)\ |_{F \text{ activated}} = \min_{i \in N_F} PC(i)$$

Notice that in this cost function reconvergent fanouts are ignored. This computational simplicity may occasionally result in the failure to detect a fault.

4. IMPLEMENTATION

A program, CONTEST (CONcurrent TEst generator for Sequential circuit Testing), has been implemented in the C language to run in the Unix® environment. It accepts logic-level circuit description. Rise and fall delays can be specified for the output of each gate in the circuit. The program works in two modes: 1) Synchronous mode, and 2) Asynchronous mode.

In the synchronous mode, clock signals and their transition sequence within a period must be specified. The test generator follows each change in primary inputs by a clock sequence. In the asynchronous mode, no clock signal is identified and the test generator treats all primary inputs alike. For circuits that are largely synchronous with a limited amount of asynchronous circuitry, test generation in the synchronous mode is recommended. If the coverage by this mode is inadequate, then for the remaining faults asynchronous mode can be used. This is because the speed of test generation depends upon the number of primary inputs that must be manipulated. In the synchronous mode, clock signals are prespecified.

A fault list is another input to the test generator. CONTEST contains an event-driven concurrent fault simulator. Race analysis in feedback structures is automatic and is performed through logic modeling [16]. By default, potentially detectable faults (that produce an

unknown faulty output) are considered detected. This option can be turned off by the user. If the number of changes in a signal within one vector period exceeds a prespecified number, the simulator assumes oscillation and sets the gate output to the unknown state.

The program works in three phases. Each phase can be independently run. A phase is characterized by its specific function: initialization, concurrent fault detection, or single fault detection. Based on the function, the appropriate cost function is used. A phase begins with one or more vectors (supplied by the user or a previous phase or the default all-0 vector). The cost function is computed for the *last vector* in the sequence. *Trial vectors* are obtained by one-bit changes in the last vector. Ideally, the trial vector with lowest cost should be selected as the next vector in the test sequence. In CONTEST, we adopt a greedy heuristic. The program generates trial vectors by changing the bits in the order in which the primary inputs are specified in the circuit description. As soon as a trial vector with a cost lower then the last vector in the test set is found, it is included in the test set. Further processing of the remaining trial vectors is suspended and trial vectors for the newly added *last vector* are processed. If no cost reduction can be obtained by all single-bit-change trial vectors, then the search is abandoned assuming that the algorithm has reached a local minimum.

The Unit Hamming Distance Heuristic. In directed search, before deciding on a move, one normally considers the cost at all neighboring points in the search space. We have defined the neighborhood of a given vector as all vectors that are at unit Hamming distance (i.e., differ in just one bit) from it. This seemingly arbitrary choice is similar to computing *partial differences* with respect to the input variables for a steepest descent solution. In the present situation, however, a test can be guaranteed if the neighborhood includes vectors that differ in 1, 2, ..., n bits where n is the number of primary inputs. Thus, our heuristic corresponds to considering only n of these 2^n neighbors. The judgement on whether we have achieved a practical trade-off in reducing computational complexity without giving up accuracy should be based on the experimental results of Section 5.

Cost Function Computation. Three types of cost functions are described in Section 3. Vector generation in each phase is directed by a different cost function.

Phase 1 (Initialization). All flip-flop output signals are identified. The number of unknown signals among these is used as the cost. As more flip-flops are initialized to known states, the cost reduces. A complete initialization corresponds to zero cost. User can specify the acceptable percentage of uninitialized flip-flops. The default in CONTEST is 10 percent.

Phase 2. This is the concurrent test generation phase. All faults in the list are treated as targets. Cost is computed as described in Section 3. Typically, this phase begins with initialization vectors. In general, user can also supply vectors for which concurrent fault simulation is performed to eliminate the detected faults. For remaining faults, new vectors are generated. This phase terminates when either an adequate fault coverage is achieved or cost reduction with one-bit input change becomes impossible. At the end of this phase, in practice, a coverage of about 85% is achieved.

Phase 3. In this phase, activation and propagation cost, as discussed in Section 3, are computed for each fault in the list. The lowest cost fault is targeted for detection. New input vectors are created to further lower the cost of the target fault until it is detected or the search is abandoned due to a local cost minimum. As new vectors are added to the sequence, concurrent fault simulation simultaneously eliminates any other detected faults from the list. This phase ends when either an adequate coverage is achieved or all faults that were left undetected at the end of phase 2 have been processed.

5. RESULTS

Before discussing test generation for larger circuits, we will illustrate some features of CONTEST.

Example 1: The circuit in Fig. 1, taken from Marlett's paper [8], was originally used by Miczo [17] to illustrate the problem of initialization. CONTEST generated a 4-vector initialization sequence: 00, 01, 00, 01. Starting in the unknown state, this sequence leaves the circuit in the 00 state. To observe how the test generator will move $Q1$ to 1, we generated a test for the fault $Q1$ stuck-at-0. Since it is a single fault target, CONTEST used Phase 3 to generate four more vectors (10,11,00,01) to detect it.

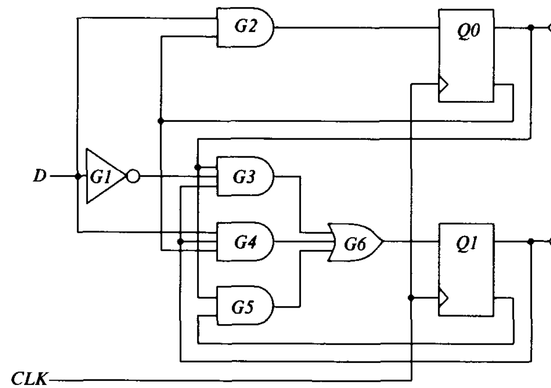

Fig. 1 Circuit of Example 1.

In a separate run, a set of 18 vectors was generated to detect 24 out of 25 faults. The remaining fault, third input (from $Q1$) of $G3$ stuck-at-1, is redundant.

Example 2: Muth [5] used the asynchronous circuit of Fig. 2 to illustrate the necessity of a nine-value model when a combinational test generation method is employed. Following Muth's illustration, we ran CONTEST to generate a test sequence for the fault "*d* stuck-at-1". Four vectors were produced: 000, 100, 101, 111. The last two vectors are the same as given in Muth's paper [5]. Two extra vectors are generated here because CONTEST starts with an arbitrary 000 vector and then brings the circuit to the appropriate state.

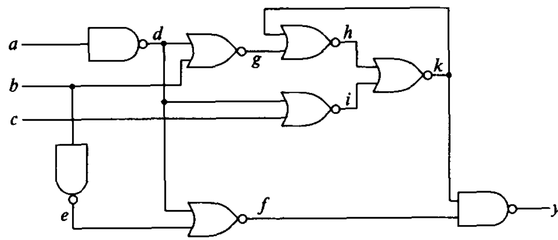

Fig. 2. Muth's circuit used in Example 2.

Example 3: A test sequence for the four faults at the inputs of the NAND gate in the circuit of Fig. 3 was generated by CONTEST. This sequence had fourteen vectors: 00, 01, 10, 11, 00, 01, 10, 11, 00, 01, 10, 11, 00, and 11. The first eight vectors, generated in Phase 1, initialize the circuit in the 0101 state. Since the fault list had only four faults, the program switched to Phase 3. One input stuck-at-1 fault was detected at the 10th vector, both input stuck-at-0 faults were detected at the 12th vector, and the other input stuck-at-1 fault was detected at the 14th vector.

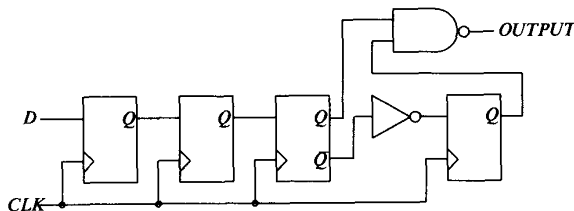

Fig. 3. Circuit used in Example 3.

Next, we give CONTEST results for eight sequential circuits. Some characteristics of these circuits are listed in Table 1. Only one circuit, MANNY*, is completely asynchronous. SSE and PLANET are finite-state machines with PLA-like implementation†. MULT4 is a 4-bit Booth multiplier designed by an automatic synthesis program. TLC is a traffic light controller circuit. MI is another finite-state machine implemented with random logic. CHIP-A is a CMOS chip designed for a graphics terminal. CHIP-B is another custom chip with standard cell design. CHIP-A is completely synchronous. CHIP-B is synchronous with the exception of one asynchronous flip-flop.

Figure 4 is a sample of CONTEST results. These vectors were generated for the TLC circuit. First four vectors were generated in the initialization phase. Test generator switched to Phase 3 after generating 550 vectors when the coverage was 85%. The coverage characteristic of random vectors, shown in the figure for comparison, is typical for sequential circuits.

Table 2 shows the results obtained by CONTEST for the eight circuits. For comparison, results of another sequential test generator, STG [7], are given. STG uses the conventional method of time-frame expansion to which

* Courtesy of P. Goel of Gateway Design Automation Corp.

† Obtained from Tony Ma of University of California, Berkeley, CA.

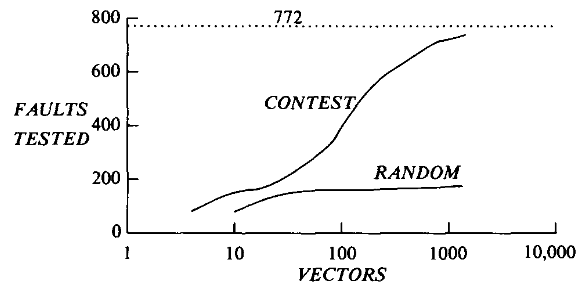

Fig. 4. Fault coverage of vectors for TLC circuit.

Table 1. Sequential Circuits used for Test Generation

Circuit	# Gate	# Input	# Output	# FFs	# Faults
MANNY	**26**	**5**	**3**	**7‡**	**67**
SSE	**207**	**8**	**7**	**6**	**454**
MULT4	**382**	**10**	**9**	**15**	**540**
TLC	**355**	**4**	**6**	**21**	**772**
PLANET	**690**	**9**	**19**	**6**	**1582**
MI	**779**	**15**	**14**	**18**	**1629**
CHIP-A	**1112**	**13**	**13**	**39**	**1643**
CHIP-B	**1539**	**17**	**5**	**73§**	**2533**

it applies a combinational test generation algorithm. In all cases, CONTEST showed better performance. Fault coverages reported in Table 2 were verified through an MOS fault simulator. None of the test generation programs can identify redundant faults. The redundancies of 16.3% and 3.29% in SSE and PLANET, respectively, are typical of PLA structures [18]. Redundant faults in MULT4 and CHIP-A were manually analyzed and were found to be undetectable due to the specific logic model used for CMOS tristate devices. STG does not have an interactively running fault simulator. Thus, the fault list is not updated and the test generator produces vectors for each fault separately. Also, for each test, a new initialization is attempted. These two reasons increase the run time for the STG. Also, STG tests for MANNY and CHIP-B had problems because of the asynchronous operation in those circuits.

Run time estimates of CONTEST are conservative as they are based on our experimental implimentation. Its speed, which largely depends on the speed of the fault simulator, can be significantly improved.

CONTEST was further compared with a commercially available test generation program [8]. This program uses an extended backtrace technique for an efficient implementation of the path sensitization procedure. It generated 330 vectors for the CHIP-A circuit. Although the reported coverage was higher, the same fault simulator that was used for the results of Table 2 gave a coverage of of 90.75% for these vectors. This is not surprising since results from different simulators often vary due to modeling differences. Considering the 4.37% redundant faults in

‡ Asynchronous flip-flops formed by cross-coupled NAND gates.

§ Includes one asynchronously controlled flip-flop.

Table 2. Results of CONTEST and STG										
Circuit	Fault Cov. (%)		Provable Red. Faults (%)	Total Fault Cov. (%)		# Test Vectors		CPU Secs. (VAX 8650)		
	CON	STG		CON	STG	CON	STG	CON	STG	
MANNY	100.00	83.95	0	100.00	83.95	32	219	5	NA■	
SSE	83.26	83.20	16.30	99.56	99.50	561	676	291	1134	
MULT4	97.04	92.78	0.37	97.41	93.15	364	148	838	1490	
TLC	94.69	94.64	NA	94.69	94.64	1256	5340	3312	32590	
PLANET	95.13	57.71	3.29	98.42	61.00	1439	132	3120	19388	
MI	94.53	NA	NA	94.53	NA	1358	NA	1261	NA	
CHIP-A	93.73	84.11	4.37	98.10	88.48	1031	384	98432	NA	
CHIP-B	91.28	NA	NA	91.28	NA	1034	NA	77904	NA	

this circuit the fault coverage can be updated to 95.12%. In comparison, the coverage obtained by CONTEST is 98.10% and that by STG is 88.48%.

Since STG also uses a path sensitization procedure, the number of vectors (384) it generated is similar to the other program. CONTEST, on the other hand, consistently generated more vectors than the other two programs. This is a consequence of the one-bit change heuristic. In general, neighboring vectors in a CONTEST-generated sequence have fewer bit changes than in sequences generated by other methods. As a result, more vectors are needed to take the circuit to a desired state starting from some given state. In practical testing environment, this may be an advantage because too many simultaneous input changes can produce hazards in the logic and power supply fluctuations due to current surge. It is our belief that the run time of CONTEST can be significantly improved since it totally depends upon the fault simulation program used. The present version uses an experimental concurrent fault simulator.

In view of the results presented here, our unit Hamming distance heuristic seem to work well. In most cases, we were unable to manually generate tests for the faults that were left undetected by CONTEST. Finding redundancies in sequential circuits was even harder. When the final fault coverage is lower than the desired goal we see two options. The first option is to start with a different (randomly selected) vector and attempt generation of tests for the undetected faults. The second option is to expand the one-bit change heuristic to include two-bit, three-bit, ... changes. One should, however, expect a rapid increase in the amount of computations.

6. CONCLUSION

We have presented a practical and completely automatic test generator for sequential and combinational circuits. Its effectiveness and accuracy is derived from the use of an event-driven concurrent fault simulator. The algorithm involves no explicit backtracking.

The performance of the new method, as shown by a preliminary implementation, exceeds that of mature tools implemented using conventional test generation methods. Implementation is simple since it can be incorporated in any existing fault simulation program without significant changes in the data structure. Test generation can be made completely automatic starting right from initialization and even the presence of asynchronous circuitry poses no problem. Tests can also be generated for any other fault models that can be simulated.

■ NA ≡ Not available at the time of writing.

Acknowledgment - Authors thank W-T Cheng and S. Wu for help with STG and the other program used for comparison. They are grateful to Prof. E. S. Kuh for his support of this work.

REFERENCES

[1] V. D. Agrawal and K. T. Cheng, "Threshold-Value Simulation and Test Generation," *Testing & Diagnosis of VLSI & ULSI (Proc. NATO Adv. Study Inst.)*, Como, Italy, June 1987.

[2] K. T. Cheng and V. D. Agrawal, "A Simulation-Based Directed-Search Method for Test Generation," *Proc. Int. Conf. Comp. Des. (ICCD'87)*, Port Chester, NY, pp. 48-51, Oct. 1987.

[3] H. Kubo, "A Procedure for Generating Test Sequences to Detect Sequential Circuit Failures," *NEC J. Res. Dev.*(12), pp. 69-78, Oct. 1968.

[4] G. R. Putzolu and J. P. Roth, "A Heuristic Algorithm for the Testing of Asynchronous Circuits," *IEEE Trans. on Computers*, Vol. C-20, pp. 639-647, June 1971.

[5] P. Muth, "A Nine-Value Circuit Model for Test Generation," *IEEE Trans. on Computers*, Vol. C-25, pp. 630-636, June 1976.

[6] R. A. Marlett, "EBT: A Comprehensive Test Generation Technique for Highly Sequential Circuits," *Proc. Des. Auto. Conf.*, Las Vegas, Nevada, pp. 335-339, June 1978.

[7] S. Mallela and S. Wu, "A Sequential Circuit Test Generation System," *Proc. Int. Test Conf.*, Philadelphia, PA, pp. 57-61, Nov. 1985.

[8] R. A. Marlett, "An Effective Test Generation System for Sequential Circuits," *Proc. Des. Auto. Conf.*, Las Vegas, Nevada, pp. 250-256, June 1986.

[9] F. J. Hill and B. M. Huey, "A Design Language Based Approach to Test Sequence Generation," *Computer*, Vol. 10, pp. 28-33, June 1977.

[10] M. J. Bending, "Hitest: A Knowledge-Based Test Generation System," *IEEE Design & Test of Computers*, Vol. 1, pp. 83-92, May 1984.

[11] N. Singh, *An Artificial Intelligence Approach to Test Generation*, Kluwer Academic Publishers, Boston, MA, 1987.

[12] E. Kjelkerud and O. Thessen, "Generation of Hazard Free Tests using the D-Algorithm in a Timing Accurate System for Logic and Deductive Fault Simulation," *Proc. Des. Auto. Conf.*, San Diego, CA, pp. 180-184, June 1979.

[13] Y. Takamatsu and K. Kinoshita, "CONT: A Concurrent Test Generation Algorithm," *Fault-Tolerant Computing Symp. (FTCS) Digest of Papers*, Pittsburgh, PA, pp. 22-27, July 1987.

[14] T. J. Snethen, "Simulator Oriented Fault Test Generator," *Proc. 14th Des. Auto. Conf.*, New Orleans, LA, pp. 88-93, June 1977.

[15] L. H. Goldstein, "Controllability/Observability Analysis of Digital Circuits," *IEEE Trans. Circ. and Syst.*, Vol. CAS-26, pp. 685-693, Sept. 1979.

[16] E. G. Ulrich and T. Baker, "Concurrent Simulation of Nearly Identical Digital Networks," *Computer*, Vol. 7, pp. 39-44, April 1974.

[17] A. Miczo, "The Sequential ATPG: A Theoretical Limit," *Proc. Int. Test Conf.*, Philadelphia, PA, pp. 143-147, Oct. 1983.

[18] H. T. Ma, Private communication.