An Iterative Heuristic for State Justification in Sequential Automatic Test Pattern Generation

Aiman H. El-Maleh **Sadiq M. Sait** **Syed Z. Shazli**

Computer Engineering Department
King Fahd University of Petroleum and Minerals
Dhahran-31261, Saudi Arabia
e-mail: {aimane,sadiq,shazli}@ccse.kfupm.edu.sa

Abstract

State justification is one of the most time-consuming tasks in sequential Automatic Test Pattern Generation (ATPG). For states that are difficult to justify, deterministic algorithms take significant CPU time without much success most of the time. In this work, we adopt a hybrid approach for state justification. A new method based on Genetic Algorithms is proposed, in which we engineer state justification sequences vector by vector. The proposed method is compared with previous GA-based approaches. Significant improvements have been obtained for ISCAS benchmark circuits in terms of state coverage and CPU time.

1 Introduction

With today's technology, it is possible to build very large systems containing millions of transistors on a single integrated circuit. Designing such large and complex systems while meeting stringent cost and time-to-market constraints requires the use of computer-aided-design (CAD) tools. Increasing complexity of digital circuits in very large scale integration (VLSI) environment requires more efficient algorithms to support the operations performed by CAD tools [1]. Testing of integrated circuits is an important area which nowadays accounts for a significant percentage of the total design and production costs of ICs. For this reason, a large amount of research efforts have been invested in the last decade in the development of more efficient algorithms for the Automatic Test Pattern Generation (ATPG) for digital circuits [2]. In order to obtain acceptably high quality tests, design for testability (DFT) techniques are in use [3]. The first technique, called *full-scan design*, can be used to reduce the sequential test generation problem to a less difficult combinational test generation problem. In this technique, all memory elements are chained into shift registers so that they can be set to desired values and observed by shifting test patterns in and out. In large circuits however, this technique adversely affects the test application time as all the test vectors have to be scanned in and out of the flip-flops. Moreover, all of the memory elements may not be scanable in a given circuit [4]. In order to alleviate the test complexity, a second technique, called *partial-scan design*, is employed. This involves scanning a selected set of memory elements. Both these methods can add 10-20% hardware overhead. In case of a full scan design, a combinational test generator can be used to obtain tests. However, a sequential test generator is necessary in case of a partial scan or no-scan design [4]. The goal in this work is to use Genetic Algorithms (GAs) for generating sequences that will help the Automatic Test Pattern Generator (ATPG) in detecting more faults by reaching specific states. GAs are very well suited for optimization and search problems [5]. Several ATPGs have been reported which use genetic algorithms for simulation-based test generation. A good comparison is given in [3]. The main advantage of GA-based ATPGs, as compared to other approaches, is their ability to cover a larger search space in lower CPU time. This improves the fault coverage and makes these ATPGs capable of dealing with larger circuits. On the other hand, the main drawback consists in their inability to identify untestable faults [2]. Deterministic algorithms for combinational circuit test generation have proven to be more effective than genetic algorithms [6]. Higher fault coverages are obtained, and the execution time is significantly smaller. However, state justification using deterministic algorithms is a difficult problem, especially if design and tester constraints are considered [7]. In simulation-based ATPGs, the search proceeds in the forward direction only. Hence there are no backtracks

and state justification is easier as compared to deterministic ATPGs. In this work, a hybrid state justification approach is proposed, where both deterministic and genetic-based algorithms are employed. In evaluating this approach, we will conduct experiments in which a deterministic test generator will be employed initially. Untestable faults will be identified. The states which could not be reached in this phase, will be attempted in a genetic phase for state justification. Since Genetic Algorithms have been used successfully for combining useful portions of several candidate solutions to a given problem [5], we will try to genetically engineer sequences which justify the leftover states. In [8], Genetic Algorithms have been used for state justification. The length of the sequence was a function of the structural sequential depth of the circuit, where sequential depth is defined as the minimum number of flip-flops in a path between the primary inputs and the farthest gate. In case of feed-back loops, the structural sequential depth may not give a correct estimate of the number of vectors required for justifying a given state. Thus, if a state requires longer justification sequence, it will not be justified. The approach also does not take into account the quality of intermediate states reached and evaluates a chromosome only on the basis of the final state reached. In this work, we will use an incremental approach in which the length of the sequences will be dynamic. State justification sequences will be genetically engineered vector by vector. Even if some state remains unjustified after the genetic phase, the best sequence obtained in a given number of generations will be viewed as a partial solution. The determinisitc ATPG will be seeded with this sequence so that it may become able to reach previously unvisited regions of the search space. The remainder of this paper is organized as follows: Section 2 discusses application of genetic algorithms to sequential ATPG. In Section 3, genetic-based state justification is presented. Experimental results are given in Section 4. Section 5 concludes the paper.

2 Sequential ATPG and Genetic Algorithms

The goal of sequential circuit ATPG using the single stuck-at fault (SSF) model is to derive an input vector sequence such that, upon application of this input vector sequence, we obtain different output responses between the fault-free and faulty circuits. The SSF model is an abstraction of defects in a circuit which cause a single line connecting components to be permanently stuck either at logic 0 or logic 1 [9]. In this work, we assume the SSF model.

2.1 Complexity of sequential ATPG

Sequential ATPG is a much more complex process than combinational ATPG due to signal dependencies across multiple time frames [10]. It has been shown in [11] that the test generation problem for combinational circuits is NP-complete. The search space is of the order of 2^n, where 'n' is the number of inputs. For sequential circuit ATPG, the worst-case search space is 9^m, where m is the number of flip-flops. This exponential search space makes exhaustive ATPG search computationally impractical for large sequential circuits [4]. In the last years, one of the main goals of researchers was to develop effective algorithms for sequential circuit test pattern generation [12]. A lot of work has been done in the area of sequential circuit test generation using both deterministic and simulation based algorithms. The bottleneck in deterministic algorithms is line justification and backtracking. In simulation-based approaches, no backtracking is required but their quality in terms of fault coverage is generally lower [12]. It can however be improved with the help of GAs which are very well suited for optimization and search problems.

2.2 Using GA in Sequential ATPG

Genetic Algorithms work by analogy with Natural Selection as follows. First, a population pool of chromosomes is maintained. The chromosomes are strings of symbols or numbers. They might be as simple as strings of bits - the simplest type of strings possible. The chromosomes are also called the genotype (the coding of the solution). These chromosomes must be evaluated for fitness. Poor solutions are purged and small changes are made to existing solutions. The gene pool thus evolves steadily towards better solutions. In this work, we have used a Simple Genetic Algorithm as given in [13]. Several approaches to test generation using genetic algorithms have been proposed in the past [2], [6] - [8], [12], [14] - [21]. Fitness functions were used to guide the GA in finding a test vector or sequence that maximizes given objectives for a single fault or a group of faults. However, hard-to-test faults often could not be detected. GAs were used in different phases of the test generation process. In [15], [6] and [16], GA-based test generators were developed which used logic simulation for fitness evaluation. A fault simulator was used in [17] [18], and [19] for computing the fitness. The fitness functions were biased towards maximizing the number of faults detected and the number of fault effects propagated to the flip-flops. Several genetic parameters were experimented with in [17] and [20]. The fault coverage improved by more

than 40% for some benchmark circuits. These genetic-based ATPGs were however, not successful in propagating fault effects to the primary outputs. Moreover, they were unable to identify redundant faults. Hence, hybrid techniques were proposed in [7], [8], [12] [14] and [21].

3 Genetic-based State Justification

State justification is the most difficult task in sequential ATPG. Storing the complete state information for large circuits is impractical. Similarly, keeping a list of sequences capable of reaching each reachable state is also infeasible. State justification is therefore performed by using a GA. In [7] and [8], deterministic algorithms were used for fault excitation and propagation, and a GA was used for state justification. Sequences were evolved over several generations. The fitness of each individual was a measure of how closely the final state reached matched the desired state. A chromosome was represented by a sequence of vectors. Candidate sequences were simulated starting from the last state reached at the end of the previous test sequence. The objective was to engineer a test sequence that justified the required state. If a sequence was found which justified the required state, the sequence was added to the test set. In this work, we use GA for traversing from one state to another. Individual vectors are represented by chromosomes in the population and genetic operators are applied at individual bit positions. Deterministic ATPG is run for every target fault. First, the fault is activated and propagated to a primary output. Next, state justification is attempted. If the required state is justified by the deterministic ATPG, then the derived sequence is fault simulated and all detected faults are dropped from the faultlist. Otherwise, our GA-based algorithm attempts to justify the required state. A block diagram of the methodology is shown in Figure 1.

We have proposed an evolutionary meta-heuristic for the state justification phase. A flowchart of the heuristic used is shown in Figure 2.

3.1 Encoding of the chromosome

In our work, a chromosome represents a single vector. Each character of a chromosome in the population is mapped to a primary input. A binary encoding is used in this implementation.

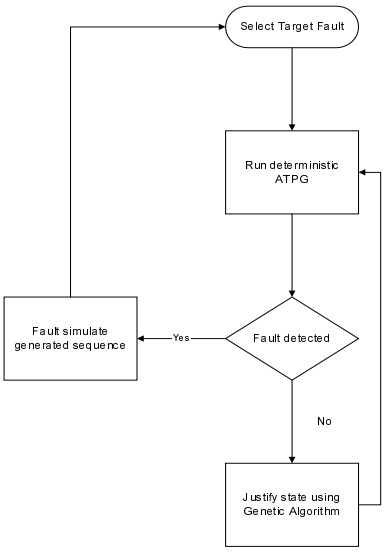

Figure 1: A block diagram of the methodology.

3.2 Fitness Function

Fitness function is the most important parameter of the GA. A solution is considered to be better than another if its fitness is higher. Each vector (chromosome) is logic simulated to give the state reached. This state is compared with all the flip-flop assignment values of the target state. The fitness $f(v_i)$ of a vector v_i is computed as follows:

$$f(v_i) = \frac{m(s_i,s_j)}{B(s_j)}$$

where s_i is the state reached by vector v_i, s_j is the target state and $m(s_i, s_j)$ are the number of matching specified bits in s_i and s_j. B(s_j) gives the number of specified bits in s_j (i.e. those which are not 'x').

3.3 Crossover and Mutation

One-point uniform crossover as mentioned in [22] has been used in this work. In one-point uniform crossover, an integer position is randomly selected within a chromosome. Each of the two parents are divided into two parts at this random cut point. The offspring is then generated by catenating the segment of one parent to the left of the cut point with the segment of the second parent to the right of the cut point. *Mutation* produces incremental changes in the offspring by randomly changing values of some genes. In this work, mutation corresponds to flipping a randomly selected bit.

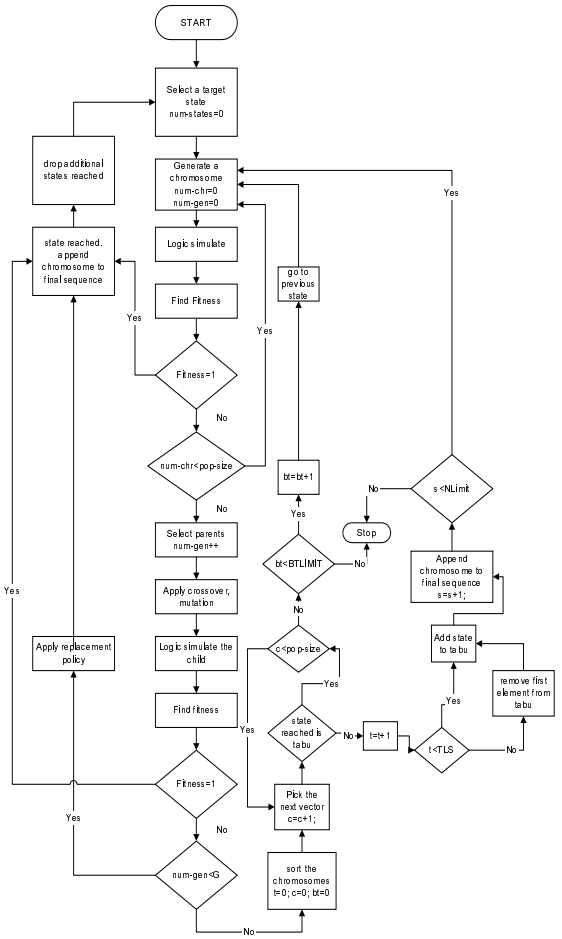

Figure 2: A flowchart of the algorithm used.

3.4 Forming a new generation

A generation is an iteration of GA where individuals in the current population are selected for crossover and offsprings are created. Due to the addition of offsprings, the size of population increases. In order to keep the number of members in a population fixed, a constant number of individuals are selected from this set for the new generation. The new population thus consists of both members from the initial generation and the offsprings created. In this work, we have used a one-change strategy as described below.

3.4.1 (n+1) selection strategy

In this strategy, we change one chromosome in every generation. One crossover is performed in every generation. If the child is more fit than the worst member of the previous generation, it is introduced into the population. Hence, we select the best n-1 members from a population of n, and the worst member gets replaced if its fitness is less than the fitness of the offspring.

3.5 Traversing from a state to a state

The algorithm is run for a fixed number of generations. If the state reached is the desired state, the algorithm stops and picks the next state from the list. However, if the algorithm is unable to reach the desired state, it picks the best chromosome found until then and adds it to the test set. Since the state reached is nearer in terms of the Hamming distance to the desired state, it is probable that it will help the ATPG in reaching the required search space and detecting the associated fault. The following parameters are used to guide the search.

3.5.1 Tabu List Size

To prevent the algorithm from visiting recently visited states, we propose a Tabu List containing the last visited states. The length of this list is a user-defined parameter. On reaching a state, the algorithm looks into the Tabu list. If the state reached is present, the next fit vector is chosen and its fitness is evaluated.

3.5.2 Backtrack limit

When all the chromosomes in the population are unable to reach a new state, (a state which is not in the Tabu List), we move to a previously visited state. This is termed as *backtracking*. We impose an upper limit on this parameter and the algorithm stops searching for a state when this parameter exceeds.

3.5.3 Nlimit parameter

The algorithm traverses *at least* Nlimit number of states before it gives up the search for the desired state. If the fitness of the currently visited state is less than the average fitness of the last Nlimit states, the algorithm stops further searching of the desired state; otherwise the search is continued.

3.6 Removing the reached states from the list of desired states

Once a sequence is generated by the algorithm, we compare the states reached by the sequence with the list of desired states. All the desired states reached by the sequence are removed. This prevents us from searching again for those states which we have already reached while searching for some other target state.

4 Experimental Results and Discussion

In this work, we have compared our state justification technique in which we use GA for traversing from a state to a state, with the one proposed in [7][8]. In [8], GA has been used in state justification and sequences are genetically engineered. GA has been applied on a sequence of vectors as opposed to individual vectors in our case. We have used five ISCAS89 benchmark circuits [23] and four re-timed circuits given in [10] for which which HITEC [24] requires very large CPU times. A list of target states was obtained for hard-to-detect faults in each of the circuits. We have experimented with several parameters and found that in general, a population size of 16, a generation limit of 400, backtrack limit of 10 and tabu list size of 15 gave the best results. Better results were obtained for an Nlimit value which was 1.5 times the number of flipflops present in the circuit. A roulette wheel selection scheme as given in [6] gave the best results. The weakest chromosome in the population was replaced by a new chromosome in every generation. Hence, the average and best fitness of the population monotonically increased in every generation as shown in Figure 3 for one of the reached target states. One-point crossover was used with a probability of 1 and mutation rate was kept at 0.01. In Figure 4, we show the state traversal for one of the states that has been reached by the algorithm. It can be seen that we progress towards better states in terms of the hamming distance as the algorithm runs for more iterations. Less fit states are reached if we we are unable to reach a better state because of the Tabu restriction. Moreover, we move towards the best state among all alternatives, even if that state is worse than the current state. This helps in avoiding the local minima. The example is for one of the target states of s1488 circuit.

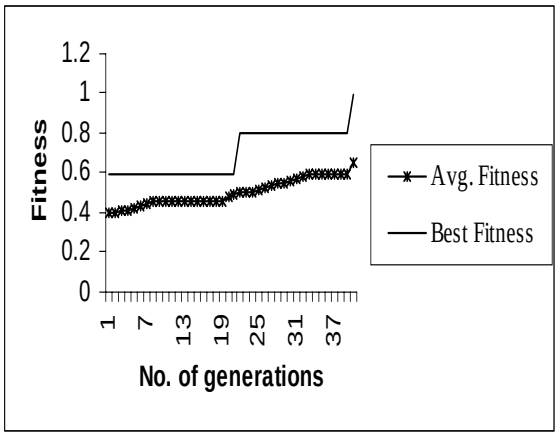

Figure 3: Average and best fitness vs. number of generations.

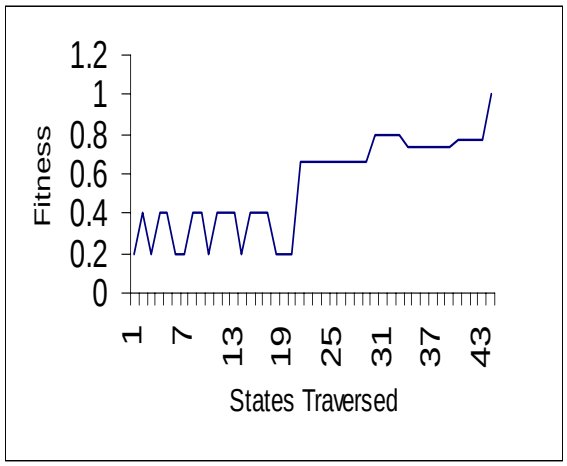

Figure 4: State traversed vs the fitness of reached states for a target state of s1488 circuit.

The parameters proposed in [8] were 32 chromosomes and 8 generations. The number of vectors in each chromosome was 4 times the sequential depth of the circuit.

To compute the fitness of chromosomes, we have used the logic simulator of HOPE [25]. The experiments were run on SUN ULTRA 10 stations and the results were obtained as shown in Table 1.

The first column in the table shows the circuit name. In the second and third columns, the number of flipflops (FFs) and the number of target states respectively is given for each circuit. The states reached and CPU time obtained by our algorithm are mentioned in the next two columns. For comparison purposes we ran the algorithm proposed in [8] for several number of generations and the results are shown in the next columns.

It can be observed from the results that the number of desired states reached by our technique are more than those reached by the technique used in [8] for all the circuits. Furthermore, our proposed technique reached a higher number of states than [8] in 8 out of 9 cases even when the latter was run for greater amount of CPU time.

5 Conclusion

In this work, we have proposed a new state justification technique based on GA which engineers the sequence

			our approach		approach in [8]			approach in [8]		
Name	# of FF	Target states	states reached	time(sec)	gens	states reached	time(sec)	gens	states reached	time(sec)
s1423	74	135	61	335	8	50	2743	50	61	3953
s3271	116	45	20	1229	8	15	1664	100	18	2390
s3384	183	102	56	8124	8	31	3794	200	42	16411
s5378	179	524	113	29274	8	45	3133	100	48	225160
s6669	239	32	30	1664	8	23	1701	50	24	2289
scfRjisdre	20	267	48	803	8	25	501	100	31	5196
s832jcsrre	31	57	8	139	8	7	120	100	7	2170
s510Rjcsrre	30	114	16	163	8	12	61	100	13	504
s510Rjosrre	32	114	16	181	8	9	62	100	13	583

Table 1: Comparison of the two techniques

vector by vector. This is in contrast to previous approaches where GA is applied to the whole sequence. The drawback of previous approaches lies in their inability to justify hard-to-reach states because of fixed-length sequences. Moreover, they do not take into account the quality of intermediate states reached and evaluate a chromosome only on the basis of the final state reached. We propose dynamic length sequences in this work and the fitness measure takes into account all the states reached by the sequence. The approach has been compared with previous approaches and improvements in reached states and CPU time have been demonstrated.

Acknowledgment: The authors thank King Fahd University of Petroleum and Minerals, Dhahran, Saudi Arabia, for support.

References

[1] Aiman El-Maleh, Mark Kassab, and Janusz Rajski. A Fast simulation-based learning technique for real sequential circuits. In *Design Automation Conference*, 1998.

[2] F.Corno, P.Prinetto, M.Rebaudengo, and Sonza Reorda. A Parallel Genetic Algorithm for aoutomatic generation of test sequences for digital circuits. In *International Conf. on High Performance Computing and Networking, Belgium*, April 1996.

[3] Y.C.Kim and K.K.Saluja. Sequential test generators: past, present and future. *INTEGRATION, the VLSI journal*, 26:41–54, 1998.

[4] M.H.Konijnenburg, J.T. van der Linden, and A.J. van de Goor. Sequential test generation with advanced illegal state search. In *International Test Conference*, 1997.

[5] Srinivas M. and L.M.Patnaik. Genetic Algorithms: A Survey. *IEEE Computer Magazine*, pages 17–26, June 1994.

[6] E.M.Rudnick, J.Holm, D.G.Saab, and J.H.Patel. Application of Simple Genetic Algorithm to sequential circuit test generation. In *European Design and Test Conference*, pages 40–45, February 1994.

[7] Elizabeth M. Rudnick and Janak H. Patel. State justification using Genetic Algorithms in sequential circuit test generation. *A survey report from CRHC Univ. of Illinois, Urbana*, January 1996.

[8] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Application of genetically engineered finite-state-machine sequences to sequential circuit ATPG. *IEEE Transactions on CAD of Integrated circuits and systems*, 17:239–254, March 1998.

[9] Xinghao Chen and M.L.Bushnell. Sequential circuit test generation using dynamic justification equivalence. *Journal of Electronic Testing*, 8:9–33, 1996.

[10] T.E.Marchok, Aiman El-Maleh, W.Maly, and J.Rajski. A complexity analysis of sequential ATPG. *IEEE Transactions on CAD of Integrated circuits and systems*, 15:1409–1423, November 1998.

[11] P.H.Ibarra and S.K.Sahni. Polynomially complete fault detection problems. *IEEE Transactions on Computing*, 24:242–249, 1975.

[12] Michael S. Hsiao. Use of Genetic Algorithms for testing sequential circuits. *Ph.D. Dissertation, UIUC*, December 1997.

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

[14] F.Corno, P.Prinetto, M.Rebaudengo, Sonza Reorda, and M.Violante. Exploiting logic simulation to improve simulation-based sequential ATPG. In *Sixth IEEE Asian Test Symposium, Arta, Japan*, November 1997.

[15] D.G.Saab, Y.G.Saab, and J.A.Abraham. CRIS: A test cultivation program for sequential VLSI circuits. In *International Conf. on Computer-aided Design*, pages 216–219, November 1992.

[16] F.Corno, P.Prinetto, M.Rebaudengo, and Sonza Reorda. GATTO: A genetic algorithm for automatic test pattern generation for large synchronous sequential circuits. *IEEE Transactions on CAD of Integrated circuits and systems*, 15:991–1000, August 1996.

[17] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Alternating strategies for sequential circuit ATPG. In *European Design and Test Conference*, pages 368–374, March 1996.

[18] E.M.Rudnick, J.H.Patel, G.S.Greenstein, and T.M.Niermann. A Genetic algorithm framework for test generation. *IEEE Transactions on CAD of Integrated circuits and systems*, 16:1034–1044, September 1997.

[19] E.M.Rudnick, J.H.Patel, G.S.Greenstein, and T.M.Niermann. Sequential circuit test generation in a genetic algorithm framework. In *Design Automation Conference*, pages 698–704, June 1994.

[20] D.G.Saab, Y.G.Saab, and J.A.Abraham. Automatic test vector cultivation for sequential VLSI circuits using genetic algorithms. *IEEE Transactions on CAD*, 15:1278–1285, October 1996.

[21] M.S.Hsiao, E.M.Rudnick, and J.H.Patel. Sequential circuit test generation using dynamic state traversal. In *European Design and Test Conference*, pages 22–28, March 1997.

[22] Sadiq M.Sait and Habib Youssef. *Iterative Computer Algorithms wirh applications in Engineering: Solving combinatorial optimization problems.* IEEE Computer Society, 1999.

[23] F.Brglez, D.Bryan, and K.Kozminski. Combinational profiles of sequential benchmark circuits. *International Symposium on Circuits and Systems*, pages 1929–19347, 1989.

[24] T.M.Niermann and J.H.Patel. HITEC: A test generation package for sequential circuits. In *European Conference on Design Automation*, pages 214–218, 1991.

[25] H. K. Lee and D. S. Ha. HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1048–1058, September 1996.