Enhancing GA-based Sequential ATPG through Guided Crossover

MICHAEL DIMOPOULOS, PANAGIOTIS LINARDIS
Department of Informatics
Aristotle University of Thessaliniki
GR-54006 Thessaloniki
GREECE

Abstract: Test Generation for digital circuits using deterministic methods is an NP-complete problem and for that reason Genetic Algorithms have been recently investigated as an alternative to test generation. In this paper a Genetic Algorithm "GATPG" is presented for generating test sequences for sequential circuits. The aim is to produce compact test sequences that attain high fault coverage. In order to fulfil these requirements a non-uniform selection probability for crossover is employed combined with an aging factor of variable-length sequences and a two-phase fitness function. Candidate test sequence evaluation is accomplished with a 3-valued fault simulator, allowing the circuit to start from an arbitrary (unknown) state. Experimental results with respect to the ISCAS'89 benchmarks are presented to show the viability of the proposed approach.

Key-Words: Genetic Algorithms, Test Generation, Sequential Digital Circuits, ATPG.

1 Introduction

Test generation of combinational digital circuits is a highly complex problem, typically it is NP-complete [3]. For sequential digital circuits the testing problem, which is the subject of this paper, is even more complex.

Sequential circuits, for testing purposes, are usually modeled as iterative arrays of combinational circuits and test generation techniques developed for combinational circuits are extended to handle sequential circuits. Mainly two approaches to testing are followed:

- Deterministic methods [3, 4, 6] that use branch and bound techniques with the aid of heuristics to prune the search space. Due to the vast search space these techniques are often unable to handle large sequential circuits[3, 12].
- Simulation-based methods [3] which are *trial-and-error* methods. They generate random test vectors which are evaluated by fault simulation according to a "cost" function. Best *trial* vector is selected and added to the test sequence.

In the "simulation" class of methods belong the Genetic Algorithm (GA) simulation based techniques [5, 9, 10, 11, 12]. In GA, initialy random test sequences guided by genetic operators [1, 2] evolve to highly fit solutions.

Recently, deterministic methods combined with GAs were devised that achieve very good results [7, 8].

In this paper we propose a GA simulation-based method, called GATPG, which contains a two phase fitness function and puts emphasis on shorter, more compact, test sequences by introducing a non-uniform crossover operator.

The paper is organized as follows: In section 2 is presented the testing problem for sequential circuits. In section 3 the structure of the GATPG algorithm is analyzed. In section 4 experimental results are given, supporting the potential of the proposed method.

2 Problem Formulation

A synchronous sequential circuit can be considered to be a Finite State Machine M defined as a quintuple $M=(I,O,S,\delta,\lambda)$ where I is the set of input vectors, O is the output set, S is the set of states, δ is the next state function and λ is the output function. A stuck-at fault f transforms the machine M into a machine $M_f=(I,O_f,S_f,\delta_f,\lambda_f)$. For a given list of stuck-at faults $F=\{f_1,f_2,\ldots,f_n)$ the test generation problem is to find a sequence of input vectors V, called *Test Sequence*, that detects the faults in F, that is when V is applied to each M_f will produce different responses from M.

It is assumed, here, that the initial state of M and M_f is unknown, as is usually the case.

In this work the test sequences V are generated with the help of genetic algorithms.

3 GATPG ALGORITHM

The algorithm is shown in Fig.1. Starting with an initial population of randomly produced sequences of test vectors (Create_random_population) each sequence (individual) is evaluated (Evaluate_fsim) by performing fault simulation to find the faults which are *detected*, propagated to flip-flops (*activated*), and state and output differences between M and M_f . These results are used to determine the fitness value of each sequence, as will be explained in section 3.2. The crossover operation (cross_over) applied here follows a non-uniform probability of candidate selection.

3.1 Individuals

An individual is a binary-coded 2-dimensional bit string corresponding to a sequence of input vectors. The length of this bit sequence is L=ninputs*no_of_test_vectors.

A characteristic of the GATPG algorithm is that it uses variable-length sequences with respect to the number of test vectors. Starting with an initial sequence length of 5 vectors, sequence length progressively increases every 3 generations by adding at the end one randomly generated vector. This approach has two advantages: lower simulation cost and more efficient test sequences.

The simulator, developed by the authors, is a PROOFS-based [4] 3-valued fault simulator. The third value X emulates the unknown initial state of the circuit.

3.2 Fitness Function

The results from the simulation are used to rank the individuals according to certain evaluation rules, which form the so called fitness function. Our fitness function is complex and has the form:

$$fitness = \begin{cases} f_1 & \text{iff } (\text{ngen/MAX\_GENERATIONS}) \le 0.25 \\ f_2 & \text{else} \end{cases}$$

where:

$$f_1 = 20 \cdot R_1 + R_3 \cdot R_2$$

$$f_2 = 20 \cdot R_1 + R_3 + R_4 \cdot R_5 \cdot R_2$$

where: R_i is a value denoting how "close" the individual is to satisfying rule i.

```
Create_random_population.
For each individual
Evaluate_fsim(individual);
endFor
Sort_Population(); /* with fit. value descending*/

ngen=0;      /*  num of generation */
do {
/******** crossover *********/
for (j=0,i=0; i < ncross; j +=2, i++)
{
cross_over(Individual[j], Individual[j+1],
child1, child2);
Evaluate_fsim(child1);
Evaluate_fsim(child2);
update_age_of(child1);
update_age_of(child2);
}
/******** mutations *********/
for (i=0; i < nmut; i +=2)
{
mutation(Individual[0],  child);
mutation(Individual[1], child1);
Evaluate_fsim(child);
Evaluate_fsim(child1);
update_age_of(child1);
update_age_of(child2);
}
Sort_Population();

If ((ngen % 3) == 0)
{
Expand_sequence(EXPAND_STEP);
Evaluate_fsim(Individual[0]); /*check best*/
}
ngen++;
} while (ngen < MAX_GENERATIONS);
```

Fig.1 The GATPG algorithm

The rules that every individual should obey are:

$$R_1 = f\,detected$$

$$R_2 = \frac{seq.len - eff.len}{seq.len}$$

$$R_3 = \frac{factivated}{fremain + 1}$$

$$R_4 = \frac{C_{FF}}{numFF \cdot factive \cdot seq.len}$$

$$R_5 = \frac{C_{OUT}}{noutputs \cdot factive \cdot seq.len}$$

where:

$$C_{FF} = \sum_{k=1}^{seq.len} \sum_{j=1}^{TotalFaults} \sum_{i=1}^{numFF} G_{ij}$$

$$C_{OUT} = \sum_{k=1}^{seq.len} \sum_{j=1}^{TotalFaults} \sum_{i=1}^{noutputs} G_{ij}$$

$$G_{ij} = \begin{cases} 1 \text{ iff } & \text{ckt}_i^{\text{good}} \neq ckt_i^{f_j} \\ 0 & \text{else} \end{cases}$$

In this fitness function emphasis is given in the maximization of detected faults while favouring smaller test sequences. A two-phase function is used. Because in practice we have "easy" and "difficult" to test faults [5] we start with f_1 and after a number of generations switch to a different function f_2. As was mentioned, test sequences are extended every three generations by appending a new randomly generated vector. In order to escape from stagnation an aging factor is incorporated so that offsprings having the same fitness value with their parents are given higher precedence in the next generation.

3.3 Genetic Operators

The creation of an offspring is accomplished with the help of a crossover operator that interchanges the bits of two individuals. The crossover operation applied here is a one-point crossover [1, 2].

Two crossover operators are used:

- Standard cut-point selection is used with uniform probability.
- A square root probability distribution function is used to direct the cut-point selection towards the end of the test sequence, thus giving emphasis on optimizing the tail of the sequence as new vectors are appended to it.

To ensure diversity, mutation is applied to the best 2 individuals in the population. Two different mutation operations are used:

- Single-bit mutation: it randomly selects a bit and complements it.
- Multi-bit-mutation: it randomly selects a vector and for every bit within it a choice is made with a probability of ½ whether to keep its value or to complement it.

4 Experimental Results

The efficiency of the GATPG algorithm, implemented in C, was measured by using some of the ISCAS'89 benchmark circuits [13]. The main characteristics of these benchmark circuits are given in Table 1, where *i*, *o*, *ff*, *gates* denote the number of inputs, outputs, flip-flops, gates. *Total detected* are the number of faults that can be detected and against which the results are judged.

a/a	circuit	i / o / ff / gates	Faults	Total Detected
1	s298	3 / 6 / 14 / 119	308	265
2	s344	9 / 11 / 15 / 160	342	329
3	s349	9 / 11 / 15 / 161	350	335
4	s382	3 / 6 / 21 / 158	399	364
5	s386	7 / 7 / 6 / 159	384	314
6	s400	3 / 6 / 21 / 164	426	384
7	s444	3 / 6 / 21 / 181	474	424

Table 1. ISCAS'89 circuits

For the GATPG we used the following parameter values.

POPULATION = 16
MAX_GENERATIONS = 300
PCROSSOVER = 0.6
PMUTATION = 0.2
EXPAND_STEP = 1

In Tables 2 and 3 we present results regarding uniform and square-root (sqrt) probability of cut-point selection. *Det.*, *Vec.* and *Time* represent the number of detected faults, of test sequence length and of the generations required to achieve these sequences.

Circuit	Det.	Vec.	Time (gen)
s298	264	79	242
s344	327	56	266
s349	332	51	295
s382	316	88	284
s386	254	39	284
s400	329	86	293
s444	360	91	258
Sum	2182	490	1922

Table 2. Uniform selection probability

Circuit	Det.	Vec.	Time (gen)
s298	265	93	292
s344	329	64	232
s349	335	65	295
s382	323	94	276
s386	275	57	238
s400	337	85	232
s444	375	85	278
Sum	2239	543	1843

Table 3. Sqrt selection probability

As we see from the above results sqrt selection probability (Table 3) is better than uniform selection probability (Table 2) because, in order of importance, on the average: (a) it detects more faults, (b) in relatively small sequences and (c) in shorter time.

In Table 4 we compare our results with results from [5, 6], were *f.c* and *Vec.* are the fault coverage (detected faults to total detectable faults) and the test sequence length.

As we see in Table 4 our results (sqrt, uniform) regarding fault coverage for the first three circuits are nearly the same with those of the others.

For the remaining circuits although our fault coverage is lower (average fault cov. 0.932 (sqrt) compared to 0.972 [5]) the size of our test sequences is 2.5 times smaller than [5] and 22.8 times smaller than HITEC.

We must note that HITEC is a state-of-the art deterministic test pattern generator that achieves high fault coverage but requires long CPU time to achieve satisfactory results. The method of [5] belongs to the same category with our method. There is no comparison with methods from [10, 12] because they assume that the circuit starts from a given initial state instead of the more general case of an unknown (arbitrary) one.

	sqrt		uniform		HITEC [5,6]		[5]	
Circuit	f.c	Vec.	f.c	Vec.	f.c	Vec.	f.c	Vec.
s298	1,000	93	0,996	79	1,000	306	1,000	161
s344	1,000	64	0,994	56	0,997	142	1,000	95
s349	1,000	65	0,991	51	1,000	137	1,000	95
s382	0,887	94	0,868	88	0,997	4931	0,953	281
s386	0,876	57	0,809	39	1,000	311	0,939	154
s400	0,878	85	0,857	86	0,997	4309	0,951	280
s444	0,884	85	0,849	91	0,976	2240	0,958	275
fault cov.	0,932		0,909		0,995		0,972	
test seq.		543		490		12376		1341

Table 4. Comparison with results from [5, 6].

4 Conclusion

A GA-based test generation algorithm is presented which has some unique features. Apart from the fitness function used here and the aging of individuals, crossover is enhanced with a non-uniform crossover-selection probability.

This "directed" cut-point selection in crossover performs better than the classical one with uniform probability cut-point selection as is evident from experimental results presented here.

Although the preliminary results that were presented are quite competitive with those of others the GATPG algorithm may be further improved by adding more circuit specific knowledge in the fitness function and elaborating on GA-operators.

References:

[1] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*, Reading, MA: Addisson-Wesley, 1989.

[2] Zbigniew Michalewicz, *Genetic Algorithms+ Data Structures=Evolution Programs*, Springer, 1996.

[3] M. Abramovici, M. Breuer, A. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[4] T. M. Niermann, W. T. Cheng, and J. H. Patel, *PROOFS: A fast, memory-efficient sequential circuit fault simulator*, IEEE Trans. Computer-Aided Design, 1992, pp. 198-207.

[5] E. M. Rudnick, J. H. Patel, G. S. Greenstein, and T. M. Niermann, *Sequential circuit test generation in a genetic algorithm framework*, Proc. Design Automation Conf., 1994, pp. 698-704.

[6] T. M. Niermann and J. H. Patel, *HITEC: A test generation package for sequential circuits*, Proceedings of the European Conference on Design Automation, 1991, pp. 214-218.

[7] E. Rudnick, J. Patel, *Combining deterministic and genetic approaches for sequential circuit test generation*, DAC.,1995, pp. 183-188.

[8] M.H.Hsiao, E.M.Rudnick, J.H.Patel, *Alternating strategies for sequential circuit atpg*, European Design &Test Conf.,1996,pp. 368-374.

[9] D. G. Saab, Y. G. Saab, J. A. Abraham, *CRIS: A Test cultivation program for sequential VLSI circuits*, ICCAD, 1992, pp 216-219.

[10] F. Corno, P. Prinetto, M. Rebaudengo, M. Sonza Reorda, *GATTO: A Genetic Algorithm for Automatic Test Pattern Generation for Large Synchronous Sequential Circuits*, IEEE Trans. on CAD, Vol. 15, No 8, 1996, pp. 991-1000.

[11] M. Hsiao, E. Rudnick, J. Patel, *Sequential Circuit Test Generation Using Dynamic State Traversal*, European Design & Test Conf., 1997, pp. 22-28.

[12] F. Corno, P. Prineto, M. Rebaudengo, M. Sonza Reorda, R. Mosca, Advanced Techniques for GA-based sequential ATPGs, European Design & Test Conf., 1996.
[13] F. Brglez, D. Bryan and K. Kozminski, *Combinational profiles of sequential benchmark circuits*, Int. Symposium on Circuits and Systems, 1989, pp. 1929-1934.