Formulation of the Sequential Circuit Test Generation Problem within a Genetic Algorithm Framework

A Term Project in
Advances in Digital and Mixed Signal Testing

For the partial fulfillment of the degree of

Bachelor of Technology

by

Arnab Sinha
Roll No: 02CS3022
Department of Computer Science and Engineering

Debrup Das
Roll No: 02EG1004
Department of Electrical Engineering

under the guidance of

Prof. Indranil Sengupta

Department of Computer Science and Engineering
Indian Institute of Technology
Kharagpur
February 2006

Contents

1 Introduction **3**

2 The Problem **3**
2.1 Conventional Methods 3
2.2 Our Approach 4

3 The Formulation of the Problem **4**
3.1 The Various Definitions 4
3.2 The Overall Algorithm 4
3.3 The Crossover Algorithm 5
3.4 The Mutation Algorithm 6

4 Conclusion **7**

List of Figures

1 The Search Space 3
2 The Overall Algorithm 5
3 The Crossover 6
4 The Mutation 6

Abstract

Generation of test-cases in case of sequential digital circuits is a non-trivial task usually involving search over a large search space. Compared to the traditional methods like time-frame expansion etc, genetic algorithm offers a more faster solution to this search problem. In this report we formulate this problem in the framework of genetic algorithm.

1 Introduction

Genetic algorithms are finding increasing application in solving various problems from several engineering disciplines. The genetic algorithm is the favourite when the search space is large. The three main features in the genetic algorithm makes it a unique technique. They are *choice of population*, *fitness function* and *regeneration rule*. With reference to our problem, as the search space for the suitable test vectors for sequential circuit is pretty large, we can well utilize the power of genetic algorithms to reach the more preferred *solution* (here test case) faster.

2 The Problem

Here our problem is to find out the test-cases which are able detect single stuck-at-faults. Some test-vectors fail to propagate the stuck-at-fault to the primary outputs. We will like to have those vectors which can propagate the faults to the PO's. The pool of test-vectors is large (refer Fig 1). Essentially, this calls for application of some heuristics, which help in pruning vectors at a greater pace.

2.1 Conventional Methods

In order to test the sequential circuits researchers have devised mainly two kinds of methods−*time-frame expansion method* and *simulation based method*. Apart from the benefits it offers, the former has higher complexity, requires long initialization sequence, and has some other modelling limitations. On the other hand the later has some advantages over the other. With the help of the advanced simulation technology, we can use accurate simulation models to find out the test-cases targeted towards various fault models. In the simulation based techniques, *Concurrent Test Generation*, *Spectral Methods* and *Genetic Algorithm based method* are prominent.

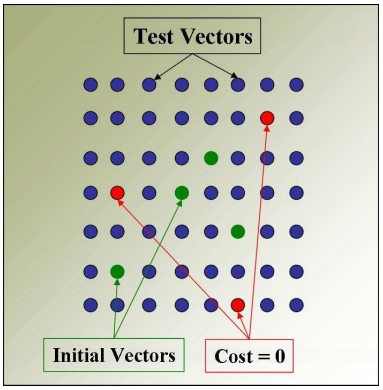

Figure 1: The Search Space

2.2 Our Approach

We are using the genetic algorithm framework for solving the problem. Our approach to the problem is divided broadly into two parts.

1. **Initialization of the circuit to some known state:** We assume that the circuit under consideration is cycle-free. Therefore, initialization of the circuit can be done within d_{seq} (i.e. the sequential depth) time-frames.

2. **Application of genetic algorithm to generate the test-cases:** We illustrate this formulation in detail in the later section.

3 The Formulation of the Problem

For any formulation using genetic algorithm framework, we need to define the elements of population, the associated cost-function and the regeneration rules i.e. *mutation* and *crossover*. We formally define the terms in following subsection.

3.1 The Various Definitions

Suppose the total population is P_{total} consisting of all possible test-vectors. In the following discussion, we will be referring to a sample set of vectors as a *generation* in a particular iteration. So if there are N inputs then, $|P_{total}| = 2^N$. We define the present generation population $P_{present}$. Initially, $P_{present}$ consists of randomly chosen k vectors from P_{total} (such that $k << P_{total}$). Next we define, the next generation population as P_{next}. The intermediate generation is denoted as P_{mix}. Moreover, we define a user-defined parameter `elitism`, which restricts the percentage of *elite* population in the sample $P_{present}$. We associate a cost(say C_i) to i-th element (v_i) in the sample population, which is computed using some huristic.

3.2 The Overall Algorithm

We will be describing the overall algorithm with reference to Fig 2. In a genetic algorithm problem, at the start of any iteration, we have $P_{present}$. Our aim in that iteration is to find P_{next}.

1. Randomly select a sample $P_{present}$ (of size k) from P_{total}.

2. Calculate C_i's $\forall v_i \in P_{present}$.

3. Sort $v_i \in P_{present}$ in ascending order of C_is.

4. Copy `elitism` percentage of population strength of $P_{present}$ with minimum C_i's to P_{mix} and P_{next}.

5. Fill the rest of the P_{mix} by any random selection from $P_{present}$.

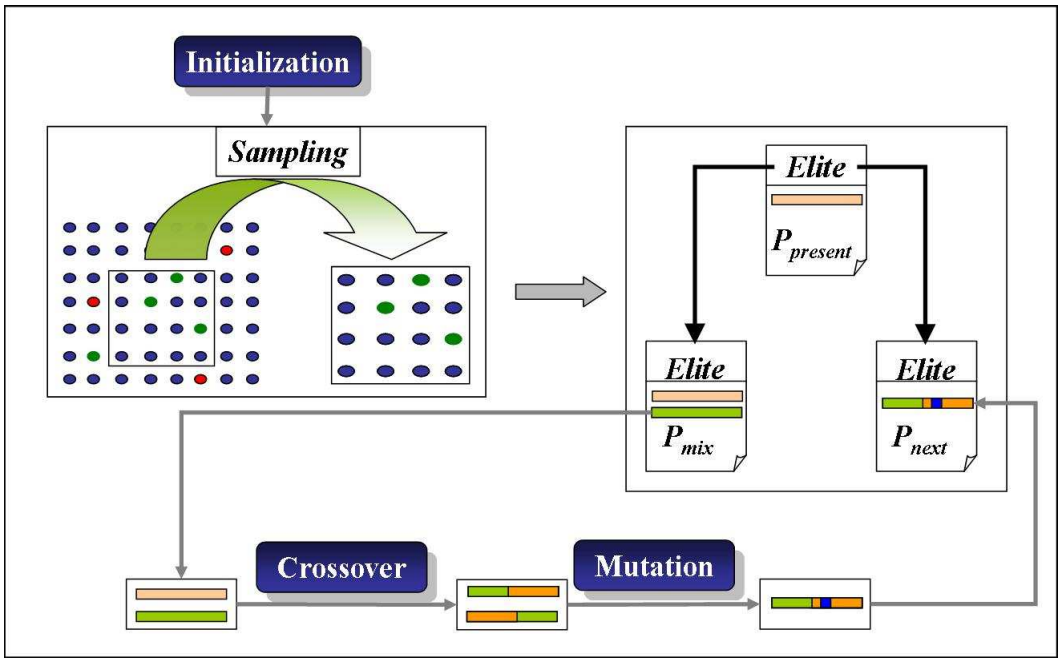

Figure 2: The Overall Algorithm

6. Next perform ***Crossover*** followed by ***Mutation*** on any two randomly selected v_i, $v_j \in P_{mix}$, as described later, to get modified $\tilde{v_i}$ and $\tilde{v_j} \in P_{next}$. This process is iteratively performed to get the non-elite part of P_{next}.

7. Sort the elements in P_{next} according to their ascending cost.

8. If either $P_{next} = P_{present}$ or max-iteration is reached then sort P_{next} in ascending order of C_i and terminate, else $P_{present} \leftarrow P_{next}$ and goto step 2.

The v_i's with minimum C_i (which if $C_i = 0$, implies that it can propagate a fault to the PO's) contained in P_{next} are the set of best possible test-vectors.

3.3 The Crossover Algorithm

In crossover, we begin with two sample points, v_i and v_j each of n-bits. In order to explain the algorithm we will take a small example, followed by the formal algorithm. Say $v_i = 1011$ and $v_j = 1110$. Suppose, we perform crossover over the 2 last bits. After crossover the modified $\tilde{v_i} = 1010$ and $\tilde{v_j} = 1111$. The process is described in the Fig 3. This can be done multiple times and at various positions depending on the vector-length. Now we give the formal algorithm.

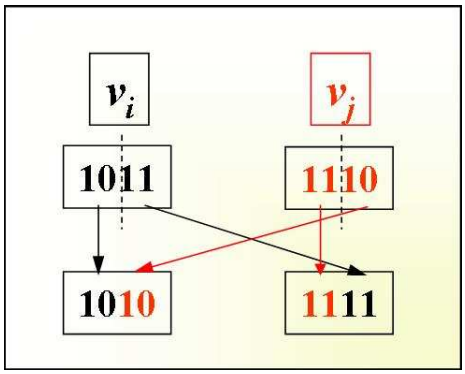

Figure 3: The Crossover

Algo. 3.1 CROSSOVER

Crossover(v_i, v_j)
begin
1: Set $k \leftarrow N/5$. // say
2: **while**(rand()%k−*iteration_no*≥0)
// so that with each iteration probability of crossover falls off
3: {
4: Set breakpoint←rand()%$(N-1)$.
5: Exchange(v_i,v_j) upto breakpoint.
6: }
end

3.4 The Mutation Algorithm

Here we simply toggle a random bit of the vector. For instance, referring to the Fig 4, where the vector $v_i = 1001$, the second MSB is randomly chosen and toggled to get $\tilde{v_i} = 1101$. We formally give the algorithm.

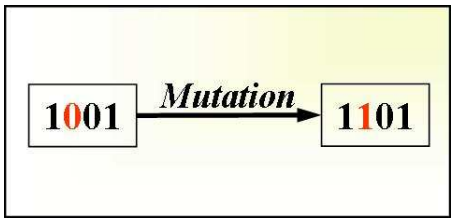

Figure 4: The Mutation

Algo. 3.2 MUTATION

Mutation(v_i)
begin
1: Set $k \leftarrow N/5$. //say
2: **while**(rand()%$k - iteration\_no \geq 0$)
3: {
4: Set *togglepoint*←rand()%(N).
5: $\tilde{v_i} \leftarrow$Toggle(v_i, *togglepoint*).
6: }
end

4 Conclusion

For very large value of N it is not always practical to do a directed search, because it might mislead us to a solution space where there exists no suitable test-case. But genetic algorithm, due to the features like crossover and mutation, the sample population consists of vectors, which are located at a greater distances from each other in the solution-space, thereby increasing chances of coming across suitable vectors.