PLAGA: A Highly Parallelizable Genetic Algorithm for Programmable Logic Arrays Test Pattern Generation

Alfredo Cruz, PhD
Adjunct Professor, Department of Electrical and Computer Engineering
Universidad Politécnica, Hato Rey, PR
E-mail: across@coqui.net

Sumitra Mukherjee, PhD
School of Computer Information Science
Nova Southeastern University, Fort Lauderdale, FL
E-mail: sumitra@scis.nova.edu

ABSTRACT

An evolutionary algorithm (EA) approach is used in the development of a test vector generation application for single and multiple fault detection of shrinkage faults in Programmable Logic Arrays (PLA). Three basic steps are performed during the generation of the test vectors: crossover, mutation and selection. A new mutation operator is introduced that helps increase the Hamming distance among the candidate solutions. Once crossover and mutation have occurred, the new candidate test vectors with higher fitness function scores replace the old ones. With this scheme, population members steadily improve their fitness level with each new generation. The resulting process yields improved solutions to the problem of the PLA test vector generation for shrinkage faults. PLA testing and fault simulation computational time is prohibitive in uniprocessor machines, however PLAGA is well suited for powerful parallel processing MIMD machines with vectorization capability.

SINOPSIS

Este artículo presenta un algoritmo genético que genera un vector de prueba para detectar fallas de encogimiento ("shrinkage faults") en Arreglos Lógicos Programables (ALP), Programmable Logic Arrays (PLA).

Un operador mutante nuevo es introducido para complementar los operadores genéticos tradicionales de cruzamiento entre parejas ("crossover") y mutación. Este operador mutante ayuda a identificar soluciones en relativamente pocas generaciones al aumentar la distancia Hamming entre las posibles candidatas. El nivel promedio de fortaleza de la población mejora con cada generación cuando los vectores de prueba nuevos, con mayor puntuación en la función de fortaleza, reemplazan los viejos. Se identifican soluciones al problema de la generación de vectores de prueba de ALP para fallas de encogimiento.

El enfoque basado en algoritmos genéticos se ajusta bien para su implementación en máquinas de procesamiento en paralelo con capacidad de vectorización. Esto es significativo debido al hecho de que las pruebas de ALP y simulación de fallas son muy costosas computacionalmente en máquinas de un procesador.

I- INTRODUCTION

The key technological advantage of using PLA in integrated circuit technology relies on the straightforward mapping between the symbolic representation and its physical implementation. At the physical implementation level, it is easy to map binary format into layout because it does not involve any difficult steps such as placement and routing which may be necessary for *random logic* implementation. The number of different basic building units for PLA is small (AND, OR and NOT) whereas the number of different basic elements for random logic could be much larger. However, this simplified structure does not ease the difficulty for test pattern generation and fault simulation in PLA.

PLA testing has attracted the attention of many researchers in recent years [1, 2]. Genetic and evolutionary algorithm-based solutions have been proposed [3, 4, 5, 6].

Algorithms proposed in [7] formulate the PLA test generation by using the sharp (T) operation. The T operation is widely used for logic manipulation algorithms (ESPRESSO II). Several other algorithms have also been proposed for PLA testing using the T operation, but they tend to be computationally expensive. A major disadvantage of this operator is that backtracking is necessary when a test cannot be found. The computational overheads required by

backtracking can be prohibitive. Bose [8] proposed an algorithm using the T operation for extra devices that utilizes the Quine-McCluskey method for testing missing devices. The memory requirements for this algorithm rise exponentially as PLA size increases. An algorithm reported in [9] assigns a proper logic value in the specified inputs of a potential vector. The aim is to achieve path sensitization by deselecting product lines connected to output functions. This pruning method uses a modified version of the T operation. Robinson [9] extended the work of [8] by using a weighted pruning approach that attempts to detect hard-to-detect faults more easily. The latter, however, may fail to find a test even if one exists.

Smith [10] suggests simplifying the algorithm by generating a test for every fault. This results in considerably larger test vectors. Hence, a minimal test set is not guaranteed. One of the disadvantages in this approach is the backtracking that could occur when the test is chosen and fails to propagate.

Other approaches have been developed for generating a PLA test set. For example, the *Design-for-testability* (DFT) uses extra components to facilitate the PLA test, making test generation unnecessary in some instances. All these methods employ additional hardware, which means greater costs, and potential degradation of PLA performance. Clearly, present DFT methods have not addressed the problem adequately [11].

A. *Basic Notation and Definitions*

The PLA consists of input lines (uncomplemented and complemented) and product term lines. The intersections between product lines and input bit lines or between output function lines and product term lines are called *crosspoints.* Each product line is used to realize an implicant (product term) of the given function by placing appropriate crosspoint devices into what is known as the AND plane.

Figure 1 shows a simple schematic of a PLA, implementing the two switching functions:

$$f_1(x_1,x_2,x_3,x_4)=(x_1 \text{ AND } x_2) \text{ OR } (x_2 \text{ AND } x_3)$$

$$f_2(x_1,x_2,x_3,x_4)=(x_1' \text{ AND } x_2) \text{ OR } (x_1 \text{ AND } x_2' \text{ AND } x_3')$$

This PLA has four inputs (x_1, x_2, x_3, x_4), four product terms (m_1, m_2, m3, m_4), and two output functions (f_1, f_2).

The following definitions apply to the discussions that follow.

Definition 1:
Hamming distance: The number of bit positions in which two product terms hold non-don't care values that are different is called the Hamming distance, d_H.

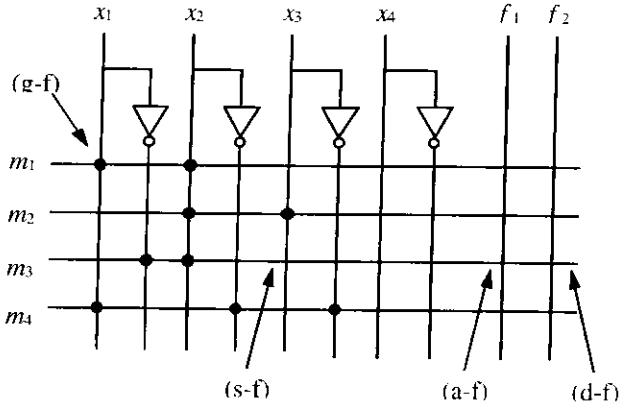

Figure 1: *A PLA Schematic*

For example, in Figure 1 the hamming distance between m3 and m4 is one, i.e. these terms differ in bit position x_3 only.

Definition 2:
X-complementary: The set $\{A, B\}$ of two terms, where $A=\{a_1,\ldots,a_n\}$, $a_I \in \{0,1\}$, and $B=\{b_1,\ldots,b_n\}$, $b_i \in \{0,1\}$ will be called x-complementary with respect to the product term $\{x_1,\ldots,x_n\}$, if the following conditions are satisfied:

$$\begin{cases} a_i = b_i = 1 & \text{if } x_i = 1 \\ a_i = b_i = 0 & \text{if } x_i = 0 \\ a_i = b_i' & \text{if } x_i = x \end{cases}$$

For example, the product term 1X0X has two pairs of minterms that are x-complementary. These are: {(1000, 1101), (1001, 1100)}.

Definition 3:
Number of minterms: The size Ω is the number of canonical minterms contained or represented in a product term (m_i), which is equal to 2^k, where k is the number of don't care inputs in the product term.

For example, in Figure 1 m_1 has $2^2 = 4$ number of minterms.

II- FAULT MODELING

In testing digital circuits, the most commonly considered fault model is the stuck-at fault (i.e., *s-a-0* or *s-a-1*). However, because of the PLA's array

structure the stuck-at fault alone cannot adequately model all physical defects in a PLA [12]. Therefore, a new fault class model, known as the *crosspoint* model is used. The unintentional presence or absence of a device in the PLA causes a crosspoint fault.

Different types of faults are shown in Figure 1. The symbols (*g-f*), (*s-f*), (*a-f*), and (*d-f*) denote the growth, shrinkage, appearance and disappearance faults respectively.

The focus of this paper is on the use of genetic algorithms for the generation of test vectors for shrinkage faults.

III- THE SHRINKAGE FAULTS

To detect extra crosspoints in a PLA, it is important to understand that shrinkage faults can occur with the presence of an unintended device in an unspecified input (*don't care value*). The presence causes the ON-set (i.e., the minterms) to shrink, and consequently the OFF-set (i.e. the maxterms) to grow.

The fault sensitization of an extra device in a faulty PLA produces a 0 at the output of the AND gate, whereas for fault-free PLA produces a 1. To test for shrinkage faults (an extra device) one or two test vectors are often generated. However, in some cases, more test patterns are needed.

The above discussion leads to the following rules that must be established for the generation of test vector, for shrinkage faults. It is assumed that the PLA under test is irredundant. In other words, if an implicant is eliminated then the function will change.

1- If a product term is isolated, i.e., the minterms covered by the isolated product term are free, then the shrinkage test vector is chosen as follows:

   (a) When the size Ω is equal to 1, then this minterm is chosen as a test vector. For example a product term without don't care inputs, i.e. 101011
   (b) When the size Ω is equal to 2, then both elements (minterms) will constitute a complete test vector.
   (c) When size Ω is > 2 (i.e., all minterms are free), then the minimum test vector is obtained by choosing a pair of minterms with d_H = k, where k is the number of don't care values.

2- If a product term is non-isolated, i.e. with minterms covered by more than one product term, then the shrinkage test vector is chosen in the following way:

   (a) If the size Ω is equal to 2, then one minterm is free, since it is assumed that the PLA is irredundant. In that case the free minterm constitute a shrinkage test vector. Hence this vector will detect a fault, either in the complement line or uncomplement line.
   (b) When the size $\Omega > 2$ and some minterms are free, the minimum test vector may result in more than 2 test vectors.

Proposition 1. The upper bound for a complete test vector set for a non-isolated product term is 2 if more than half of the minterms are free.

Proof: For a product term with 2^k minterms there are $\frac{2^k}{2}$ x-complementary pairs. Furthermore, if there are at least $\frac{2^k}{2}+1$ *free* minterms then there are at most $\frac{2^k}{2}-1$ *bounded* minterms.

Since $\left(\frac{2^k}{2}+1\right)-\left(\frac{2^k}{2}-1\right)=2$, there exist at least 2 complementary minterms. Therefore, a complete test exists.

Proposition 2. The upper bound for a complete test vector set for non-isolated product term is k if a bounded minterm has its k adjacency minterms free.

Figure 2 illustrates the four possible ways to obtain complete test vectors of a product term in a k-dimensional subspace. It also applies to product terms in an n-dimensional subspace with k unspecified values.

- Figure 2 (a) shows a minimal test vector set of size 2 when two minterms with a d_H = k are both free. The complete test vector can be any x-complementary pair *e.g.*, (V_0, V_{15}).
- Figure 2 (b) shows a complete test vector set of size 3 when one of the x-complementary pair (V_{15}) is bounded (bounded minterms are marked with a 'ρ'). In this case the two minterms adjacent to the bounded minterms (V_{11}, V_{13}) are valid test vectors if they are free. Therefore, the complete test vector is (V_0, V_{13}, V_{14}).
- Figure 2 (c) shows a complete test vector set of size 4 when two x-complementary minterms are bounded, but they both have two adjacent minterms which are unbounded. The complete test vector is (V_1, V_2, V_7, V_{11}).
- Lastly, Figure 2 (d) shows a complete test vector set of size k when neither of the cases are possible, The complete test vector is (V_1, V_2, V_4, V_8).

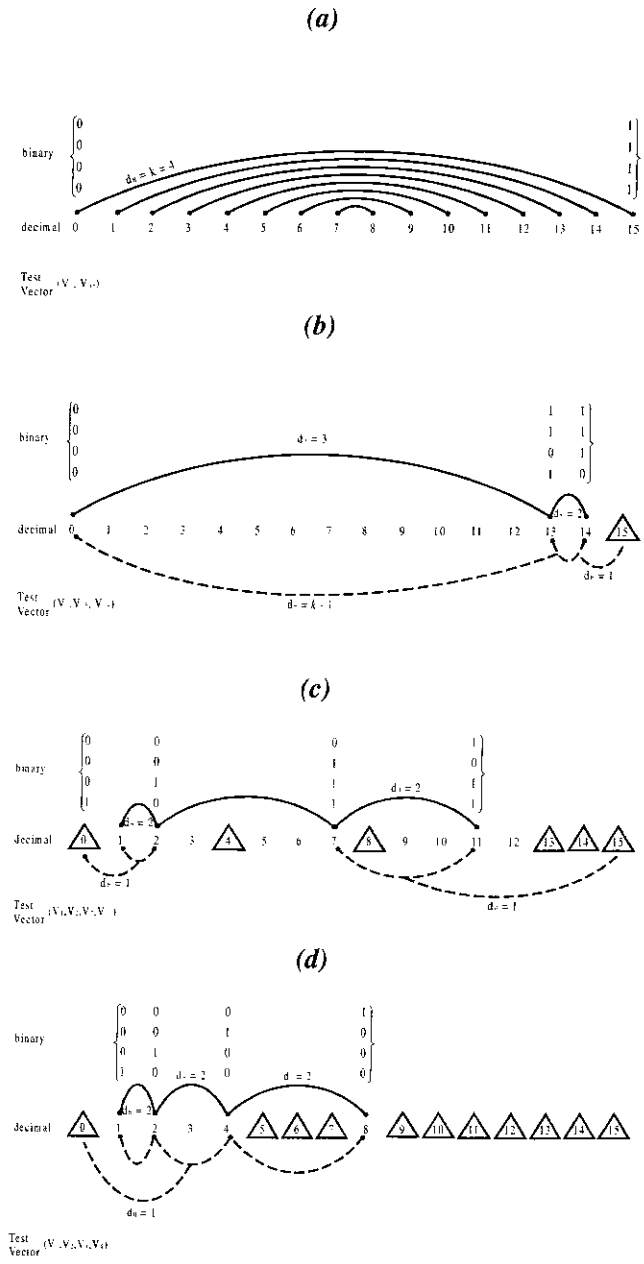

Figure 2: *Complete Test Vectors for Shrinkage Faults: (a) Test Vector of Size 2. (b) Test Vector of Size 3. (c) Test Vector of Size 4. (d) Test Vector of Size k*

Since the growth and shrinkage test vectors account for the largest number of test vectors, the upper bound for non-isolated product terms is given in terms of these faults:

$$\sum_{1}^{m} k + \sum_{1}^{m} \left(n - \log_2 \Omega\right)$$

where k is the number of don't care values, m is the number of product terms, and $n - \log_2 \Omega$ is the number of care values in each product term.

IV- THE PLA GENETIC ALGORITHM

Evolutionary Algorithms (EA) are search and optimization procedures that find their origin in the biological world. They mimic the processes of biological evolution with its ideas of natural selection and survival of the fittest to provide effective solutions for optimization problems. A genetic algorithm is an iterative procedure that consists of a constant size population of individuals, each individual represented by a finite string of symbols known as *genome*, encoding a possible solution in a given problem space. This space, referred to as the *search space*, comprises all possible solutions to the problem at hand. The symbols used most often are binary because of the computational advantages. For the test generation problem we will associate a candidate test vector with a population genome and a candidate vector set with a given population generation.

The basic genetic algorithm, where P(t) is the population of strings at generation t is given below:

```
procedure genetic algorithm
{
  set time t := 0
  select an initial population P(t)
  while the termination condition is not met, do:
  {
    evaluate fitness of each member of P(t);
    select the fittest members from P(t);
    generate offsprings of the fittest pairs
          (using genetic operators);
    replace the weakest members of P(t)
          by these offsprings;
    set time t := t+1
  }
}.
```

Selection alone cannot introduce any new individuals into the population, i.e., it cannot find new candidate test vectors in the search space. Selection is done on the basis of relative fitness and it probabilistically eliminates from the population those candidate test vectors which have relatively low fitness. Recombination, which consists of mutation and crossover, imitates sexual reproduction.

Crossover is performed with crossover probability P_{cross} between two selected strings, called *parents*, by exchanging parts of their genomes (i.e., encoding) to form two new individuals, called *offsprings*. It is implemented by choosing a random point between 1 and the string length (δ) minus one [1, δ – 1] in the selected pair of parents and

exchanging the substring defined by that point (i.e., swap the tail portion of the string) to produce new offsprings. That is, all the information from one parent is copied from the start up to the crossover point, then all the information from the other parent is copied from the crossover point to the end of the offspring (chromosome). The new chromosome thus gets the head of one parent's chromosome combined with the tail of the other. Figure 3 illustrates crossover with parents '10101011' and '11111001,' and crossover occurring after the third bit.

The crossover operation, unlike previous operators used in PLA test generation, does not use lookups or backtracking. Crossover is both simple and efficient. This operation enables the evolutionary process to move towards optimal solutions in the search space. The usefulness of crossover is due to the combination of better than average substrings coming from different individuals [13].

Mutation probabilistically chooses a bit and flips it. Mutation can be applied to population members with a frequency P_{mut} of around 0.01. The usual interpretation of bit mutation rate is the following:

```
For each string (candidate test vector) in the population
{
 for each bit within the string
     {
        generate a random number r between 0 and 1
        if (r > p_mut)
              toggle the bit
     }
}
```

Mutation is needed because if selection and crossover together search new solutions, they tend to cause rapid convergence and there is a danger of losing potentially useful genetic materials, such as 0s and 1s at particular location of the specified values of the candidate test vector under evolution. Notice in Figure 3 that for bit positions 1, 3, 5, 6, and 8 the children retain the same value as their parents, even though crossover occurred. An extreme instance occurs when both parents are identical. In such cases, crossover cannot introduce diversity in the offspring. In order to introduce diversity and to avoid search stagnation, bit mutations are allowed. However, mutation frequencies have to be low; otherwise the search tends to generate a random walk.

A new mutation strategy using a template operator is introduced in this work to help accelerate the search of the minimal test vectors with a Hamming distance $d_H = k$, where k is the number of don't care values. This operator helps increase the Hamming distance between the parents and their children. It also helps avoid the loss of genetic

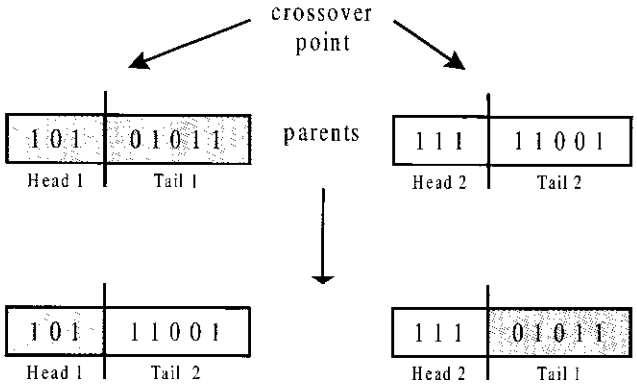

Figure 3: *The effect of the crossover operation*

material and hamming convergence. The template has the value 1 in the position of the unspecified values (don't cares), and 0 in the specified values of the candidate test vector under consideration. The 1's are used to toggle the alleles in their respective bit position on the candidate test vector in evolution. The 0's leave the alleles intact in their respective bit position. The use of the template does not affect the fitness value of the candidate test vector. On the contrary, this strategy ensures that **PLA**GA evolves a population dominated by individuals which are close to the optimum in the phenotype space, but far apart in terms of Hamming distance. For example, the template for the product term 10X01X is 001001. If a candidate test vector is 001010 after the crossover operation, then it becomes 000011 after mutation using the template is applied. This operator is applied to very few individuals in each generation randomly. This ensures that every allele bit position is defined and that the population spans the entire code space of the problem.

In summary, the effect of this operator makes **PLA**GA more resistant to deception, simplifies the encoding of the genetic algorithm and increases the step-size (Hamming distance) between a parent and its offspring.

The fundamental steps to generate a minimal test vector set for a PLA constitute a minimal *covering* problem. This problem is known to be NP-complete. Faster heuristic methods to find a near minimal test set are more desirable than optimal methods from a computational point of view. Figure 4 shows the Personality Matrix (PM) (inputs = 4, product terms = 4, functions = 1)-PLA implementing a switching function:

$$f(x_1, x_2, x_3, x_4) = (x_4')\,\text{OR}\,(x_1'\ \text{AND}\ x_3)\,\text{OR}\,(x_1'\ \text{AND}\ x_3'\ \text{AND}\ x_4)\,\text{OR}\,(x_1\ \text{AND}\ x_2'\ \text{AND}\ x_3\ \text{AND}\ x_4)$$

This simple example will be used to illustrate the **PLA**GA algorithm for the generation of a test set for shrinkage faults. The PM is the cubical representation of the sample PLA.

PLAGA uses the GA to find the test vectors that excites the fault effect to the primary output. Each candidate test vector of the PM represents the values applied to the primary inputs of the AND gate under consideration.

The following GA parameters are used for testing shrinkage faults of the sample PLA:

- Uniform Crossover Single Cut Point
- Number of generations : Until a minimal test set is found
- Size of Population : 8
- Crossover Probability : 1.0
- Mutation Probability : 0.01
- Mutation Probability with Template : 0.25

To apply the GA more efficiently the order of the product terms of the PM should be arranged in a decreasing order of *don't care* values. The product terms with more *don't cares* should be at the top of the matrix. This configuration allows a multi-target test vector detection for multiple product terms simultaneously. Genetic algorithms are inherently parallel since evolution takes place with individuals acting simultaneously in spatially extended domains [14, 15, 16].

PLAGA begins, at generation 0, with a population of randomly created individuals (candidate test vectors). For each generation, each individual in the population is evaluated for fitness.

m_1	X	X	X	0
m_2	0	X	1	X
m_3	0	X	0	1
m_4	1	0	1	1

Figure 4: *PLA ordered by the number of don't care values*

The fitness function is calculated as the number of bits in the individual that matches the corresponding bits in the product term. If a minterm of the population makes the product term (with 4 bits) under consideration **true**, then the minterm has a fitness value of 4. For example, for the product term m_4 (1011) of Figure 4, 0100 has fitness 0, 0101 has fitness 1, 0111 has fitness 2, 0011 has fitness 3, and 1011 has fitness 4 and is a solution.

A maximum value of 4 in this example means that the candidate test vector match each allele (including both the care and don't care values) of the product term under consideration (shown by ⇑ in Table I) and consequently it is a possible test vector. These test vectors are compared for uniqueness. The strings 1 (0100), 2 (1110), 3 (0100) and 6 (1100) are free test vectors for product term m_1. Notice that strings 1 and 3 have the same value (0100). Free minterms are represented with a '○' in Table I. These three vectors do not constitute a complete test set for m_1 since a test for the uncomplemented bit line b_2 is not feasible. However, it is possible to derive a minimal shrinkage test set for m_1 of size 2 if the x-complementary minterm of any of the four test vectors is found. The string 5 (0011) is a free

TABLE I: *GENERATION 0*

String	Population				Minterms				Fitness	# of copies reproduced	Mate Pool (cross point site shown)	Mate #	Crossover P_{cross}= 1.0					Mutation w/template (twice each generation) & P_{mut}= 0.01			
	X_1	X_2	X_3	X_4	m_1	m_2	m_3	m_4	m_1												
(1)	0	1	0	0	④	–	–	–	4	1	0 1 0 | 0	6	0	1	0	1		0	1	0	1
(2)	1	1	1	0	④	–	–	–	4	1	1 | 1 1 0	8	1	1	0	0		1	1	0	0
(3)	0	1	0	0	④	–	–	–	4	2	0 1 0 | 0	4	0	1	0	0	$\leftarrow t_1 \rightarrow$	1	0	1	0
(4)	1	0	1	1	3	–	–	④	3	0	0 1 0 | 0	3	0	1	0	0		0	1	0	0
(5)	0	0	1	1	3	④	–	–	3	2	0 0 | 1 1	7	0	0	0	0		0	0	0	0
(6)	1	1	0	0	④	–	–	–	4	2	0 0 1 | 1	1	0	0	1	0	$\leftarrow t_1 \rightarrow$	1	1	0	0
(7)	0	0	0	1	3	–	④	–	3	0	1 1 | 0 0	5	1	1	1	1		1	1	1	1
(8)	1	0	1	1	3	–	–	④	3	0	1 | 1 0 0	2	1	1	1	0		1	1	1	0
					⇑			*													

$$m_1 = \begin{cases} \text{Sum} & 28 \\ \text{Average} & 3.5 \\ \text{Max} & 4 \\ \text{Min} & 3 \end{cases}$$

TABLE II: GENERATION 1

String	Population				Minterms				Fitness	# of copies reproduced	Mate Pool (cross point site shown)				Mate #	Crossover $P_{cross} = 1.0$					Mutation w/template (twice each generation) & $P_{mut} = 0.01$			
	X_1	X_2	X_3	X_4	m_1	m_2	m_3	m_4	m_2															
(1)	0	1	0	1	3	3	④	–	3	2	0	1	0	1	3	0	1	0	0		0	1	0	0
(2)	1	1	0	0	④	2	–	–	2	0	0	1	0	1	5	0	1	0	0	$\leftarrow t_2 \rightarrow$	0	0	0	1
(3)	1	0	1	0	④	3	–	–	3	1	1	0	1	0	1	1	0	1	1		1	0	1	1
(4)	0	1	0	0	④	3	–	–	3	1	0	1	0	0	7	0	1	1	1		0	1	1	1
(5)	0	0	0	0	④	3	–	–	3	2	0	0	0	0	2	0	0	0	1		0	0	0	1
(6)	1	1	0	0	④	2	–	–	2	0	0	0	0	0	8	0	0	0	0	$\leftarrow t_2 \rightarrow$	0	1	0	1
(7)	1	1	1	1	3	3	–	–	3	1	1	1	1	1	4	1	1	⓪	0	$\leftarrow P_{mut} \rightarrow$	1	1	①	0
(8)	1	1	1	0	④	3	–	–	3	1	1	1	1	0	6	1	1	1	0		1	1	1	0
					*	⇑	*	*																

$$m_1 = \begin{cases} \text{Sum} & 29 \\ \text{Average} & 3.625 \\ \text{Max} & 4 \\ \text{Min} & 3 \end{cases} \qquad m_2 = \begin{cases} \text{Sum} & 22 \\ \text{Average} & 2.75 \\ \text{Max} & 3 \\ \text{Min} & 2 \end{cases}$$

minterm test vector for product term m_2. This vector yields a test for a fault in the b_2. The strings 4 and 8 have the same value (1011) in m_4. This test vector is a complete test vector for m_4 since it is assumed that the PLA is irredundant. Hence the implicant is essential and part of the final set. Obviously, one shrinkage test vector is sufficient for testing isolated product term with one element. The product term m_4 is marked with an '*' in Table I to disable it from further consideration. The string 7 (0001) can test for a fault in the complement bit line b_2 of m_3. The fittest members can be selected more than once. Table I shows how many times an individual is reproduced.

Next, the Crossover operator is applied to two individuals chosen from the population using the selection operator. A crossover site along the bit strings is randomly chosen and the values of the two strings are exchanged up to this point. Mating between strings 1 (0101|0) and string 6 (0011|1) with a crossover point 3 (see Table I) yields two new offspring (0101) and (0010), which are placed into the next generation of the population. By recombining portions of good individuals, this process is likely to create even better individuals.

Next, the mutation operators are applied. The mutation with template (t_1 for m_1) is applied randomly with probability of 0.25. This avoids "Hamming cliffs". The regular mutation operator is not applied until generation 1.

Tables II and III represent generations 1 and 2, respectively. The test vectors found after generations 0, and 1, are shown in Tables IV and V, respectively. The final Shrinkage Test Vectors for the PLA under consideration are shown in Table VI.

Table III of *generation 2* shows the test vector generated for m_2 (0111). The test vector {(0011),

TABLE III: GENERATION 2

	X_1	X_2	X_3	X_4	m_1	m_2	m_3	m_4	m_2															
(1)	0	1	0	0	–	3	–	–	3	1	0	1	0	0	8	0	1	1	0		0	1	1	0
(2)	0	0	0	1	–	3	–	–	3	1	0	0	0	1	7	0	0	1	0		0	0	1	0
(3)	1	0	1	1	–	3	–	–	3	1	1	0	1	1	4	1	1	1	1		1	1	1	1
(4)	0	1	1	1	–	④	–	–	4	2	0	1	1	1	3	0	0	1	1		0	0	1	1
(5)	0	0	0	1	–	3	–	–	3	0	0	1	1	1	6	0	1	1	1	$\leftarrow t_2 \rightarrow$	0	0	1	0
(6)	0	1	0	1	–	3	–	–	3	1	0	1	0	1	5	0	1	0	1		0	1	0	1
(7)	1	1	1	0	–	3	–	–	3	1	1	1	1	0	2	1	1	0	1	$\leftarrow t_2 \rightarrow$	1	0	0	0
(8)	1	1	1	0	–	3	–	–	3	1	1	1	1	0	1	1	1	0	0		1	1	0	0
					*	⇑	*	*																

$$m_2 = \begin{cases} \text{Sum} & 25 \\ \text{Average} & 3.125 \\ \text{Max} & 4 \\ \text{Min} & 3 \end{cases}$$

(0111)} yields an incomplete test for m_2 since the minterms (0010) and (0110) are bounded. Therefore, a test for the uncomplemented bit line b_4 is not possible.

V- CONCLUSIONS AND PARALLEL IMPLEMENTATION

This article describes the use of genetic algorithms to generate patterns for testing programmable logic arrays. Existing methods tend to be computationally expensive. Our proposed algorithm overcomes this problem to generate good solutions efficiently. While the preliminary results are encouraging, further testing with larger size PLA is necessary to validate these results.

The algorithm described is well suited for parallel processing. PLA fault simulation in parallel using GA should help to overcome the well-known bottleneck of serial simulation. Genetic algorithms are inherently parallel algorithms. Hence **PLAGA** should be easily scaleable to multiple processor systems with shared memory.

VI- REFERENCES

1- P. Bose, "A Novel Technique for Efficient Implementation of a Classical Logic/Fault Simulation Problem," *IEEE Trans. on Computers*, Vol. 37, pp.1569-1577, December 1984.

2- A. Cruz and R. Reilova, "A Hardware Performance Analysis for a CAD Tool for PLA Testing, " *39th Midwest Symposium on Circuits and Systems*, 1997.

3- M. S. Hsiao, E, M. Rudnik and Janak H. Patel, "Automatic Test Generation Using Genetically Engineering Distinguishing Sequences," *Proceedings of the VLSI Test Symposium*, pp. 216-223, April 1996.

4- E. M. Rudnik, J. G. Holm, D. G. Saab and Janak Patel, "Application of Simple Genetic Algorithms to Sequential Circuit Test Generation," *Design Automation Conference*, pp. 717-721, June 1994.

5- Janak Patel et al., "Parallel Genetic Algorithms for Simulation-based Sequential Circuit Test Generation," *Proceedings of the International Conference on VLSI Design*, pp. 475-481, January 1997.

6- E. M. Rudnik and Janak Patel, "A Genetic Approach to Test Application Time Reduction for Full Scan and Partial Scan Circuits," *Proceedings of the Eight International Conference on VLSI Design*," pp. 288-293, January 1995.

TABLE IV: *TEST VECTORS FOUND AFTER GENERATION 0*

	Product Term	Candidate Shrinkage Tests	Final Shrinkage Test
m_1 =	X X X 0	{ (0100), (1100), (1110) }^{G0}	
m_2 =	0 X 1 X	{ (0011) }^{G0}	
m_3 =	0 X 0 1	{ (0001) }^{G0}	
m_4 =	1 0 1 1	{ (1011) }^{G0}	{ (1011) }

TABLE V: *TEST VECTORS FOUND AFTER GENERATION 1*

	Product Term	Candidate Shrinkage Tests	Final Shrinkage Tests
m_1 =	X X X 0	{ (0000), (0100), (1010), (1100), (1110) }$^{G0 \cup G1}$	{ (0000), (1110) }
m_2 =	0 X 1 X	{ (0011) }^{G0}	
m_3 =	0 X 0 1	{ (0001), (0101) }$^{G0 \cup G1}$	{ (0001), (0101) }
m_4 =	1 0 1 1	{ (1011) }^{G0}	{ (1011) }

TABLE VI: *FINAL SHRINKAGE TEST VECTORS*

	Product Term	Candidate Shrinkage Tests	Final Shrinkage Tests
m_1 =	X X X 0	{ (0000), (0100), (1010), (1100), (1110) }$^{G0 \cup G1}$	{ (0000), (1110) }
m_2 =	0 X 1 X	{ (0011), (0111) }$^{G0 \cup G2}$	{ (0011), (0111) }
m_3 =	0 X 0 1	{ (0001), (0101) }$^{G0 \cup G1}$	{ (0001), (0101) }
m_4 =	1 0 1 1	{ (1011) }^{G0}	{ (1011) }

7- R. S. Wei and Sangiovanni-Vicentelli, "PLAtypus: A PLA Test Generation Tool." *Trans. on Computer Aided Design*, Vol. CAD-5, October 1986.

8- P. Bose, "Generation of Minimal and Near-Minimal Test Set for Programmable Logic Arrays," *International Conference on Computers*, December 1984.

9- M. Robinson and Rajski, "An Algorithm Branch and Bound Method For PLA Test Pattern Generation," *International Test Conference*, pp. 66-74, 1988.

10- J. E. Smith, "Detection of Faults in Programmable Logic Arrays," *IEEE Transactions on Computers*, Vol. C-28, No. 11, November 1979.

11- Hua and et al., "Built-in Tests for VLSI Finite State Machines," *Dig. of Papers 14th Int'l Conf. on Fault Tolerant Computing*, June 1984.

12- M. Abramoci and et al., "Digital Systems Testing and Testable Design," 41 Madison Avenue, New York, NY 10010: *Computer Science Press*, 1990.

13- G. Goldberg, "Genetic Algorithms in Search, Optimization and Machine Learning," *Addison Wesley, Reading, MA*, 1989.

14- R. Tanese, "Parallel Genetic Algorithms for a Hypercube," in *Proc. of the Second Int. Conf. on Genetic Algorithms*, ed. J. J. Grefenstette (Lawrence Erlbaum Associates, 1987), p. 177.

15- B. Manderick and P. Spiessens, "Fine-grained Parallel Genetic Algorithms," in *Proc. of the Third Int. Conf. on Genetic Algorithms*, ed. J. D. Schaffer (Morgan Kauffman, 1989), p. 428.

16- T. Starweather, D. Whitley and K. Mathias, "Optimization Using Distributed Genetic Algorithms," in *Parallel Problem Solving from Nature, Lectures Notes in Computer Science,* Vol. 496, eds. H. P. Schwefel and R. Manner (Spring-Verlag, 1991), p. 476.