⊙ 速報研究論文

Fault Coverage Improvement of Test Patterns for Combinational Circuit using a Genetic Algorithm

H. C. Park*

유전알고리즘을 이용한 조합회로용 테스트패턴의 고장검출률 향상

박 휴 찬*

Key words : Fault coverage(고장검출률), Test pattern(테스트패턴), Combinational circuit(조합회로), Genetic algorithm(유전알고리즘)

Abstract

Test pattern generation is one of most difficult problems encountered in automating the design of logic circuits. The goal is to obtain the highest fault coverage with the minimum number of test patterns for a given circuit and fault set. Although there have been many deterministic algorithms and heuristics, the problem is still highly complex and time - consuming. Therefore new approaches are needed to augment the existing techniques. This paper considers the problem of test pattern improvement for combinational circuits as a restricted subproblem of the test pattern generation. The problem is to maximize the fault coverage with a fixed number of test patterns for a given circuit and fault set. We propose a new approach by use of a genetic algorithm. In this approach, the genetic algorithm evolves test patterns to improve their fault coverage. A fault simulation is used to compute the fault coverage of the test patterns. Experimental results show that the genetic algorithm based approach can achieve higher fault coverages than traditional techniques for most combinational circuits. Another advantage of the approach is that the genetic algorithm needs no detailed knowledge of faulty circuits under test.

1. Introduction

Test pattern generation is one of most difficult problems encountered in automating the design of logic circuits. The goal is to obtain the highest fault coverage with the minimum number of test patterns for a given circuit and fault set. Although many deterministic algorithms and heuristics have been proposed to solve the problem[1~4], it is still highly complex and time -

* 한국해양대학교 자동화 · 정보공학부(원고접수일 : 98년 6월)

consuming. To cope with the complexity of the problem, we need new approaches to augment the existing techniques.

In this paper, we first define the problem of *test pattern improvement* as a restricted sub-problem of the test pattern generation, and then propose a new approach to the problem by use of a genetic algorithm. The test pattern improvement problem is to maximize the fault coverage with a fixed number of test patterns for a given circuit and fault set. Improving the fault coverage with the fixed number of test patterns is a very useful and plausible approach under some practical situations, because the testing time of a circuit is in proportion to the number of test patterns. That is, if the permitted time to be used for testing a large number of devices is not enough, we must restrict the number of test patterns in order to complete all of the tests. In addition, the fault coverage must be maximized to increase the test patterns' ability distinguishing whether each tested device is good or not. The size of search space for the test pattern improvement problem is ${}_{2^n}C_k$ when the number of primary inputs of a given circuit is n and the number of test patterns is k. Furthermore, the search space is discontinuous, with good test patterns adjacent to bad ones, so traditional methods for local improvement cannot be relied upon to find globally competitive solutions. Some encouraging results obtained from similar problems motivate us to employ a genetic algorithm for the test pattern improvement problem.

Genetic algorithms are search procedures based on the mechanics of natural selection and genetics. They have been widely used in several applications and accepted as the methods for solving difficult optimization problems. We briefly review some terminologies and structures related to the genetic algorithms. Genetic algorithms work with a population of individuals, each of which represents a candidate solution of the given problem. For each individual in the population, a measure of its quality, called fitness, is calculated. Pairs of the individuals are selected from the population based on the fitness to become parents. Then, reproduction occurs between the selected pairs of individuals to produce new population. Some genetic operators such as crossover and mutation can be used in this phase. The newly created population becomes the next generation, and the above process is repeated[5~8].

An important feature of our genetic algorithm based approach for the test pattern improvement problem is that the solution does not rely on the details of a given circuit. Therefore, the approach can be used for various kinds of logic circuits and fault models. We present some details on the concept and implementation of the genetic approach to the test pattern improvement problem. Results from some experiments are also discussed.

2. Test Pattern Improvement using a Genetic Algorithm

In this section, we describe in detail the genetic algorithm based approach to the problem of test pattern improvement. To clarify the discussion, some of the terms used in this paper are reviewed as follows. A *test pattern* is a string of n logic values (0 or 1) that are applied to the n corresponding primary inputs of a circuit at the same time. A *test set* is a set of test patterns applied to a circuit one by one to detect faults in the circuit. The *fault coverage* of test patterns (test set) is the ratio of faults detected by the test patterns to the total number of possible faults in a circuit.

The following shows the outline of the *genetic*

algorithm based test pattern improvement, called GA - TPI. It follows the core structure of genetic algorithms.

[GA - TPI: Genetic algorithm based test pattern improvement]

Step 1 (*Initialization*): Initialize genetic parameter values such as crossover probability P_c and mutation probability P_m. Randomly generate an initial population of test sets $\{c_1, c_2, \cdots, c_p\}$ where the p is a population size.

Step 2 (*Evaluation*): Calculate the fault coverage of each test set c_i in the current population through a fault simulation.

Step 3 (*Selection*): Select pairs of test sets from the current population.

Step 4 (*Crossover*): Apply crossover operator with the probability P_c to each of the selected pairs in Step 3 to generate $p - 1$ test sets.

Step 5 (*Mutation*): Apply mutation operator with the probability Pm to each of the generated $p - 1$ test sets in Step 4.

Step 6 (*Elitist*): Add the best test set in the previous population to the current population.

Step 7 (*Termination test*): If a prespecified stopping criterion is satisfied, stop the algorithm. Otherwise return to Step 2.

Although the genetic algorithm provides a very general search mechanism for a wide range of applications, the effectiveness of the overall algorithm depends partially on the characteristics of the given problem. Therefore, the genetic algorithm needs to be specialized for the problem of test pattern improvement. The key factor in enabling the genetic algorithm to solve the problem efficiently is an appropriate representation of each individual on which genetic operators manipulate. For the test pattern improvement problem, an individual should be a test set which is a set of test patterns. That is, if we denote a test pattern as v_i, then an individual consisting of k test patterns is represented as $v_1v_2\cdots v_k$ which is actually coded in the form of binary string as a whole. The string length of the individual is *primary_ input* $* k$, because the length of each test pattern is the same as the number of primary inputs for a given circuit. Each generation is a population of the individuals $\{c_1, c_2, \cdots, c_p\}$, where c_i is an individual and p is the size of the population.

The next requirement for the genetic algorithm is a means of measuring the quality of each individual in the population. We use the fault coverage of each individual as its measure of fitness. The fault coverage is obtained through a fault simulation which applies the test patterns represented in the individual to the target circuit. Also, we adopt the fitness scaling technique, such as linear scaling, to prevent premature convergence at early generations and randomness at later generations. Based on their scaled fitness, parent individuals are selected by use of the biased roulette wheel selection mechanism in which the size of each slot is proportional to its measure of fitness. Offspring are created by mating and applying genetic operators to these parents. With our representation, defining these genetic operators is not a difficult task. The genetic operators employed in this paper are crossover and mutation. Two - points crossover is used because it offers more genetic diversity than one - point crossover and preserves more schemata than multiple - points crossover. The standard mutation operator is employed throughout the search. Another useful mechanism we adopt is *elitism* in which the best individual of the previous generation survives in

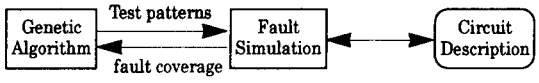


Fig. 1 Conceptual view of GA – TPI

the next generation.

The conceptual view of GA – TPI is shown in Fig. 1. In the figure, the genetic algorithm and the fault simulation are the core of GA – TPI. The genetic algorithm creates test patterns represented by an individual through several genetic mechanisms, and the fault simulation evaluates them for the given circuit and fault model. The main advantage of separating the genetic algorithm from the fault simulation is that the genetic algorithm can improve the quality of test patterns with respect to their fault coverage without knowledge of the details of the circuits. That is, the genetic algorithm needs only the fault coverage of each individual represented by a binary string. Therefore, the genetic algorithm can be used to improve test patterns for various type of circuits and fault models, because its performance does not depend on the given circuit and fault model. We concentrated only on the stuck – at fault model of combinational circuits in the current experiment.

3. Experimental Results and Discussions

We constructed an experimental environment of the GA – TPI to evaluate the feasibility of the genetic approach to the test pattern improvement problem. The fault simulation was conducted by the fault simulator, *fsim*. The test circuits used in our experiments are the ISCAS'85 benchmark circuits[2] whose characteristics are depicted in Table 1.

The experiments were conducted as follows. In the first set of experiments on the C432, one of the benchmark circuits, we tried to find the

Table 1. Characteristics of test circuits

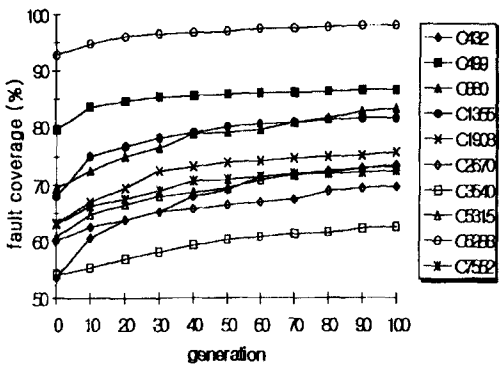
Circuit Name	Primary Inputs	Primary Outputs	Lines	Interior Gates	Faults
C432	36	7	432	153	524
C499	41	32	499	170	758
C880	60	26	880	357	942
C1355	41	32	1355	514	1574
C1908	33	25	1908	855	1879
C2670	233	140	2670	1129	2747
C3540	50	22	3540	1647	3428
C5315	178	123	5315	2184	5350
C6288	32	32	6288	2384	7744
C7552	207	108	7552	3405	7550

best parameter values affecting the performance of the genetic algorithm. The values of parameters, such as crossover rate and mutation rate, were varied and then the fault coverages obtained from each distinct set of parameter values were compared. These preliminary experiments provided that no parameter has a noticeable effect on the fault coverage. The variance was less than 3%, which means the performance of GA – TPI is less sensitive to the particular parameter settings. The best parameter values were 0.7 for the crossover rate, 0.008 for the mutation rate, and 6.0 for the scaling factor. Using these parameter values, we experimented on all of the benchmark circuits. The number of test patterns (k) contained in each individual was 10. It means that we tried to maximize the fault coverage with 10 test patterns. The population size (p) was 10, and the initial population was created randomly.

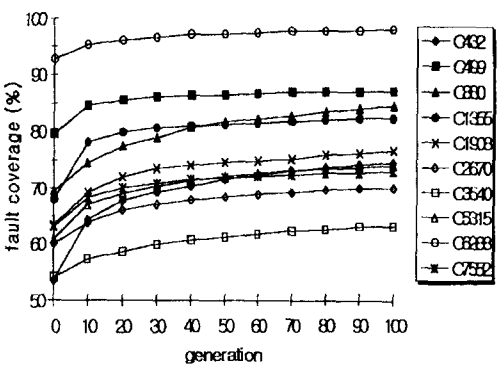
The fault coverages obtained by GA – TPI with two different sets of parameter values at generation 100 are depicted in Table 2. We also compared them against the fault coverages of randomly – generated test patterns and the highest fault coverages obtained by using the deterministic algorithm, *atalanta*. As the table shows, GA – TPI improves the fault coverages about 5%~20% compared to those of initial

Table 2. Comparisons of fault coverages

Circuit Name	Fault Coverage (%) (number of test patterns : 10)			
	Random	Deterministic (*atalanta*)	GA - TPI crossover: 0.9 mutation: 0.016	GA - TPI crossover: 0.7 mutation: 0.008
C432	53.6	64.7	73.3	74.2
C499	79.7	86.5	86.5	86.9
C880	69.4	75.2	83.4	84.3
C1355	68.0	79.3	81.7	82.0
C1908	63.2	71.4	75.7	76.3
C2670	60.1	65.9	69.7	69.7
C3540	53.8	54.3	62.5	62.8
C5315	60.8	66.4	73.1	73.7
C6288	92.7	97.6	98.1	97.7
C7552	62.9	67.1	72.5	72.6



(a) crossover rate : 0.9, mutation rate : 0.016



(b) crossover rate : 0.7, mutation rate : 0.008

Fig. 2 Improvement of fault coverages

random test patterns, and all the results are better than the deterministic algorithm. Fig. 2 presents the detailed improvement of fault coverages as the generation increases. As shown in the figure, GA - TPI improves the fault coverages efficiently within a few generations. We can also clarify that the performance of GA - TPI is less sensitive to the particular parameter settings.

Although the current implementation of GA - TPI works well on the test pattern improvement application, the performance can be further improved in several ways. First, the repairing procedure or more special genetic operators, such as inversion and shuffle crossover, may be used to make the search more efficient. These can also overcome the limitations of binary string representation of candidate solutions for combinatorial optimization problems such as the test pattern improvement of this paper. Secondly, although genetic algorithms can escape from local optima easily, there is a possibility that it cannot exploit the global optimum near the current point for possible improvement. To overcome this phenomenon, we need other techniques, such as a hybrid algorithm with hill - climbing. Thirdly, the adaptive change of parameter values based on the progress of the algorithm may give better performance. A good example of this paradigm is the temperature in the simulated annealing.

4. Conclusion

The problem of test pattern generation is highly complex and time - consuming. To cope with the complexity of the problem, we first defined a restricted subproblem, test pattern improvement, which maximizes the fault coverage with a fixed number of test patterns for a given circuit and fault set. We then adapted a genetic algorithm as the conceptual basis to solve the test pattern improvement problem. The genetic algorithm based approach is shown

to be a feasible and useful technique. Experimental results show that the fault coverages achieved by the genetic algorithm based approach is higher than those obtained by conventional approaches.

Further study needs to be done toward the following directions. First, our approach can be applied to sequential circuits. We expect to obtain similar results in sequential circuits by virtue of the robustness of genetic algorithm itself. Secondly, a parallelization of our approach on a multi-processor computer can obtain faster execution and better performance.

Acknowledgement

The author wishes to thank Professor D. S. Ha at the Virginia Polytechnic & State University for providing the fault simulator, *fsim*, and the test pattern generator, *atalanta*.

References

1) H. Fujiwara and S. Toida, "The complexity of fault detection problems for combinational logic circuits," *IEEE Trans. on Computers*, vol. C-31, no. 6, pp. 555-560, June 1982.
2) F. Brglez and H. Fujiwara, "A neutral netlist of 10 combinational benchmark circuits and a target translator in Fortran," *Proc. Int. IEEE Symp. on Circuits and Systems*, June 1985.
3) R. L. Pickholtz, *Digital Systems Testing and Testable Design*, Computer Science Press, 1990.
4) J. D. Calhoun and F. Brglez, "A framework and method for hierarchical test generation," *IEEE Trans. on Computer-Aided Design*, vol. 11, no. 1, pp. 45-67, Jan. 1992.
5) D. E. Goldberg, *Genetic algorithms in Search, Optimization, and Machine Learning*, Addison-Wesley, 1989.
6) L. Davis, ed., *Handbook of Genetic algorithms*, Van Nostrand Reinhold, 1991.
7) M. Srinivas, "Genetic algorithms: a survey," *IEEE Computer*, pp. 17-26, June 1994.
8) T. Back, ed., "Genetic algorithms versus experimental methods: a case study," *Proceeding of the Seventh International Conference on Genetic Algorithm*, Morgan Maufmann, 1997.

저 자 소 개

박휴찬(朴烋讚)
1963년 3월생. 1995년 서울대학교 공과대학 전자공학과 졸업. 1987년 한국과학기술원 전기및 전자공학과 졸업(석사). 1995년 동대학원 졸업(박사). 1987년-1990년 LG 반도체(주) 연구원. 1997년-현재 한국해양대학교 자동화·정보공학부 전임강사. 당학회회원.