CHAPTER 2

FAULT SIMULATION

Fault simulation is to be necessarily performed on every manufactured circuit to verify its functional correctness and to check for the defects using a set of test vectors which result due to the manufacturing process. The number of test vectors required increases exponentially with the increase in the number of primary inputs and primary outputs. So, an optimization technique is required, which produces the minimum set of test vectors to test all the possible faults in a circuit. Various fault models such as Stuck-at Fault, Bridge Fault, and Delay Fault are targeted in this chapter using Genetic Algorithm (GA), Modified Genetic Algorithm (MGA) and Hierarchical Modelling. Both Combinational and Sequential circuits are used for fault simulation.

2.1 INTRODUCTION

A designer when designs a logic, fabricates and tests it, and if the test fails, then there must be a cause of the failure. Either of the following may be a cause: (1) the test is wrong, or (2) the fabrication process is faulty, or (3) the design is incorrect, or (4) the specification has a problem. Anything out of the above options can be true. So, the role of testing is to detect if something wrong has happened and the role of diagnosis is to determine exactly the cause.
of failure, and to identify where the process needs to be altered. So, to generate the perfect products, the correct and effective testing procedure has to be followed. The benefits of testing are Quality and Economy. Quality and Economy can be explained as means satisfying the user’s need at a minimum cost [Bushnell M, et al., 2002]. A good test process can weed out all bad products before they reach the user. It is naïve to think that every fabricated chip is good. Impurities in materials, equipment malfunctions and human errors are some causes of defects [Giacomo J. D, et al., 1989]. The likelihood and consequences of defects are the main reasons for testing. Another very important function of testing is the process diagnosis where the root cause can be found which results in a faulty chip, whether it is in fabrication, or in design, or in testing. The faulty chip analysis is called Failure Mode Analysis (FMA).

**Definition of Fault Simulation problem**

To ensure the quality, the fabricated chips must be tested with the help of test patterns generated by ATPG (Automatic Test Pattern Generator) or manually. In addition, the desired quality level of the circuits has placed an increasing demand on the quality of test pattern generation [Roth J. P, et al., 1980]. The functionality of a manufactured circuit is evaluated by modeling the physical fault in its replica form by using the corresponding fault model and is called as fault simulation. There are various types of faults present in an integrated circuit and the modeling of faults is closely related to the modeling of the circuit [Bushnell M, et al., 2002]. The most popular faults are:

1. **Bridging Fault:** A bridging fault represents a short between the groups of signals and the logic value of the shorted net may be modeled as 1-dominant (OR bridge), 0-dominant (AND bridge), or indeterminate, depending upon the technology in which the circuit is implemented. Non-feedback bridging faults are combinational and their coverage by stuck-at fault tests is normally very high.

2. **Stuck-at Fault:** A stuck at fault represents the situation when the signal node is stuck at either logic 1 or logic 0 level permanently irrespective of the applied input value at the primary inputs. This fault is modeled by assigning a fixed (0 or 1) value to a signal line in the circuit which is usually an input or an output of a logic gate or a flip-flop. The most popular forms are the single stuck-at faults, i.e., two faults per line, stuck-at-1 (s-a-1 or sa1) and stuck-at-0 (s-a-0 or sa0).
3. **Transition Fault:** Delay faults cause errors in the functioning of a circuit based on its timing. The faults caused by the rise and fall times are called transition delay faults. Due to this finite time, it takes for an input of a gate to show up on the output, faults may arise if the signals are not given the time to settle. The gate delay, usually an increase over the nominal value, is assumed to be large enough to prevent a passing transition from reaching any output within the clock period, even when the transition propagates through the shortest path. Possible transition faults of a gate are slow-to-rise and slow-to-fall types and hence the total number of transition faults is twice the number of gates.

**Objective of Fault Simulation problem**

*A defect or a physical fault in a real circuit is often modeled by a logical fault for testing.* In order to generate a test for a fault condition, it is necessary to model the fault condition and simulate the circuit operation with the modeled fault condition present. The fault model can be used to represent a physical defect by altering the characteristics of a circuit to enable it to perform as if a fault condition is present [Kumar R, et al., 2008]. Here, the quality of fault test vectors is evaluated using a fault simulator [Bening L, et al., 2000]. To detect all such faults, an optimization approach is to be adopted to choose a minimum of the total possible vector combinations which are formed by trying different input combinations. **So, an optimization algorithm is to be designed to solve the fault simulation problem using the following fitness function:**

\[
\text{Fitness} = \begin{cases} 
1 & \text{if fault is excited} \\
0 & \text{otherwise}
\end{cases}
\]

**2.2 FAULT SIMULATION IN COMBINATIONAL CIRCUITS**

Fault simulation in digital circuits is an extremely important step in ensuring the yield of the production line as it validates that the resultant hardware contains no defects that could have otherwise, adversely affect the product’s correct functioning [Berglez F, 1984]. **Fault models are used to replicate the behavior of the physical fault in software form.** The test is a sequence of input vectors that can differentiate the good circuit from the faulty circuit [Iyer M. A, et al., 1994]. Typically, for combinational circuits, each test consists of a single input vector. **A set of tests is called a test set.** For a combinational circuit, if a fault is undetectable, the fault is redundant [Stephan P, et al., 1996]. Every digital circuit is
composed of logic gates which can be classified differently depending on the function performed. The function of each gate is described using its truth table. By using the information from the truth table, the test vectors required to uncover the faults can be found for each logic gate. The details of a logic gate are given below.

**Detection of Fault in Logic Gates**

In a 2 input AND gate, let A and B be the inputs and Q be the output. Let stuck-at-0 and stuck-at-1 faults at the different nodes be denoted by A/0, A/1, B/0, B/1, Q/0, and Q/1. The input combinations which can detect the specific faults are shown in Table 2.1.

![AND Gate Diagram]

**Table 2.1: Truth Table of 2 Input AND gate**

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>Q</th>
<th>Faults detected</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>A/1, B/1, Q/1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>A/1, Q/1</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>B/1, Q/1</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>A/0, B/0, Q/0</td>
</tr>
</tbody>
</table>

Each combination of input vector values is used to detect some stuck at faults depending on the excitation which they make at the primary output of the logic gate, for e.g. with A and B = 0, three stuck at faults i.e. stuck-at-1 at A, stuck-at-1 at B, stuck-at-1 at Q can be detected respectively. By using the information from the above-described truth table, the fault model for stuck-at fault can be defined.

*The detection principle of different types of faults in combinational circuits is explained in sections (i), (ii) and (iii) below.*

**(i) Stuck-at Fault Model & its Detection**

It is the most common fault present in combinational circuits. A stuck-at fault implies that an input or output line of a logic gate is permanently stuck at logic value 1 or 0 (stuck-at-1 or stuck-at-0). While the *single stuck-at fault model* assumes only one fault in a circuit, the *multiple stuck-at fault model* allows several stuck-at faults simultaneously. However, in most cases, multiple faults can be detected by the tests designed for the individual single stuck-at faults [Arslan T, et al., 1997].
Consider a circuit, as shown in Figure 2.1, in which a node Y is stuck at logic 0 and a test vector is generated to detect it. In order to detect it, the nodes must be driven to logic 1 [Takahashi H, et al., 2002]. For this to happen all input nodes, i.e. A to J must all be connected to logic 1. Next, to propagate the fault to node Z, input nodes from K to S must be connected to fixed logic. Nodes A through J must, therefore, be set to 1. To propagate the fault effects at any circuit node to output node Z, nodes K through S must be connected to 0. Such a specific input pattern is very difficult to be generated randomly [Aylor J, et al., 1991].

![Figure 2.1: Stuck-at fault in a circuit](image)

Therefore, these faults can only be detected by using specific input combinations. To check for all possible combinations of inputs is a very cumbersome job. Hence an optimization technique is required, which can result into fittest individual, i.e. test vector based on heuristics [Takahashi H, et al., 2002].

(ii) Delay Fault Model & its Detection

Another important category of faults is the delay faults which have been introduced due to some problem in the fabrication process [Ohtake S, et al., 2003]. Delays are present due to the slow transitional response at the circuit node and to detect them, a test stimulus pair is used [Abdulnazzaq N.M, et al., 2003]. The first stimulus initializes the value of the node and second causes a change in value at that node. To detect it the resulting transition must be taken to the primary output [Xu Q, et al., 2003]. The fault is classified in two categories, slow-to-rise or slow-to-fall condition. A delay defect may cause an error at the primary output of the circuit.

In order to test for delay defects, it is necessary to initialize the state of all nodes within the circuit, before a test pattern may be applied to the primary inputs of the circuit [O'Dare M. J, et al., 1996]. The test to detect such faults is comprised of a set of test patterns \{T1, T2\}, applied in order sequence as shown in Figure 2.2.
If the sequence of input vectors results into an equivalent transition at the output node then only the fault is traceable otherwise not \cite{Cheng K. T, et al., 1993; Gharaybeh M. A, et al., 1997}. To evaluate the technique, a test circuit as shown in Figure 2.3, is used. \textit{As the circuit may contain more than one type of fault also, so the test vector set which detects the delay faults can also detect stuck-at faults in the same step by using same input pattern.}

Two delay faults are presented:

\begin{enumerate}
  \item \textbf{Slow to Fall (STF) at F}: It is detected with the input pair as “0001” & “1101”
  \item \textbf{Slow to Rise (STR) at E}: It is detected with the input pair as “1101” & “1100”
\end{enumerate}

The true circuit behavior with no faults present is evaluated first and stored. The states of the circuit nodes are initialized using the first set of input patterns and are stored in the reference table. The next step is the application of another part of the vector pair to initialize a transition at the chosen node. The transition is reflected in the output node to get compared with the stored response in the reference table. If found different, then the fault is properly excited and detected also.

\textit{(iii) Bridge Fault Model \& its Detection}

In both the bridge faults and stuck at faults, the logic is permanently fixed to some logic
value [Wu J, et al., 2000]. But, in bridge faults, different values which are available on the connected lines, decide the final value. In a situation, when two signal lines $W_1$ and $W_2$ has a short in between, a new logic function gets created. It is called a bridging fault. There are two types of bridging faults, if logic 1 overrides logic 0, it is called OR bridging fault and if logic 0 overrides logic 1, it is called AND bridging fault. In Figure 2.4, let us assume that logic 1 overrides logic 0 and consider a bridging fault instance, between signal lines $w_1$ and $w_2$ [Lavo, et al., 1997]. Let $A_0$ and $B_0$ as the logic values in a fault free situation and $A_f$ and $B_f$ be the logic value in the presence of logic faults. Consider that the OR bridging fault is present. As a result $A_f = B_f = A_0 + B_0$. In another figure, AND bridging fault is present. Hence, $A_f = B_f = A_0 B_0$.

![Figure 2.4: Non-Feedback Bridging Fault](image)

2.2.1 Fault Simulation using Genetic Algorithm

The fault simulation is targeted using Genetic Algorithm. It is a nature inspired optimization approach which mimics the natural evolution process [Patel Lalit, et al., 2013]. The approach is explained using a simple test circuit as shown in Figure 2.5. The circuit consists of two NAND gates and an OR gate with three inputs A, B, and C. The test vector comprises of three bits \{A, B, C\}, of the possible $2^3 = 8$ input vectors.

![Figure 2.5: Test Circuit with 3 inputs and 1 output.](image)
For the test circuit, a population of six chromosomes representing potential test vectors is manually generated randomly by using a random function. All test vectors are applied as possible inputs to a circuit represented in a software module form for a fault-free circuit and the corresponding expected output is recorded. Each node within the circuit is then forced to logic ‘1’ and a logic ‘0’ to stimulate stuck-at-one and stuck-at zero respectively. If the circuit’s output differs from that of the fault-free response the simulated fault condition is said to be detected by the applied pattern. A scoring function is used to keep track of the fitness of each pattern and, the fault patterns detected are recorded [Shahhoseini H. S, et al., 2007]. From the current mating pool, the GA selects only those individuals with fitness values higher than others. A figure of ten points is awarded for each fault detected by a pattern; the total value awarded to a pattern is that pattern's fitness value are as given in Table 2.2.

Table 2.2: Fitness value evaluation scheme

<table>
<thead>
<tr>
<th>Input Pattern</th>
<th>Faults detected</th>
<th>Fitness value</th>
</tr>
</thead>
<tbody>
<tr>
<td>011</td>
<td>D/0, A/1, E/0, Out/1</td>
<td>40</td>
</tr>
<tr>
<td>101</td>
<td>D/0, B/1, E/0, C/0, Out/1</td>
<td>50</td>
</tr>
<tr>
<td>000</td>
<td>D/0, C/1, E/1, Out/0</td>
<td>40</td>
</tr>
<tr>
<td>101</td>
<td>D/0, B/1, C/0, E/0, Out/1</td>
<td>50</td>
</tr>
<tr>
<td>001</td>
<td>D/0, C/0, E/0, Out/1</td>
<td>40</td>
</tr>
<tr>
<td>011</td>
<td>D/0, E/0, A/1, Out/1</td>
<td>40</td>
</tr>
</tbody>
</table>

Having determined fitness values for the initial population, it is necessary to select parents to for crossover operation. The Roulette Wheel Method is used for parent selection, which ensures that the assigned fitness score is proportional to the probability of their selection [Takahashi H, et al., 2002]. Each parent undergoes crossover and the two parents selected for crossover interchange their strings and generate new patterns having properties of both parents. This procedure continues until a whole new population of chromosomes is created. The probability of crossover is 0.8. The next genetic operator to be applied to the population is bit mutation, which introduces new information into the population. The probability of any bit within the population being altered is approximately 0.25. The reference table provides information about the undetected faults during each iteration. This information is used as the basis of the scoring function that assigns new fitness values to the population members. The algorithm determines the fittest member in each generation; this pattern is now a candidate for entry into the test set, but with a condition of improvement in fitness.
score. Thus, the possibility of duplication of any pattern within the test set is eliminated. Once the above procedure is completed, a new population has been created. The new population must be re-assessed for assignment of fitness values. If a test pattern is successful in being chosen for entry into the test set, the Universal Record Table is updated with the fault coverage of that pattern. From the fitness value in Table 2.2, the two test vectors selected for crossover be 101 and 011. The faults covered by each of them and the faults that remain to be covered are entered into the Universal Record Table as shown in Table 2.3.

Table 2.3: Universal Record Table

<table>
<thead>
<tr>
<th>Test Vector</th>
<th>Faults detected</th>
<th>Faults remaining</th>
</tr>
</thead>
<tbody>
<tr>
<td>101</td>
<td>D/0, B/1, C/0, E/0, Out/1</td>
<td>A/0, A/1, B/0, C/1, D/1, E/1, Out/0</td>
</tr>
<tr>
<td>011</td>
<td>D/0, E/0, A/1, Out/1</td>
<td>A/0, B/0, B/1, C/0, C/1, D/1, E/1, Out/0</td>
</tr>
</tbody>
</table>

A new fitness value is calculated for a new set of test vectors based on the Universal Record Table. For each test vector detecting a fault, which has not been detected so far as observed from the Universal Record Table, 10 points are allotted to the vector (individual). If the test vector, however, detects a fault, which has already been detected as seen from the Universal Record Table, only 1 point is allotted to the vector. Hence, this Universal Record Table provides the model with a ready reference of undetected faults and also prevents against awarding unnecessary high value to a vector, which is duplicating information. Fitness values awarded to a new set of test vectors based on the Universal Table are shown in Table 2.4.

Table 2.4: Fitness evaluation using Universal Record Table

<table>
<thead>
<tr>
<th>Test Vector</th>
<th>Faults detected</th>
<th>Fitness value</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>B/0, A/0, E/0, Out/0, D/1</td>
<td>41</td>
</tr>
<tr>
<td>000</td>
<td>D/0, C/1, E/1, Out/0</td>
<td>31</td>
</tr>
<tr>
<td>100</td>
<td>B/1, D/0, C/1, E/1, Out/0</td>
<td>32</td>
</tr>
<tr>
<td>101</td>
<td>B/1, D/0, C/0, E/0, Out/1</td>
<td>5</td>
</tr>
<tr>
<td>011</td>
<td>A/1, D/0, E/0, Out/1</td>
<td>13</td>
</tr>
<tr>
<td>001</td>
<td>D/0, C/0, E/0, Out/1</td>
<td>4</td>
</tr>
</tbody>
</table>

The fittest member of the population in each generation is then entered into the test site, and the information in the Universal Record Table is updated. However, a chromosome is only entered if it improves the fault coverage for the circuit, this procedure helps to achieve one more goal of removing the possibility of duplication.
**Algorithm for Fault Simulation using a Genetic Algorithm**

The Genetic Algorithm as implemented is described below in a sequence of steps.

The pseudo code for GA is represented below in abstract form as:

```plaintext
Algorithm: GA(n, χ, μ)

1. Initialise generation 0:
   - \( k := 0 \);
   - \( P_k \) := a population of \( n \) randomly-generated individuals;

2. Evaluate \( P_k \):
   - Compute \( \text{fitness}(i) \) for each \( i \in P_k \);

3. Do:
   - Create generation \( k + 1 \):
     - 1. Copy:
       - Select \( (1 - \chi) \times n \) members of \( P_k \) and insert into \( P_{k+1} \);
     - 2. Crossover:
       - Select \( \chi \times n \) members of \( P_k \); pair them up; produce offspring; insert the offspring into \( P_{k+1} \);
     - 3. Mutate:
       - Select \( \mu \times n \) members of \( P_{k+1} \); invert a randomly-selected bit in each;
     - Evaluate \( P_{k+1} \):
       - Compute \( \text{fitness}(i) \) for each \( i \in P_k \);
     - Increment:
       - \( k := k + 1 \);

4. While \( \text{fitness} \) of fittest individual in \( P_k \) is not high enough:

5. Return the fittest individual from \( P_k \);
```

Here, \( n \) is the number of individuals in the population; \( \chi \) is the fraction of the population to be replaced by the crossover in each iteration; and \( \mu \) is the mutation rate.

**Flowchart for Fault Simulation using Genetic Algorithm**

A flow chart view representing different functional blocks of the algorithm is presented in Figure 2.6.

![Flowchart for Fault Simulation using Genetic Algorithm](image-url)

Figure 2.6: Fault simulator flowchart using GA
Results and Discussion

The GA-based fault simulator is tested over three test circuit which are described below as case 1, 2 and 3.

Case 1: 3 Input logic circuit for stuck-at fault simulation

The GA’s performance for a 3 input circuit shown in Figure 2.5 is illustrated in a plot shown in Figure 2.7. The genetic algorithm with an initial population of 6 has generated test vectors which have uncovered all the faults present in the circuit. As the generation counter progresses, the number of remaining faults decreases progressively and in 4 iterations it becomes 0.

The population size with which it has been started has an important effect on the number of iterations required to find the total faults. Figure 2.8 shows that with less number of initial population members, i.e., 3, the GA has to search more in the solution space as compared to a number of initial population members and thus takes more iterations to reach the optimum solution.
Evolutionary Algorithms for VLSI Applications

The Figure 2.9 shown below depicts that the maximum fitness and the average fitness values for each population set generated during algorithm execution are decreasing. This happens because, in the first loop of the genetic algorithm, the maximum score is awarded and after that since some faults detected are already discovered by the previous fittest test vector, so a very low score is awarded.

**Hence, the max fitness and the average fitness scores reduce progressively with the generation counter.**

![Maximum Fitness and Average Fitness Variation](image)

Figure 2.9: Maximum fitness and Average fitness variation Iteration wise

**Case 2: 4 Input logic circuit for both stuck at fault and delay fault simulation**

The circuit shown in Figure 2.10 is a four input circuit with a total number of input combinations as 16. For the test circuit in consideration, 30 stuck at faults and 20 delay faults are detected in total.

![Test Circuit](image)

Figure 2.10: Test circuit with 4 Inputs and 11 logic gates
During each iteration, the test vector information is referred and then referenced from the result of crossover and mutation operator to be referred for the coming generation. The results are shown in Figure 2.11. It saves time by not searching for the same pattern of inputs which have been searched during previous iterations for the small circuit. The approach is applicable for both delay and stuck at faults.

![Figure 2.11: Fault simulation of Test circuit](image)

**Case 3: Four-bit adder circuit for stuck at, delay and bridge fault simulation**

The test circuit shown in Figure 2.12 is with 9 inputs and 5 outputs. It is a benchmark problem to start with fault simulation using an optimization. Module M1 is 74283 circuit and is used to implement $A \oplus B$ operation.

![Figure 2.12: Four-bit Adder Test circuit](image)

**Statistics:** 9 inputs; 5 outputs; 54 gates.

**Function:** The module M1 produces generate i.e. $G$, propagate i.e. $P$, and XOR functions i.e. $A \oplus B$. 
Due to the presence of numerous intermediate nodes, the chances of physical defects in this test circuit are much more than the previous two test circuits i.e. case 1 & case 2. In this case, from gate level implementation, the fault information obtained is stuck at faults are 172, delay faults are 65 and bridge faults are 47. Due to increase in the number of inputs, a tremendous rise in the fault probability takes place. Figure 2.13 provides the result of the test circuit.

Figure 2.13: Fault detection for Four bit Adder Test circuit

Figure 2.14, gives an explanation about the effect of population size on the number of generations taken by the genetic algorithm to find the optimum solution.

*With the reduction in population size from 50 to 30 the iterations required to cover all the faults got increased to 70 from 50.*
The crossover rate and mutation rate are normally fixed in GA. In circuits with a number of logic gates more than 50, when the circuit is not evolving, then these rates can be varied during the course of simulation to increase the fault detection chance. Figure 2.15, shows the effect of varying the mutation rate on the number of generations. When the generated solutions are not giving good detection rates, then increased mutation rates from 0.01 to 0.2 leads to the new population members generation which provides new search space to the solution fabric. It has been found that the variable assignment in crossover and mutation leads to better solutions in less iteration count. The genetic algorithm is effective in detecting the test set for most of the circuits considered. But it suffers from the inherent drawback that the implementation time also increases considerably with the increase in circuit size and the control parameters are also not always helpful in escaping local minima. So, its modified approach termed as Modified Genetic Algorithm (MGA) is used.

2.2.2 Fault Simulation using Modified Genetic Algorithm

The modified genetic algorithm is an efficient population-based differential technique for solving various optimization problems over continuous space. Its success depends upon the proper selection of parameters such as population size $NP$, scaling factor $F$, and crossover rate $CR$ and the mutation strategy [Huang Z, et al., 2013]. In a $D$-dimensional search space, an individual is represented by a $D$-dimensional vector $(x_{i1}, x_{i2}, \ldots, x_{iD})$, $I = 1,2,\ldots, NP$ where NP is the population size (number of individuals).

There are three control operators, namely: Mutation, Crossover, and Selection. Initially, a population is generated randomly with uniform distribution and then by applying mutation,
crossover and selection operators, new members are generated [Aslantas V, et al., 2007]. The trial vector generation is important and mutation and crossover are used to generate them. The selection operator is used to select the best trial vector for next generation [Storn R, et al., 1997].

(i) **Mutation**

For each individual of the current population, a trial vector is generated by mutation operator. To generate it, a target vector is mutated with a weighted differential. A new vector is produced in the crossover operation using the newly generated trial vector. If the generation counter is $G$, the mutation operator for generating a trial vector $u_i(G)$ from parent vector $x_i(G)$ is defined as follows:

- Select a target vector, $x_{i_1}(G)$, from the present population, such that $i \neq 1$.
- Randomly select two individuals, $x_{i_2}$ and $x_{i_3}$, from the present population, such that $i \neq i_1 \neq i_2 \neq i_3$.
- The target vector is mutated for trial vector calculation as in eq. 2.1:

$$u_i(G) = x_i(G) + F(x_{i_2}(G) - x_{i_3}(G))$$  

Where $F \in (0, \infty)$ is the mutation scale factor used in the control of differential variation.

(ii) **Crossover**

The child $x'_j(G)$ is generated using the crossover of parent vector, $x_j(G)$ and the trial vector, $u_i(G)$ as in eq. 2.2:

$$x'_j(G) = \begin{cases} u_j(G), & \text{if } j \in J \\ x_j(G), & \text{otherwise.} \end{cases}$$  

Where $J$ is set of crossover points, $x'_j(G)$ is the $j$th element of the vector $x_j(G)$.

(iii) **Selection**

It selects the individual for the mutation operation to generate the trial vector and it selects the best, between the parent and the offspring based on their fitness value for the next generation. If the fitness of a parent is greater than offspring than the parent is selected, otherwise offspring is selected as in eq. 2.3:
\[ x_i(G+1) = \begin{cases} x_i(G), & \text{if } f(x_i(G)) > f(x_i(G)) \\ x_i(G), & \text{otherwise.} \end{cases} \] \hspace{1cm} \text{(2.3)}

It ensures that the population’s average fitness does not deteriorate.

**Algorithm for Fault Simulation using a Modified Genetic algorithm**

The Pseudocode for modified genetic algorithm approach is described as follows:

Initialize the control parameters, F and CR  
Randomize to create the initial population, P (gen), of NP individuals;  
While stopping condition(s) not true do  
For each individual, \( x_i(G) \) \( \in \) P(G) do  
\text{Evaluate the fitness, } f(x_i(G)) \text{.}

Create the trial vector, \( u_i(G) \) by using the mutation operator;  
Create the child, by using crossover operator;  
If \( f(x_i(G)) \) is better than \( f(x_i(G)) \) then  
Add \( x_i(G) \) to P (G+1)  
else  
Add \( x_i(G) \) to P (G+1)  
end if  
end for  
end while

Return the individual with the best fitness as the solution

**Flowchart for Fault Simulation using Modified Genetic algorithm**

The flowchart of the MGA based fault simulator is shown in Figure 2.16.

![Figure 2.16: Modified Genetic Algorithm flowchart](image-url)
Results and Discussion

A set of three test circuits is used to investigate the effect of the improved algorithm. The 74182 is used to demonstrate the performance of the algorithm.

Test Circuit: 74182: Carry Look-Ahead Circuit (CLA)

The CLA circuit is described in Figure 2.17.

![Figure 2.17: Four bit Magnitude comparator Test circuit](image)

Statistics: 9 inputs; 4 outputs; 19 gates

Function: The 7485 logic can be functionally modeled as shown in Appendix B. The performance is illustrated with the 9 inputs circuit as shown in Figure 2.18. The Figure 2.18 demonstrates the efficiency of MGA both for stuck-at faults and delay faults. The test circuit is having 66 stuck at faults and 28 delay faults. All the stuck-at faults are detected in 12 executions of the algorithm while delay faults need 20 iterations of the algorithm.

![Figure 2.18: Fault Simulation for Test circuit](image)
The population size has a major effect on the number of iterations occurred to find the total faults. A comparison between different population sizes is made and the result obtained is drawn as reflected in Figure 2.19.

![Figure 2.19: Effect of Population size variation on Fault simulation](image)

The differentiation constant and crossover constant are the two most important parameters of the modified genetic algorithm. By varying their values during optimization, it has been found that the convergence rate of the fault simulator can be enhanced as depicted in Figure 2.20 and Figure 2.21. The details of the parameters used for optimization are:

Specifications: Differentiation constant: 0.2, Crossover constant: 0.35

![Figure 2.20: Differentiation constant & Crossover constant value 1 variation](image)

The delay faults need 8 iterations and stuck-at faults need 16 iterations for their detection.
Specifications: Differentiation constant: 0.8, Crossover constant: 0.55

Figure 2.21 demonstrates that in comparatively lesser generations higher fault coverage can be achieved by using varying crossover rate. The increase in differentiation constant and crossover constant helps the GA to generate more efficient test vectors in less number of iterations. *With this set of parameters, all the faults have been detected in 10 iterations while in the earlier case it requires 6 more iterations to complete the fault simulation.* If no fault detection is achieved in 3 successive generations. In order to measure the efficiency of the modified genetic algorithm technique for test simulation, the results have been compared in the form of Table 2.5, with Genetic Algorithm (GA) where GA is used as an alternative optimization technique for the same set of three Test Circuits, a comparative chart is made to emphasize the importance of using MGA for the test simulation of VLSI combinational circuits.

<table>
<thead>
<tr>
<th>74Series of Test Circuits</th>
<th>Genetic Algorithm</th>
<th>Modified Genetic Algorithm</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Fault</td>
</tr>
<tr>
<td>1 74153</td>
<td>5</td>
<td>100 %</td>
</tr>
<tr>
<td>2 7485</td>
<td>46</td>
<td>90 %</td>
</tr>
<tr>
<td>3 74182</td>
<td>28</td>
<td>96 %</td>
</tr>
</tbody>
</table>
The Table 2.5 shows that from an optimization point of view the MGA outperforms the GA for all the three test circuits simulated using the proposed algorithm. For 74153, the MGA requires 0.8 sec less time as compared to GA. For 7485, the MGA requires 12 sec less time as compared to GA, for 74182 the MGA requires 7 sec less time as compared to GA.

2.3 FAULT SIMULATION IN SEQUENTIAL CIRCUITS

The Fault Detection process in sequential circuits involves the application of a sequence of patterns and the states of the Circuit under Test (CUT) advance step by step for time frames according to the applied test vector patterns [Rudnick E, et al., 1994]. Initially, the states of the flip-flops of the circuits are at the unknown state “X”. As the patterns are continued to be applied, the states of the flip-flops gradually set to the known states “1” or “0” and the faults are detected and are dropped by simulated patterns [Yu X, et al., 2005; Hsiao M. S, et al., 2000].

2.3.1 Fault Simulation using Genetic Algorithm

GA generates each test vector in a randomly generated initial population; each bit of an individual string represents the value applied to the primary input. The best vector is selected by considering their fitness values in each generation to act as an element of the final test vector set.

*Algorithm for Fault Simulation using Genetic algorithm*

The abstract form of the proposed algorithm is presented below.

1. Phase I
   Perform the good circuit simulation, i.e. Initialize
   The flip-flops with a set of input vectors.
   End;

2. Phase II
   While (stop ==0 | Remaining faults > 0)
   Apply the stream of input vectors to circuit inputs
   Perform the simulation for the input sequences
   Compare the obtained responses with the true responses
   End loop;
   End Phase II;
   End GA;
The proposed algorithm consists of two phases. The test vectors are generated to initialize the flip-flops in the first phase. Then test vectors are generated to detect as many faults as possible in phase 2. So, by first performing the true value simulation for several patterns and then followed by performing the fault simulation for the remaining patterns, the time-consuming faults can be dropped in the initial runs thereby saving the total simulation time. The first phase initializes the flip-flops so that the flip-flops remain in a known state when the fitness is evaluated for test vectors. So the test vector, which causes an increase in the fraction of the flip-flops set or reset, is awarded higher fitness value. It can also be called a good circuit simulation. In the second phase, the objective is to detect the maximum number of faults as possible so the fitness of the test vector in this phase is a measure of a number of previously undetected faults in previous generations. In order to have fault detection at the output, the propagation of fault effects is to be accomplished [Ibrahim W, et al., 2006].

**Results and Discussion**

The two-phase fault simulation technique has been successfully applied over two sequential circuit’s namely 4-bit parallel load Shift Register and Vending Machine Controller.

**Test Case I: 4-bit parallel load shift register**

The 4-bit parallel load shift register is shown in Figure 2.22.

![Figure 2.22: 4-bit parallel load shift register](image)

**Statistics:** 5 Primary Inputs, 4 Primary Outputs, 13 Gates, excluding flip-flop gates.

The circuit has been fully evaluated for the 60 stuck at faults instances and the proposed simulator produces complete fault coverage.
The details of the different parameters used in the following figures are:

Here Popsize: Population size, Pc: Probability of Crossover, Pm: Probability of Mutation, lchrom: Length of Chromosome, Maxg: Maximum Number of Generations, AN: Fitness Award Number, CT: Crossover Type.

(i) Effect of Population Size:

Specifications: Popsize: 3, 6,10,12,15, P_c: 0.6, P_m: 0.25, l_chrom: 5, Maxg: 30, AN: 1:10, CT: 2pt and Stuck fault remaining: 6, 3,0,0,0

![Figure 2.23: Effect of Population size variation on Fault Simulation](image)

The figure 2.23 shows that if the population size parameter of GA is varied, it directly affects the number of iterations required for full fault coverage. In order to reduce the number of iterations, the population size should be increased. **With Popsize of 15, the number of iterations required is 7, while for Popsize as 12, it got increased to 12, for Popsize of 10, it is 18, for Popsize of 6, it is 25 and for Popsize of 3 it is maximum with a number of generations as 2 and few faults still remain undetected.**

(ii) Effect of Maximum Number of Generations:

Specifications: Popsize: 4, P_c: 0.6, P_m: 0.25, l_chrom: 5, Maxg: 5,10,15,20, AN: 1:10, CT: 2pt and Stuck fault remaining: 19, 10, 3, 0.

The figure 2.24 shows the effect of a number of generations on that fault detection that as the generation count progresses more faults gets detected. **As the number of generation’s increases, more faults will get simulated and fitness of best the solution increases along with that. If a number of generations are 5, 12 faults still remains however if it has been allowed to run for 20, it gets reduced to 0.**
(iii) Effect of Probability of Crossover:

Specifications: Popsize: 7, P_c: 0.4, 0.5, 0.55, 0.65, 0.8, P_m: 0.1, lchrom: 5, Maxg: 25, AN: 1:10, CT: 2pt and Stuck fault remaining: 6, 4, 2, 0, 0.

The Figure 2.25 shows that the probability of crossover needs to be higher for a faster convergence to the solution.

*With P_c = 0.8, in 21 number of generations all faults got detected, however, with P_c = 0.4, 7 faults still remain.*

(iv) Effect of Probability of Mutation:

Specifications: Popsize: 7, P_c: 0.6, P_m: 0.01, 0.1, 0.3, lchrom: 5, Maxg: 25, AN: 1:10, CT: 2pt and Stuck fault remaining: 8, 3, 0.
Figure 2.26: Effect of Mutation rate variation on Fault Simulation

The figure 2.26, demonstrates that by merely a reversal of a bit of a vector stream may lead to the generation of new test sequences to have better fault coverage in a lesser number of generations. With $P_m = 0.01$, even after 25 iterations, 7 faults remain. However, for $P_m = 0.3$, in 21 iterations all faults got detected.

(v) Effect of Fitness Award Value:

Specifications: Popsize: 7, $P_c$: 0.6, $P_m$: 0.1, lchrom: 5, Maxg: 15, AN: 1:2, 1:4, 1:8, 1:10, CT: 2pt and Stuck fault remaining: 8,4,2,0

Figure 2.27 has been plotted to illustrate the effect of fitness award value on the number of iterations required. It has been shown that as the algorithm progresses if a higher fitness value gets awarded to those test vectors which has detected newer faults as compared to the already detected faults, then a better solution will be obtained in fewer generations. With $AN = 1:10$, a steep fall in remaining faults takes place. In 13 number of generations, all faults got detected. While for $1:2$, or $1:4$ even after 15 generations more than 10 faults still remain.

Figure 2.27: Effect of Award number variation on Fault Simulation
(vi) Effect of Adaptive Crossover and Mutation Rate:

Specifications: Popsize: 7, \( P_c: 0.4 \) to \( 0.8 \), \( P_m: 0.01-0.3 \), \( lchrom: 5 \), Maxg: 25, \( AN: 1:2 \), CT: 2pt and Stuck fault remaining: 0.

By using adaptive crossover and mutation rate, the GA explores the newer regions of the search space and hence prevents it to be caught in local minima as shown in Figure 2.28. Using adaptive \( P_c \) and \( P_m \), in 14 iterations all faults got detected which requires 20 iterations for the same circuit.

(vii) Effect of Different Crossover Techniques

Specifications: Popsize: 7, \( P_c: 0.5 \), \( P_m: 0.1 \), \( lchrom: 5 \), Maxg: 25, \( AN: 1:2 \), CT: 1pt, 2pt, use and Stuck fault remaining: 0.

The Figure. 2.29 has been plotted to demonstrate that it is possible to change the number of generations required for full fault detection by merely changing the crossover style.
Figure 2.29 shows that different crossover techniques require different number of iterations for fault coverage. It has been found that instead of following a fixed approach of a crossover between two parents more realistic results are obtained if some rule based technique like 2-point crossover and uniform crossover are adopted for an offspring generation.

*With Uniform crossover (uc), in 16 iterations all faults get detected while in 2-point crossover (2pt) it requires 18 iterations and in 1-point (1pt) crossover it needs 21 iterations to detect all faults.*

### 2.3.2 Fault Simulation using Modified Genetic Algorithm

To test the sequential circuits, a sequence of patterns is applied, and the states of the circuit advance step-by-step according to the applied patterns. The first pattern initializes the memory elements and the second pattern is used for fault simulation. The dependency of the next state on the previous state distinguishes the sequential fault simulator from the combinational fault simulator.

**Optimization of Test Vectors using Modified Genetic Algorithm**

The optimization process involves running several iterations of *Modified Genetic Algorithm (MGA)* wherein each iteration, the present set of test vectors got evaluated and thus evolves using control parameters of MGA [15].

The sequences of steps involved are outlined below:

(i) **Initialization of the Parameter Vectors**

It begins with a randomly generated initial population of an NP numbered, D-dimensional binary valued parameter vectors where each vector known as genome forms a candidate solution and is represented in eq. 2.4:

\[
\vec{X}_{i,G} = [x_{1,i,G}, x_{2,i,G}, x_{3,i,G}, \ldots, x_{D,i,G}]
\]  

\[\ldots(2.4)\]

Where \(X_{i,G}\) is the \(i^{th}\) vector of the population at the current generation.

(ii) **Mutation with Difference Vectors**

After first set of population elements has been initialized, MGA creates a donor vector \(\vec{V}_{i,n}\) for each population member (target vector) \(\vec{X}_{i,G}\) in the current generation through mutation. Five most frequently used mutation schemes are listed below from eq. 2.5 to eq. 2.9.
The indices \( i, j, r_1, r_2, r_3, r_4 \) and \( r_i \) are different from index \( i \) and their values are lying between [1, NP]. For each donor vector, these indices are randomly generated. \( F \) is the scaling factor whose value lies in the range (0.4, 1).

(iii) Crossover

To increase the potential diversity of the population, a crossover operation is performed. In this, the donor vector exchanges its components with the target vector \( \mathbf{X}_{i,G} \) to form the trial vector \( \mathbf{U}_{i,G} = [u_{i,1}, u_{i,2}, u_{i,3}, \ldots, u_{i,D}] \).

The uniform (binomial) crossover is used that is performed on each of the \( D \) variables whenever a randomly generated number between 0 and 1 is less than or equal to a positive constant \( C_r \), called crossover rate. \( C_r \) usually lies in the range (0.8, 1). It is represented as below in eq. 2.10.

\[
u_{j,G} = \begin{cases} v_{j,G}, & \text{if } \text{rand}_{i,j}(0,1) \leq C_r \text{ or } j = j_{\text{rand}} \\ x_{j,G}, & \text{otherwise,} \end{cases} \quad \text{...(2.10)}
\]

Where \( \text{rand}_{i,j}[0,1] \) is uniformly distributed random number, generated for each \( j^{th} \) component of the \( i^{th} \) parameter vector. \( J_{\text{rand}} \in [1, 2, \ldots, D] \) is a randomly chosen index which ensures that \( \mathbf{U}_{i,G} \) gets at least one component from \( \mathbf{V}_{i,G} \).

(iv) Selection

The crossover operation is followed by selection to determine whether the target or the trial vector survives to the next generation, i.e., at \( G = G + 1 \).

The operation is defined in eq. 2.11 as:
\[ u_{i,j,G} = \begin{cases} v_{i,j,G}, & \text{if } (\text{rand}_{i,j}(0,1) \leq C_x \text{ or } j = j_{\text{rand}}) \\ x_{i,j,G}, & \text{otherwise,} \end{cases} \] \quad \ldots (2.11)

Where \( f(\tilde{X}) \) is the objective function to be maximized [Chowdhury A, et al., 2010].

The overall computational efforts are depending heavily on fitness evaluation function. The MGA performance can be further improved by using a less accurate fitness function, but it has an adverse effect on the quality of the simulation [Skobtsov Y. A, et al., 2004].

**Results and Discussion**

The two-phase fault simulator technique has been successfully applied over various sequential circuits. The technique is explained using one test vector below.

**Test Circuit First (TCI): 4 Bit Parallel-Load Shift Register**

The circuit is shown in Figure 2.30.

![Figure 2.30: 4-bit parallel load shift register test circuit](image)

**Statistics:** 4 inputs; 4 output; 13 gates, excluding flip-flop gates.

The circuit has been fully evaluated for the possible stuck at faults instances and the proposed simulator produces complete fault coverage.
(1) **Effect of Population Size:**

*Specification: Pop size: 3, 6, 10, 12, 15*

![Figure 2.31: Effect of Population size variation on Fault Simulation](image)

As population size directly controls the variation between the qualities of the test vectors so it has a profound effect on the number of iterations taken during fault detection. With the increase in population size as shown in Figure 2.31 between the min and max boundary limits the unexplored regions of solutions space is represented and thus chances of global optima increase considerably.

*With Popsize = 3, even after 25 iterations 5 faults, still remains, however, for Popsize = 15, in 8 iterations all faults got detected.*

(2) **Effect of variation of Differentiation constant & Crossover constant:**

*Specifications: Differentiation constant: 0.2, 0.8, Crossover constant: 0.35, 0.55*

![Figure 2.32: Differentiation and Crossover constant value variation](image)
Evolutionary Algorithms for VLSI Applications

The Figure 2.32 shows that by varying the control parameters of MGA in the form of differentiation and crossover constants the convergence rate can be increased and the simulation time can be reduced [Liu, 2002]. This variation has a direct effect on the quality of trial vectors and can be explained as if MGA vector moves near to optimum it means during crossover it passes more genetic information to the offspring (trial vector). With the increase in the crossover, constant steep fall in remaining faults and faster generation of new members takes place.

(3) Effect of Different MGA mutation Techniques:
Specifications: MGA/rand/1, MGA/best/1, MGA/target-to-best/1, MGA/best/2 and MGA/rand/2

Here rand/1, best/1, target-to-best/1, best/2 and rand/2 are mutation strategies. The Figure 2.33 has been plotted to demonstrate the effect of changing the mutation style on the convergence rate of the problem. It has been found that more realistic solutions can be obtained if some rule based technique is adopted [Wu L. H, et al., 2006], for trial vector generation and the best results for sequential fault simulation are obtained here by using the technique.

Five different mutation techniques are tested for their effectiveness in fault simulation. The following are the results obtained for each of them.

The best-1 mutation strategy takes 17 iterations to detect all faults, target-to-best-1 takes 21 iterations, rand-2 takes 22 iterations, best-2 takes 24 iterations and rand-1 mutation strategy takes 28 iterations to detect all faults.

Figure 2.33: Different Mutation strategies performance in Fault Simulation
The performance of MGA is compared with GA. The results are compiled in the form of Table 2.6 which shows that better performance in terms of simulation time and fault coverage is achieved by using MGA with Population size = 10.

Table 2.6: Comparison of MGA with GA for Sequential circuits

<table>
<thead>
<tr>
<th>Test Circuits</th>
<th>Genetic Algorithm</th>
<th></th>
<th>Modified Genetic Algorithm</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Time (sec)</td>
<td>Fault</td>
<td>Generation</td>
<td>Time (sec)</td>
</tr>
<tr>
<td>1 4 bit Register</td>
<td>10</td>
<td>100%</td>
<td>18</td>
<td>9.3</td>
</tr>
<tr>
<td>2 Vending Machine</td>
<td>52</td>
<td>95%</td>
<td>70</td>
<td>48</td>
</tr>
</tbody>
</table>

The MGA for 4-bit register uses 0.7 sec less as compared to GA and 4 sec less as compared to GA for Vending Machine Controller. In terms of a number of generations, for 4-bit register, MGA requires 1 fewer generations and 4 fewer generations for Vending Machine Controller.

The GA-based fault simulator is effective in generating optimum results, but due to its inherent parallel nature of working on all population members the overall development of the population takes place. As the circuit components increase, the time of computation also increases and fault simulation on the complete circuit for all population members becomes a tedious task. So, another technique in the form of hierarchical fault simulation is used.

2.4 HIERARCHICAL FAULT SIMULATION TECHNIQUE

In hierarchical fault detection strategy, the main goal is to speed up test generation by using module level evaluation [Min B, et al., 1993]. The description of the circuit in terms of its hierarchy and the function of each component or module can be obtained from the language or netlist in which it is implemented [O'Dare M. J, et al., 1995]. A speedup can be achieved in test generation if modules can be evaluated without referring to the lower levels of circuit hierarchy.

The majority of EDA tool editors for e.g. Xilinx and Mentor Graphics are coming with lots of cell libraries, which provide the necessary detail of the implementation of a block used
in the design. This higher level of use in test generation is basically helping us in saving the
time needed to expand the design and exploring the faults incurred in it. The circuit shown
in Figure 2.34 is used to describe the hierarchical description of the circuit. The X is the top
level module consisting of two Y modules and a Z-module at a lower level of the hierarchy.
Three Z-modules form a Y module at the third level of the hierarchy. Each Z-module is
finally implemented using three gates at the fourth and lowest level of the hierarchy.

![Figure 2.34: Demonstration of Hierarchical Expansion](image)

It is a tree type structure with X representing the root of the tree while the gates are the
leaves of this tree representation. If only leaves have been selected, the circuit is described
by a flat gate level model. If a fault is provided, the whole circuit can be reconfigured
dynamically according to the location of the fault.

Hierarchical test generation is faster because operations like backtracking and fault
propagation can be performed faster on high-level models than on corresponding gate level
models.

**Algorithm for Hierarchical Fault Simulation**

To achieve this simulation method must be provided with the functionality of each block at
the architecture level. The technique used for expansion followed by detection is based on
the points described below:

1. Test patterns are generated for circuits by expanding modules (adders, multiplexers,
decoders etc.) to a gate level realization in order to detect faults within the target module,
i.e., the hierarchy must be expanded so that the simulation technique would be able to look
at the design modules for their fault simulation. All the modules if operate correctly in terms
of their functions, they represent that the expected and the obtained response are exactly
matched with each other.
2. An HDL i.e., Verilog is used to enter the modules by using the structural style of coding. In this, a big design is further described into its constituent designs and all resides in the same library. Whenever a sub-module is referred to, its description is simply called in from the library and the interconnecting nodes and circuit forming logic gates are discovered thereafter. This style of modeling is referred as the structural style of modeling in which the hierarchy is preserved and module-to-module interaction is fully achieved. The desired test vectors are simply transferred to the module under test during its mapping and the obtained vector can be stored in a table.

3. In GA, the main system functions must interact with each other to rapidly evolve the test set as the fault simulation function interact with RT i.e. Reference Table, before the fitness function assigns it an appropriate fitness value. It implies that the algorithm is coded such that its main constituent blocks are interacting with each other and updating the RT on the basis of the fitness obtained for each test vector such that the fittest vector is chosen and the chances of crossover and mutation between the fittest binary strings have been enhanced.

4. The maximum test generation takes place at the highest level of abstraction, i.e., Architecture level where the complete design is described at the block level. It is because during simulation of the complete design the intermediate node values are not accessible and hence the final values of the primary outputs are always compared with the desired responses. The fault detection procedure must be such that the faulty node value must be propagated to the primary output node only then the fault is possible to be detected. This may be accomplished within the same simulation routine, by employing an identical procedure to the fault simulation used in the original system. This can only be achieved if the fault simulation process is provided with the logical function of each of the modules at the architectural level.

5. The fault simulation is performed at the module level, one module at a time is expanded to detect any fault occurring on internal nodes. It starts from the assumption such that the complete design is fault free and the final responses are to be monitored for their correctness and the moment some faulty responses are obtained; the hierarchy needs to be expanded to the module level. For the detection of a particular fault, different test vector patterns are applied at its primary input and patterns for which fault has been excited and can be obtained at the output are taken as test vectors for that particular fault.
Consider a circuit as shown in Figure 2.35 with its hierarchy. The leaves of the tree represent gates and all other nodes are modules. The tree has 4 level hierarchy and each module has three leaves (children) i.e., \( k = 3 \). The root is on the 0th level and leaves are on the 3rd level. Let \( n_{ij} \) be the \( j_{th} \) node on the \( i_{th} \) level in circuit hierarchy and \( S_{ij} \) be the evaluation cost of the module during test generation. Then the cost, \( C_{3,4} \), of test generation as shown in eq. 2.12 for a target fault in \( n_{3,4} \) is the sum of the cost of module evaluations.

\[
C_{3,4} = (S_{2,1} + S_{2,2}) + (S_{1,1} + S_{1,2}) + (S_{3,4} + S_{3,5} + S_{3,6}) \quad \ldots(2.12)
\]

Since each gate has \( f_g \) faults, the cost of test generation for faults in gate \( n_{3,4} \) is \( f_g \times C_{3,4} \). Similar cost for remaining gates at the same level of the hierarchy can be calculated. The total cost \( C_3 \) is the sum of all these costs as shown in eq. 2.13.

\[
C_i = f_g'(4^i - 1) \sum_{j=1}^{k} S_{i,j} + 4^i (4 - 1) \sum_{j=1}^{k} s_{2,j} + 4 \sum_{j=1}^{k} s_{3,j} \quad \ldots(2.13)
\]

It can be generalized for \( L \) level hierarchy with \( k \) children per module.
The cost of generating a test for each gate is given as a product of the number of faults for that gate $f^g$ and the cost of generating a test for a single fault in that gate is $C_{ij}$ where $i$ and $j$ provide the two-dimensional address for a gate of modules within a hierarchy and $S_{ij}$ denotes the evaluation cost of the module. Then the total cost is the sum of the cost of test generation for all gates individually.
The generalized equation describing the cost of generating a test for a fault is given in eq.2.14 mentioned below:

\[
C_L = f_g^{*} \left( K^{L-1} (K-1) \sum_{j=1}^{K} S_{i,j} + K^{L-1} (K-1) \sum_{j=1}^{K} S_{2,j} + \cdots K^{L-1} \sum_{j=1}^{K} S_{L,j} \right)
\]

\[= f_g^{*} (K-1) \sum_{i=1}^{L} \left( K^{L-1} \sum_{j=1}^{K} S_{i,j} \right) + f_g^{*} \sum_{j=1}^{K} S_{L,j} \]

\[\text{.... (2.14)}\]

Here \( L \) represents the number of levels of expansion needed to transform the module into a gate level realization and \( k \) represents the number of modules that are produced by each expanded module. Finally, the total number of gates \( G \) within the circuit is equal to \( KL \).

**Results and Discussion**

If the fault detection is performed at the gate level, it is obviously time-consuming since it requires a lot of test vectors to be evaluated and hence not reliable also in terms of time required to detect total faults. In terms of cost, it is very costly as the testing equipment tends to be a complex one and difficult to be designed. However, *if the detection is performed at the module level, then the time and hence speed can be achieved*. In other words, without testing a maximum of test vectors full fault coverage can be achieved. The circuit under consideration is an ISCAS-85 Benchmark circuit, i.e. 7485, a 4-bit magnitude comparator with following statistics: 11 inputs, three outputs as shown in Figure 2.36.

![4-bit Magnitude comparator Test circuit](image)

Figure 2.36 4-bit Magnitude comparator Test circuit

The cell-based approach has helped in comparing the responses since the simulation of the fault-free circuit can be performed quickly by referring to the library. Figure 2.73 describes that for the circuit under test i.e. 7485, full fault coverage has been achieved.
In most of the circuits, the fitness function is used to evaluate the circuit. It chooses the fittest member of the population, but sometimes the strategy has to change, that first the same pair of test vectors are first used to find the possible stuck at faults and then delay faults are only considered, by adopting this technique the number of iterations can be reduced by having a directed approach for the GA. In order to prevent the algorithm from falling into local optima, the control parameters are continually judged to verify that the maximum fitness and the average fitness are variables as shown in Figure 2.38. The use of control parameters ensures that maximum fitness for a given population is or can be increased if the higher award number, i.e. AN, varying crossover probability, i.e. $P_c$ and mutation probability i.e. $P_m$ can be employed. The main focus of fault simulation is on adopting an approach resulting in a hierarchical expansion of the circuit under test on the basis of the response obtained from different blocks by comparing them with their expected values. Those which give correct results need not any expansion to their lower levels of abstraction while the remaining can be expanded to their gate level details to fine tune the faults possibilities.
The technique is fully adapted to the circuit and is able to find all the 154 stuck at faults and 52 delay faults in it. If no improvement is taking place for three successive generations, then fault detection can be increased by varying the mutation rate as described in Figure 2.39.

Figure 2.39: Fault Simulation with varying mutation rate variation

Moreover, it has also been observed that the detection rate can be further enhanced if the AN i.e. Award Number, can be increased for the undetected faults with respect to the detected ones. With the increase in the ANs the probability of choosing only those test vectors which detect mostly the undetected faults gets increased by manifolds. This strategy serves a dual purpose: first it detects the faults in less number of generations and secondly it reduces the size of the test set also. But this process does not repeat for long, generally after a fixed increase in AN. It has been observed that no significant gain in test size reduction takes place if this increase continues thereafter also as shown in Figure 2.40.

Figure 2.40: Effect of Award Number variation on Fault Simulation
It has been emphasized that the fault detection rate can be increased if the fault modules are identified at the architecture level itself and instead of finding the faults at the global level, the block level is targeted. Moreover, the use of GA helps in finding the test patterns that detect a high number of previously undetected faults. By modifying the award numbers and adopting two points crossover and varying mutation rates, a compact test vector set has been obtained in the minimum number of generations.

### 2.5 SUMMARY

Various test generation mechanisms using genetic algorithms have been presented, including a Universal Record table that primarily helps the genetic algorithm in an effective search of the solution space. The genetic search has been adopted here because of the randomization in its approach thus not permitting the solution to be limited to a local minimum. Fault simulation is targeted in both combinational and sequential circuits. Different types of faults such as stuck-at faults, bridge faults, and delay faults are considered. The binary encoding based test vectors are used to apply a stimulus to test circuits. The fitness evaluation is based on the number of undetected faults of previous generations. The GA-based fault simulator is designed which iterates in the sequence of generations and generates new population members from existing using crossover and mutation operators for combinational circuits. The effect of the increase in the number of generations due to increase in population size is studied. The delay faults are targeted along with stuck-at faults in the same test circuits. In order to detect them, a pair of patterns is required, one excites the circuit nodes and another simulates them. For a test circuit with 30 stuck at faults and 20 delay faults, it requires 15 iterations. Bridge faults are also tested along with stuck at and delay faults and the same test vectors are used to simulate them.

After this, an MGA is designed and tested for the fault simulation problem. At each generation, it uses mutation and crossover operations to generate a trial vector for each target vector in the current population. By modifying the award numbers and adopting two-point crossover and varying mutation rates, it has been observed that the system tries to be an adaptive one instead of focusing on only fault detection.

The 74153 test circuit is considered and in 9 generations, it detects all the stuck at and delay faults. Similar results are obtained for 7485 and 74182 benchmark problems. Various
control parameters such as differentiation constant and crossover constant are varied to test their effect on a number of generations.

Fault simulation in sequential circuits is also evaluated using GA. It is a two-phase fault simulation process. In the first phase, flip-flops are initialized and in the second phase, fault simulation progresses. Two phases involve true simulation followed by fault simulation to detect the possible faulty joints are open in a sequential circuit. The technique is tested on 4-bit parallel load shift register and vending machine controller. After this MGA is used for sequential fault simulation. The effect of population size, differentiation constant, and crossover constant, different mutation strategies are tested. In comparison with GA, MGA uses 0.73 fewer seconds for 4-bit register and 0.48 fewer seconds for vending machine controller.

Finally, another approach in the form of hierarchical fault simulator is tested. Test generation can also be implemented as a hierarchical based genetic technique, which uses a dynamically updated reference table (RT). The fault detection rate can be increased if the fault modules are identified at the architecture level itself and fault simulation is done at the block level instead of at the global level. The genetic technique has been very well used for test pattern generator (test, simulator) at the gate level, now it has been expanded to one abstraction level above to module level. The technique is tested on 74 series of benchmark problems with 11 inputs and 33 gates. It detects all 154 stuck at and 52 delay faults in 35 iterations.