International Journal of Emerging Engineering Research and Technology
Volume 3, Issue 9, September, 2015, PP 18-24
ISSN 2349-4395 (Print) & ISSN 2349-4409 (Online)

Design and Implementation of Genetic Algorithm as a Stimulus Generator for Memory Verification

M. Avinash
GRIET, Hyderabad

ABSTRACT

The paper describes an approach for the generation of deterministic test pattern generator logic. This approach employs a genetic algorithm that searches for an acceptable practical solution in a large space of implementation. Its effectiveness (in terms of result quality and CPU time requirement) for circuits previously unmanageable is illustrated. The flexibility of the new approach enables users to easily trade off fault coverage and CPU time to suit their needs.

Keywords: Test Pattern Generation (TPG), Genetic Algorithm (GA), Simulation Based Approaches, Switch Level Faults, Evolutionary Algorithm.

INTRODUCTION

Darwin"s principle *"Survival of the fittest"* captured the popular imagination. This principle can be used as a starting point in introducing evolutionary computation [5]. Due to the growing complexity of modern integrated circuits and increasing testing demands, boundary-scan approach has been developed and is widely adopted in practice [9]. A limited number of input/output pins represent a bottleneck for testing of complex embedded cores where apply large amounts of test patterns and test results between the automatic test equipment and the circuits under-test are required.

The most popular technique in evolutionary computation research has been the genetic algorithm. A genetic algorithm is a stochastic optimization technique inspired by the principles of evolution. John Holland proposed the first simulated evolution algorithm that mimicked the evolutionary process in nature. In the traditional genetic algorithm, the representation used is a fixed-length bit string. Each position in the string is assumed to represent a particular feature of an individual, and the value stored in that position represents how that feature is expressed in the solution. Usually, the string is "evaluated as a collection of structural features of a solution that have little or no interactions". The analogy may be drawn directly to genes in biological organisms. Each gene represents an entity that is structurally independent of other genes.

Although test pattern generation (TPG) techniques for transistor stuck-open & stuck-short faults in Integrated Circuits are today considered mature, the test generation cost for large industrial circuits is still non-trivial, and very much time consuming process. The worst case complexity of the test generation is exponential as proven in [6].

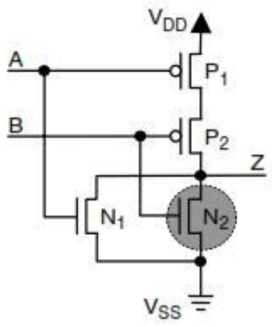


Figure1. *Two input NOR gate.*

Address for correspondence:
avireddym@gmail.com

With the increasing complexity and quality requirements for Integrated circuits, performance of TPG remains critical and methods that reduce the CPU time by a factor of two to three are still of great interests.

There are many approaches for ATPG, like deterministic approach, simulators etc. The aim of this technique should be both to reduce execution time and to improve fault coverage [3]. Genetic algorithm described by Goldberg [5] is to solve large scale combination optimization problem. GA has been successfully applied in different integrated circuit designs.

Faults in Transistor

As we all know that there are lots of transistorized faults occur in the VLSI circuits. i.e. stuck-at fault, transistor open and short fault, delay faults and crosstalk[7]. Defects in VLSI devices can include opens and shorts in the wires that interconnect the transistors to form circuit. Opens in wires tend to behave like transistor stuck-open faults when the faulty wire segment is interconnecting transistors to form gates.

On the other hand, opens tend to behave like stuck-at faults when the faulty wire segment is interconnecting gates. Therefore, a set of test vectors that provide high stuck-at fault coverage and high transistor fault coverage will also detect open faults. Now the DC analysis of this CMOS circuit is carried out.

Now the DC analysis of this CMOS circuit is carried out. The Faults are detected based on how far the output value from the desired output value.

GENETIC ALGORITHM

A simple genetic algorithm (GA) can be used for the generation of individual test vectors for combinational as well as sequential circuits. In a typical GA, a "population of individuals" (or chromosomes) is defined, where each individual is a solution for the problem at hand. As the individual represents a test vector for combinational circuit test generation, each character in the individual is mapped to a primary input.

If a binary coding is used, the individual simply represents a test vector. Each individual is associated with a fitness, which measures the quality of this individual for solving the problem. In the test generation context, this fitness measures how good the candidate individual is for detecting the faults.

The fitness evaluation can simply be computed by logic or fault simulation. Based on the evaluated fitness, the evolutionary processes of selection, crossover, and mutation are used to generate a new population from the existing population. The process is repeated until the fitness of the best individual cannot be improved or is satisfactory [7].

One simple application of GAs for test generation is to select the best test vectors for each GA run. The pseudo code of genetic algorithm for Automatic TPG is shown in Fig.1.

GENERATING TEST VECTORS

In this step, test vectors are created randomly. Population size should be large in order to ensure adequate diversity; however, it is a tradeoff between getting higher convergence rate with larger search space and less genetic operation time. Population size in algorithm of [2] is constant value for all circuits. Experiments have proven that required population size increases with increasing test vector length.

One simple application of GAs for test generation is to select the best test vectors for each GA run. A simple view of a test pattern generation using Genetic Algorithm is illustrated in Figure.2. The test generator starts with a random population of n individuals, and a (fault) simulator is used to calculate the fitness of each individual. The best test vector evolved in any generation is selected and added to the test set. Then, the fault set is updated by removing the detected faults by the added vector(s). The GA process repeats itself until no more faults can be detected [7].

Calculating the Fitness

The fitness function provides a quantification of the quality of the chromosome. It is the fitness of the Chromosome that determines whether the chromosome will be selected to produce offspring and quantifies its chance for survival among the other chromosome in the population to the next generation. The fitness function is problem specific. In this paper fault simulation with fault is with fault dropping is used in order to evaluate the test vectors. The score given to each individual is equal to the number of fault it detects, the fitness function for given test vector is calculated by equation (1) given below:

$$F(x) = \frac{\text{NO.of faults detected by test vector}}{\text{Total no.of faults detected}} \quad (1)$$

Creating a New Population

New population is created by repeating the following steps until the new population is complete.

Selection

The genetic algorithm uses selection operator to simulate natural evolution. In GA, individual with high fitness is inherited to the next generation with greater probability. Usually, chromosomes with high fitness are selected for crossover to converge faster to best solution.

Chromosomes with high fitness should not be selected for mutation to prevent the danger of diverting from good solutions in the search space. Therefore, chromosomes with low fitness are usually selected for mutation. All selection methods are based on the fitness of chromosomes [3]. The disadvantage of selecting chromosomes with high fitness is the probability of less diversity in the search space. Therefore, chromosome with low fitness is usually

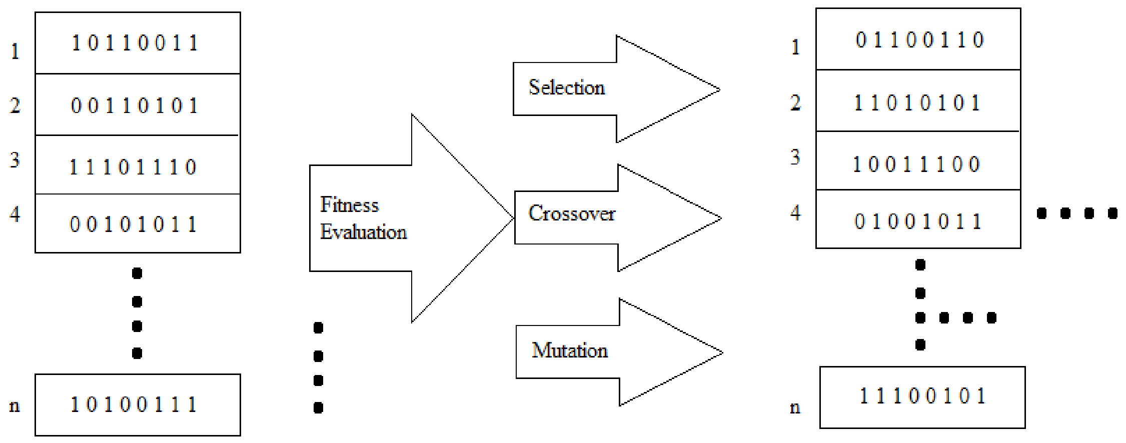


Figure2. *Test Pattern Generation using GA.*

selected for mutation. All Selection methods are based on the fitness of the chromosome.

Algorithm of Simple GA TPG

```
test set T =Ø;
while there is improvement do
    initialize a random GA currentPopulation;
    compute fitness of currentPopulation;
    for i = 1 to maxGenerations do
        add the best individual to test set T;
        nextPopulation =Ø;
        for j = 1 to populationSize/2
        do
        select parent1 and parent2 from currentPopulation;
        crossover(parent1, parent2, child1, child2);
        mutate_child1;
```

```
            mutate_child2;
            place child1 and child2 to nextPopulation;
            end for
        compute fitness of nextPopulation;
        currentPopulation = nextPopulation;
        end for
end while
```

In this paper Roulette wheel selection is used to select the individuals.

Roulette Wheel Selection

In this scheme, the probability that an individual will be chosen as a parent for the current crossover operation is equal to the proportion of the individual‟s fitness as compared to the total fitness value of the current population [10]. The proportion of offspring produced in this scheme by a fit individual as compared to the offspring by a less fit individual will be proportional to the ratio of their corresponding fitnesses. Imagine a roulette wheel where are placed all chromosomes in the population, every chromosome has its place big accordingly to

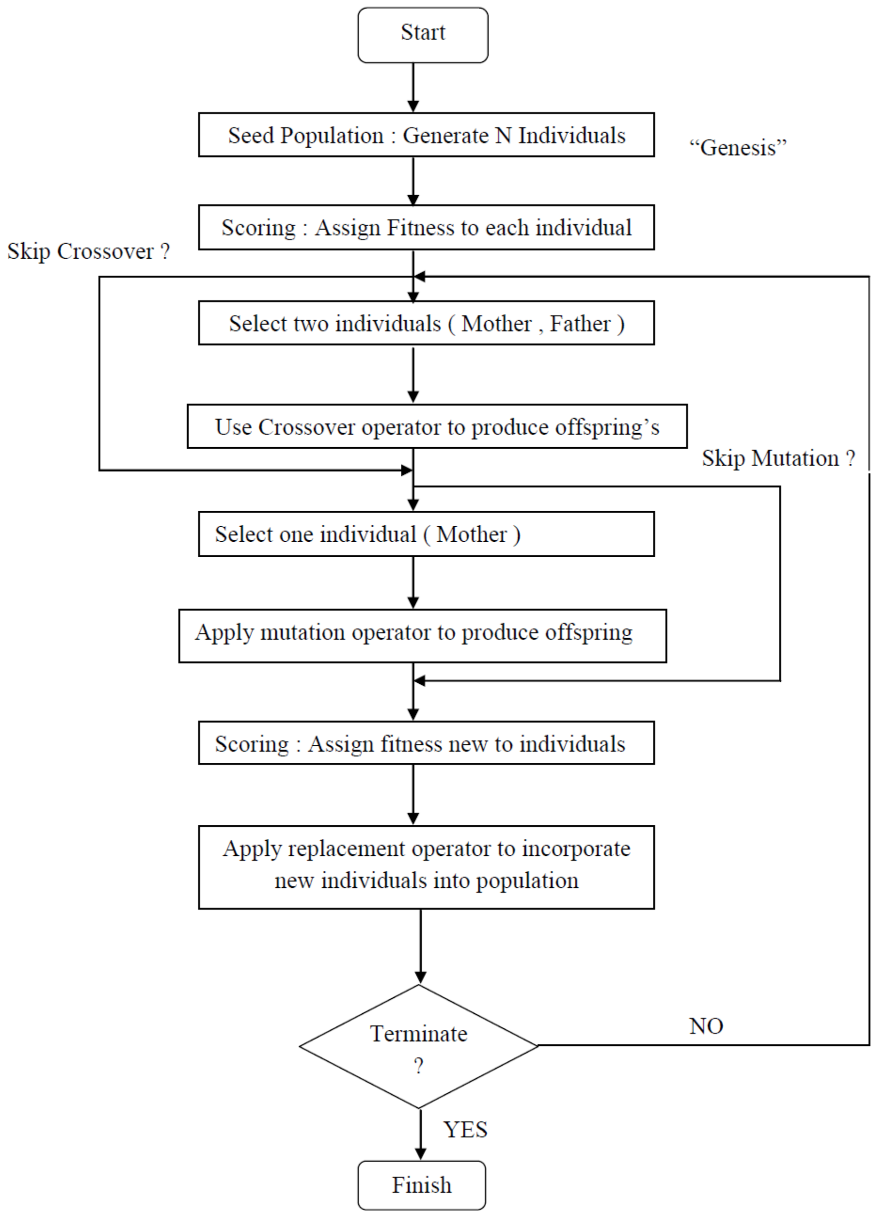
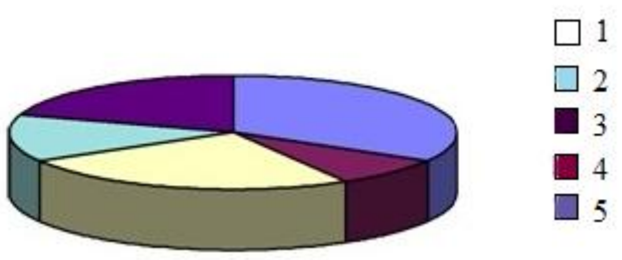

Start
Seed Population : Generate N Individuals
"Genesis"
Scoring : Assign Fitness to each individual
Skip Crossover ?
Select two individuals (Mother , Father)
Use Crossover operator to produce offspring's
Skip Mutation ?
Select one individual (Mother)
Apply mutation operator to produce offspring
Scoring : Assign fitness new to individuals
Apply replacement operator to incorporate new individuals into population
Terminate ?
NO
YES
Finish

Figure3. *Flow chart of Genetic Algorithm*



Chromosome 1

Chromosome 2

Chromosome 3

::

Chromosome 5

Figure4. *Roulette Wheel Selection.*

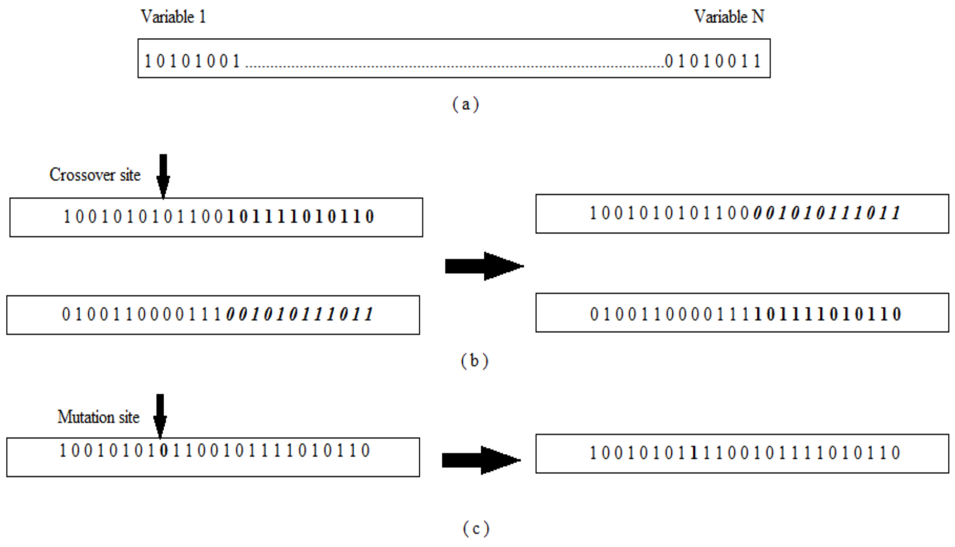


Figure5. *Crossover & Mutation.*

Table1. *Result table for faults in 2 inputs NOR gate.*

Sr. No.	**Test Vectors AB**	**P1**		**P2**		**N1**		**N2**		**Desired Output**	**Detected faults**	**Fitness Function f(x)**
		O	**S**	**O**	**S**	**O**	**S**	**O**	**S**			
	00	-	1	-	1	1	0	1	0	1		
1	01	0	0	0	1	0	0	-	0	0	6	6/8= 0.75
	00	-	1	-	1	1	0	1	0	1		
2	10	-	1	-	0	-	0	-	0	0	6	6/8= 0.75
	00	-	1	-	1	1	0	1	0	1		
3	11	0	0	0	0	0	0	0	0	0	4	4/8=0.50
	01	0	0	0	1	0	0	-	0	0		
4	10	-	1	-	0	-	0	-	0	0	4	4/8=0.50
	01	0	0	0	1	0	0	-	0	0		
5	11	0	0	0	0	0	0	0	0	0	2	2/8=0.25
	10	-	1	-	0	-	0	-	0	0		
6	11	0	0	0	0	0	0	0	0	0	2	2/8=0.25

its fitness function, like on the following picture.

Crossover

Traditional genetic algorithms worked on binary encodings of the problem instances. These genetic algorithms use simple crossover operators such as the uniform crossover and the one-point crossover to produce the offspring individual [10]. Crossover is the key to genetic algorithm, power that is to exchange corresponding genetic properties from the two parents, to allow useful genes on different parents to combine in their offspring. Most common crossover types are one-point, two-point, uniform crossover. In this paper, as shown in fig (5), two-point crossover is used.

Mutation

After a crossover is performed, mutation takes place. This is to prevent falling all solutions in population into a local optimum of solved problem. Mutation changes randomly the new offspring. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. It says how often will be parts of chromosome mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of chromosome is changed. If mutation probability is 100%, whole chromosome is changed, if it is 0%, nothing is changed. Mutation is made to prevent falling GA into local extreme, but it should not occur very often, because then GA will in fact change to random search.

EXPERIMENTS & RESULTS

As mentioned in above table DC analysis of two input NOR gate is carried out. By performing this experiment we are getting some of the test patterns to find transistorized fault like stuck-open and stuck-short.

As per result table test vector 00-01 & 00-10 has highest fault coverage. So instead of applying all the test vectors the optimized test vector is applied. Test vectors having very less fitness can be rejected in testing of the integrated circuits.

However transistor level test derivation algorithms are complex and time consuming, hence they may not be practical for large circuits.

CONCLUSION

This paper uses genetic algorithm for fault finding in VLSI circuits. Automatic TPG tools can reduce the amount of effort and cost of test generation. As we go through the result table we can easily select the test pattern, has highest fault coverage.

Our experimental results showed that genetic algorithm based method are efficient in test generation for the transistorized fault finding in VLSI circuits.

Here the experiment is carried out at single gate level only, but it can also be applicable for large Integrated circuits like Micro Controller, Processors, ASICs etc.

REFERENCES

[1] "An Efficient Automatic Test Pattern Generator for Stuck-Open Faults in CMOS Combinational Circuits" VLSI Design 1994, Vol. 2, No. 3, pp.199-207.

[2] Genetic Algorithms in Test Pattern Generation. By Eero Ivask Tallinn Technical University, Tallinn 1998.

[3] "Implementation of Genetic Algorithm for Automatic Test Pattern Generation". By-MrsRachna singh, Dr.Arvind Rajawat International Journal of Scientific & Engineering Research Volume 3, Issue 4, April-2012. ISSN 2229-5518.

[4] Goldberg,D.E,"genetic algorithms in search, optimization and Machine Learning.Reading,MA:Addison-Wesley.1989.

[5] "Introduction to Genetic Algorithms" Library of Congress control Number: 2007930221 ISBN 978-3-540-73189-4 springer Berlin Heidelberg New York.

[6] H.Fujiwara and S.Toida, "The Complexity of Fault Detection Problems for Combinational Logic Circuits", IEEE Trans, on Computers, june1982, pp. 555-560.

[7] VLSI test principals and architectures" Edited by: Laung-Terng Wang Cheng-Wen Wu Xiaoqing Wen.

[8] An Automatic Test Pattern Generator for Large Sequential Circuits based on Genetic Algorithms. By:- P. Prinetto, M. Rebaudengo, M. Sonza Reorda.

[9] Genetic algorithm for test pattern generator design, Automatic evolution of circuits by- Tomasz Garbolino · Gregor Papa , Published online: 23 February 2010 © Springer Science+Business Media, LLC 2010.

[10] Genetic algorithm based design and optimization of VLSI ASICs and reconfigurable hardware. Dt. 6-1-2009, By-Pradeep Ruben Fernando,University of South Florida.