An Evolutionary Meta-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

Sequential circuit test generation using deterministic, fault-oriented algorithms is highly complex and time consuming. New approaches are needed to enhance the existing techniques, both to reduce execution time and improve fault coverage. Evolutionary algorithms have been effective in solving many search and optimization problems. A common search operation in sequential ATPG is to justify a desired state assignment on the sequential elements. State justification using deterministic algorithms is a difficult problem and is prone to many backtracks, which can lead to high execution times. In this work, we propose a hybrid approach which uses a combination of evolutionary and deterministic algorithms for state justification. A new method based on Genetic Algorithms is proposed, in which we engineer state justification sequences vector by vector. This is in contrast to previous approaches where GA is applied to the whole sequence. 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. Furthermore, it is demonstrated that the state-justification sequence generated, helps the ATPG in detecting a large number of hard-to-detect faults.

1 Introduction

Testing of integrated circuits is an important area which nowadays accounts for a significant percentage of the total design and production costs of ICs. Various design for testability (DFT) techniques are used to obtain acceptably high quality tests [1]. In the first technique, called *full-scan design*, 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. In order to alleviate the test complexity, a second technique, called *partial-scan design*, is used. This involves scanning a selected set of memory elements. A sequential test generator is necessary for partial or no-scan designs. 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 [2]. In the last years, one of the main goals of researchers was to develop effective algorithms for sequential circuit test pattern generation [3]. Both deterministic and simulation-based algorithms have been used. The bottleneck in deterministic algorithms is line justification and backtracking. In simulation-based approaches, the search proceeds in the forward direction only. Hence, there are no backtracks and state justification is easier as compared to deterministic ATPGs. The main drawback of simulation-based approaches lies in their inability to detect untestable faults. In this work, a hybrid state justification approach is proposed, where both deterministic and genetic-based algorithms are used. Several approaches to test generation using genetic algorithms have been proposed in the past [3] - [8]. A major difference in various GA-based approaches lies in the way the fitness is computed. Some techniques use logic simulation for evaluation of candidate vectors or sequences, while other techniques use fault simulation. In addition, there are certain other techniques which target different objectives in various phases of test generation. These techniques typically, use both logic and fault simulation in evaluating candidate sequences. Genetic Algorithms have been used for state justification in [5]. 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,

0-7803-7044-9/01/$10.00 ©2001 IEEE

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: In Section 2, the proposed genetic-based state justification technique is presented. Experimental results are given in Section 3. Section 4 concludes the paper.

2 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 [5] and [9], 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 fault list. Otherwise, our GA-based algorithm attempts to justify the required state. If the state remains unjustified, the generated sequence is still saved, as the reached state is close to the required state, which could help the ATPG in justifying it in the next attempts. The generated sequence is then fault simulated and the detected faults are dropped. A block diagram of the methodology is shown in Figure 1.

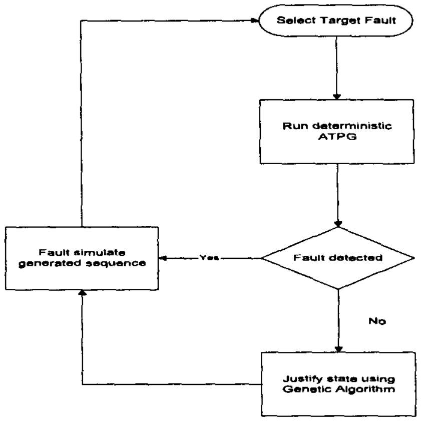
Select Target Fault
Run deterministic ATPG
Fault detected
Yes
No
Fault simulate generated sequence
Justify state using Genetic Algorithm

Figure 1: A block diagram of the methodology.

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

2.1 Encoding of the chromosome

In this work, a binary encoding is used. A chromosome is represented by a single vector. Each bit of a vector corresponds to the value at a primary input.

2.2 Fitness Function

In Genetic Algorithms, a solution is considered to be better than another if its fitness is higher. We logic simulate each vector to get 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').

2.3 Crossover and Mutation

We use One-point uniform crossover as mentioned in [10] in this work. In one-point uniform crossover, a random cut-point is selected. Each of the two parents are divided into two parts at this random cut point. We generate an offspring by catenating the segment of one

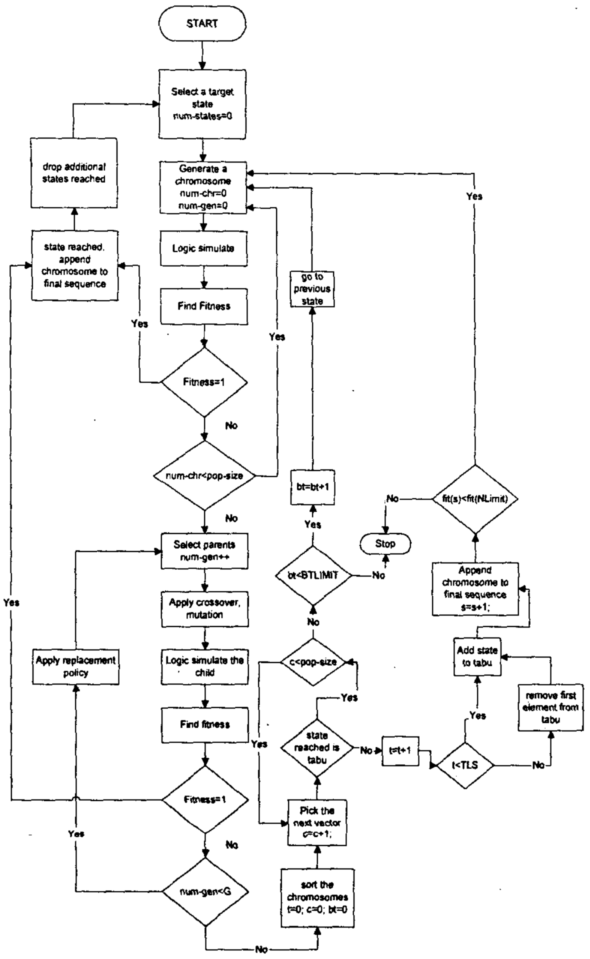

Figure 2: A flowchart of the algorithm used.

parent to the left of the cut point with the segment of the second parent to the right of the cut point. *Mutation* introduces new characteristics in the offspring by randomly changing values of some genes. In this work, mutation corresponds to flipping a randomly selected bit.

2.4 Forming a new generation

A generation corresponds to an iteration of GA where parents are selected for crossover and offsprings are created. A constant number of individuals are selected from the offsprings for the new generation. The new population thus consists of both members from the initial generation and the offsprings created. In this work, three replacement strategies have been experimented with:

2.4.1 (n+1) selection strategy. In this strategy, one chromosome is changed in every generation. A crossover is performed on two selected parents. A child replaces the worst member of the previous generation, if its fitness is higher. Hence, the best n-1 members are selected for the new generation from a population of n.

2.4.2 Random Elitist strategy. We produce n offsprings, by performing $n/2$ crossovers on a population of n chromosomes. Best $n/2$ members of both the offsprings and the original population, are transferred to the next generation. The remaining $n/2$ members of the new generation are selected randomly from the leftover chromosomes.

2.4.3 Roulette Elitist strategy. This strategy is the same as Random Elitist strategy except that the second half of the members of the new generation are selected based on a roulette wheel mechanism. This gives an advantage to the relatively more fit members of the population to be transferred to the next generation.

2.5 Traversing from a state to a state

We run the algorithm for a fixed number of generations. If we reach 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 state justification sequence. As the state reached is nearer to the desired state in terms of the Hamming distance, it is probable that it will help the ATPG in reaching the required search space and detecting the associated fault. The following parameters have been used to guide the search.

2.5.1 Tabu List Size. A Tabu List is used to prevent the algorithm from visiting recently visited states. On reaching a state, the algorithm looks into the Tabu list. If the state reached is present, the next fit vector is chosen.

2.5.2 Backtrack limit. We backtrack to the last visited state, when all the chromosomes in the population are unable to reach a new state (a state which is not in the Tabu List). An upper limit is imposed on the number of backtracks and the algorithm stops searching for a state when this limit is exceeded.

2.5.3 Nlimit parameter. At least Nlimit number of states are traversed before the algorithm gives up the search for a desired state. If the fitness of the currently visited state, $fit(s)$, is less than the average fitness of the last Nlimit states, $fit(Nlimit)$, the algorithm stops further searching of the desired state; otherwise the search is continued.

2.6 Removing the reached states from the list of desired states

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

3 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 [5][9]. In [5], 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 [11] and four re-timed circuits given in [12] for which which HITEC [13] requires very large CPU times. A list of target states was obtained for hard-to-detect faults in each of the circuits (those faults which HITEC aborted after 1000000 backtracks). We have experimented with several parameters and found that in general, a population size of 32, 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 [4] gave the best results. Three replacement policies were experimented with. The results of the simulations carried out using these three replacement policies are shown in Table 1.

In Table 1, the number of states reached (SR) and the time taken to reach those states are given for each replacement strategy described above. It was observed that the (n+1) replacement strategy was the best in terms of execution time. It also reached a comparable number of states for most of the circuits. This strategy changes only one member of the previous generation and hence the number of operations in one generation of (n+1) replacement strategy requires less time as compared to other strategies. Moreover, changes in the characteristics of the population do not occur as abruptly as in the other two schemes. Figure 3 shows the average and best fitness of the population against the number of generations for one of the target states that is justified by the algorithm using the (n+1) replacement strategy. It can be seen that the average fitness increases monotonically with the number of generations. This is due to the fact that we are always preserving the best chromosome in each generation. 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 target states of the circuit s1423 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 s1423 circuit.

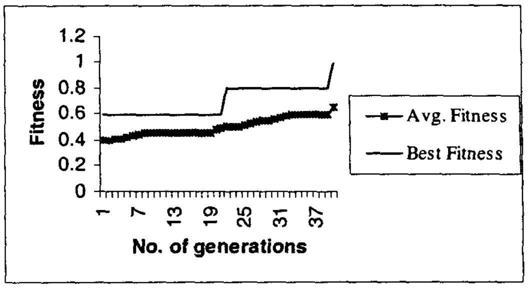

Figure 3: Average and best fitness vs. number of generations.

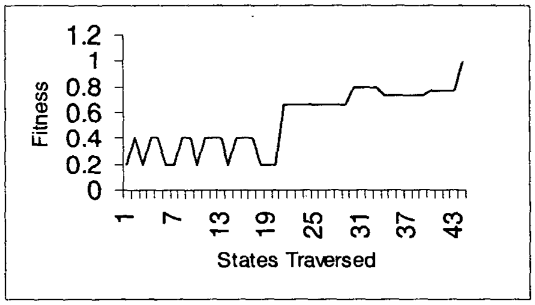

Figure 4: State traversed vs the fitness of reached states for a target state of s1423 circuit.

The parameters proposed in [5] 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 HOPE [14]. The experiments were run on SUN ULTRA 10 stations and the results of comparing the two state justification techniques are shown

circuit	CHR	GEN	BT	NLimit	TLS	(n+1)		Random Elitist		Roulette Elitist	
						SR	Time	SR	Time	SR	Time
s1423	16	100	10	120	150	58	126	19	508	32	748
	32	100	10	120	150	64	365	31	778	49	3586
	64	100	10	120	150	64	572	49	11300	68	13704
s3271	16	800	20	225	150	20	4592	11	5023	15	11214
	32	100	20	225	150	21	6244	18	11805	20	18113
	256	100	20	225	150	21	10625	19	12976	21	109612
s3384	16	800	10	375	150	65	11849	23	14912	34	17445
	64	800	10	375	150	66	23115	51	24905	41	30023
	256	800	10	375	150	66	41225	65	68428	50	100615
s5378	16	400	10	275	150	64	25294	22	84225	45	112610
	32	400	10	275	150	113	29274	53	100324	61	141251
	64	400	10	275	150	115	34893	55	117520	61	161225
s6669	16	10	10	375	150	19	130	19	871	22	914
	16	100	10	375	150	27	503	19	5151	22	8681
	16	400	10	375	150	30	1664	22	17905	22	24668
scfRjisdre	16	100	10	40	150	18	25	17	285	26	836
	64	100	10	40	150	19	42	34	832	43	6700
	256	100	10	40	150	20	114	46	5055	50	48820
s832jcsrre	16	400	100	100	150	7	79	6	77	6	82
	256	400	100	100	150	7	190	7	1946	7	2126
	1024	400	100	100	150	9	360	8	3441	9	4956
s510Rjcsrre	16	400	10	45	150	12	14	8	120	6	140
	256	400	10	45	150	16	132	23	523	23	1208
	512	400	10	45	150	23	260	31	2340	31	5038
s510Rjosrre	16	800	10	45	150	12	92	5	233	4	305
	64	800	10	45	150	19	661	13	1171	11	2841
	256	800	10	45	150	19	2740	17	9870	19	19342

Table 1: Comparison of the selection schemes

			our approach		approach in [5]			approach in [5]		
Name	# of FF	Target states	states reached	time(sec)	gens	states reached	time(sec)	gens	·states reached	time(sec)
s1423	74	135	74	3119	8	50	2743	50	61	3953
s3271	116	45	21	6015	8	15	1664	200	18	6319
s3384	183	102	67	18314	8	31	3794	250	45	21161
s5378	179	524	115	31281	8	45	3133	100	48	225160
s6669	239	32	30	1764	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 2: Comparison of the two state-justification techniques

in Table 2.

The first column in the table shows the circuit name. In the second and third columns, the number of flip-flops (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 [5] 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 [5] for all the circuits. Furthermore, our proposed technique reached a higher number of states than [5] in all the circuits even when the latter was run for a greater amount of CPU time.

In order to verify the effectiveness of the generated state justification sequences in detecting hard-to-detect faults, we seeded them to a deterministic test pattern generator HITEC [13]. HITEC makes use of previously visited states while doing state justification. The faults detected by an initial run of HITEC with 1000000 backtracks were removed from the fault list. The results are shown in Table 3.

It can be observed that a large number of hard-to-detect faults are detected when we seed HITEC with the state justification sequence obtained by the proposed strategy. The number of faults detected are

		faults detected			reached states	
Name	TF	approach in [5]	our approach	TS	approach in [5]	our approach
s1423	926	312	578	135	61	74
s3271	61	34	41	45	18	21
s3384	376	91	116	102	45	67
s5378	1221	103	285	524	48	115
s6669	40	29	31	32	24	30
scfRjisdre	1920	1397	1802	267	31	48
s832jcsrre	293	38	147	57	7	8
s510Rjcsrre	374	45	85	114	13	16
s510Rjosrre	459	232	431	114	13	16

Table 3: Faults detected by the two state-justification techniques

significantly higher than the faults detected when the ATPG is seeded with the state justification sequences generated by the technique proposed in [5]. Apart from justifying more states, our technique takes advantage of the partial justification sequences generated.

4 Conclusion

In this work, we have proposed a hybrid approach which uses a combination of evolutionary and deterministic algorithms for state justification. Genetic Algorithms (GAs) were used for generating sequences that will help the Automatic Test Pattern Generator (ATPG) in detecting more faults by reaching specific states. A new state justification technique based on GA is proposed which engineers the sequence vector by vector. In previous approaches, GA has been applied to the whole sequence. The previous approaches fail to justify many hard-to-reach states because of fixed-length sequences. Moreover, they evaluate a chromosome only on the basis of the final state reached. In this work, we propose dynamic length sequences 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 fault coverage have been demonstrated.

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

References

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

[2] 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.

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

[4] 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.

[5] 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.

[6] 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.

[7] 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.

[8] F.Corno, M.Rebaudengo, and Sonza Reorda. Experiences in the use of evolutionary techniques for testing digital circuits. In *Application and Science of Neural Networks, Fuzzy Systems and Evolutionary computation, SPIE*, 1998.

[9] 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.

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

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

[12] 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.

[13] 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.

[14] 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.