A Hybrid Deterministic/Genetic Test Generator to Improve Fault Effectiveness and Reduce CPU Time Run

Alfredo Cruz, PhD
Electrical & Computer Engineering and Computer Science Department
Polytechnic University of Puerto Rico
Hato Rey, Puerto Rico
alcruz@pupr.edu

ABSTRACT

This paper focuses on an evolutionary algorithm (EA) approach in the development of effective test vector generation for single and multiple fault detection in VLSI circuits. The genetic operators (selection, crossover, and mutation) are applied to the CNF-satisfiability problem for the generation of test vectors for growth faults in Programmable Logic Arrays (PLAs). The CNF-constraints satisfaction problem has several advantages over other approaches used for PLA testing. The method proposed eliminates the possibility of intersecting a redundant growth term with a valid candidate test vector. Deterministic procedures are used to allow the identification of untestable faults and to improve the fault coverage. This hybrid deterministic/genetic test generator helps improve fault effectiveness and reduce CPU time run. Experimental results have confirmed that the number of untestable faults identified contributed to test generation effectiveness.

SINOPSIS

Este artículo se enfoca en un método para desarrollar pruebas de fallas efectivas para la detección de errores sencillos y múltiples en circuitos de VLSI utilizando algoritmos genéticos. Los operadores genéticos (selección, "crossover", mutación) son aplicados al problema de satisfiabilidad-CNF para la generación de pruebas de fallas crecientes en Arreglos Lógicos Programables (PLA's). El problema de satisfiabilidad-CNF tiene ventajas sobre otros métodos utilizados para la generación de pruebas a los Arreglos Lógicos Programables (PLA's). El método propuesto elimina la posibilidad de interceptar un término de fallas crecientes redundante con una prueba de falla que es una candidata válida para probar fallas. Procedimientos determinísticos son utilizados para permitir la identificación para permitir la identificación de fallas que no pueden ser detectadas y para mejorar la cubierta total de fallas. Este generador de pruebas híbrido que es determinístico/genético ayuda a mejorar la efectividad de la detección de fallas y reduce el tiempo de uso del CPU en la ejecución del mismo. Resultados experimentales han confirmado que el número de fallas no-detectables identificadas por este método contribuye a la efectividad en la generación de las pruebas.

I- INTRODUCTION

With the increasing use of PLAs in systems design at the Very Large Scale Integration (VLSI) level, PLA testing has become very important. There is a need to keep pace with the increase in PLA size by enhancing the efficiency of PLA test generation. However, the PLA simplified structure does not ease the inherent difficulty for test pattern generation and fault simulation in PLAs. Several algorithms have been proposed for PLA testing using the T operation, but they tend to be computationally expensive (Wei & Sangiovanni-Vicentelli, 1986; Robinson & Rajski, 1988). 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. This can affect test size and test application time. Clearly, present DFT methods have not addressed the problem adequately (Hua, Jou, and Abraham, 1984; Xu & Breuer, 1988; Thompson, 1996; Miranda, 1997; Carbine & Feltham, 1998; Needham, 1999; Kapur, Hay, and Williams, 2000; Rajsuman, 2001).

In this paper, we present our approach to the problem of PLA testing using an evolutionary algorithm based solution aimed to address the shortcoming of existing methods.

II- FAULT MODELS AND DEFINITIONS

The most commonly considered fault model when testing digital circuits is known as the stuck-at fault (i.e., s-a-0 or s-a-1) (Abramovici, Breuer, and Friedman,

1990). As a consequence of the PLAs array structure, the stuck-at fault alone cannot adequately model all physical defects. 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 cross-point fault.

Crosspoint faults can be divided into two classes: missing crosspoint faults and extra crosspoint faults. Different types of faults include growth, shrinkage, appearance, and disappearance faults.

The shrinkage and growth faults derived from the cross-point fault model also cover other faults such as the appearance and disappearance of cross-point in the OR-plane. Fault models also help limit the number of necessary tests, as opposed to testing for all possible fault types.

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 cross-point devices into what is known as the AND plane.

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

$$f_1(x_1, x_2, x_3, x_4) = (x_1 \cap x_2) \cup (x_1 \cap x_3)$$

$$f_2(x_1, x_2, x_3, x_4) = (x_2 \cap x_3) \cup (\neg x_1 \cap x_2) \cup (x_1 \cap \neg x_2 \cap \neg x_3)$$

This PLA has four inputs (x_1, x_2, x_3, x_4), four product terms (m_1, m_2, m_3, m_4), and two output functions (f_1, f_2).

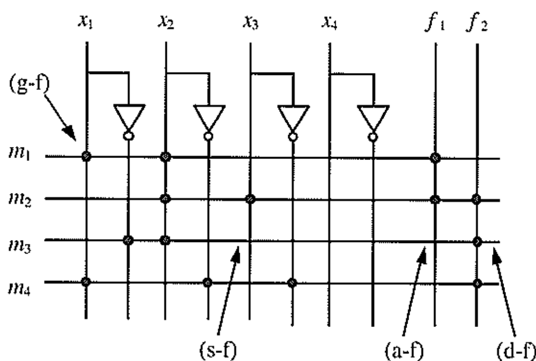

Figure 1: *A PLA Schematic test generation procedures*

In this section we shall discuss procedures and present specific algorithms to derive minimal test vectors for missing crosspoint faults.

Growth faults occur due to the absence of crosspoint devices in the AND plane. These faults correspond to the removal of a literal from a product term (implicant) of the function in the AND array. This causes the product term (implicant) to grow because it becomes independent of an additional input variable and hence, the product term covers more minterms.

The two important requirements for fault detection are fault sensitization and fault propagation.

A missing device fault in the AND plane can be sensitized only if the implicant under testing carries a 0 when fault-free and a 1 in the presence of a fault. A propagation path must be established once the fault has been sensitized, otherwise the fault is masked. Propagation is done by deselecting all other product lines connected to the output except the product term under testing.

The procedure for deriving the growth test vectors is explained with the aid of Figure 2. The product term is represented by an AND gate of 4 inputs. A dash '-' in the input lines indicates the absence of a device, whereas a circle '•' in the input lines indicates the presence of a device.

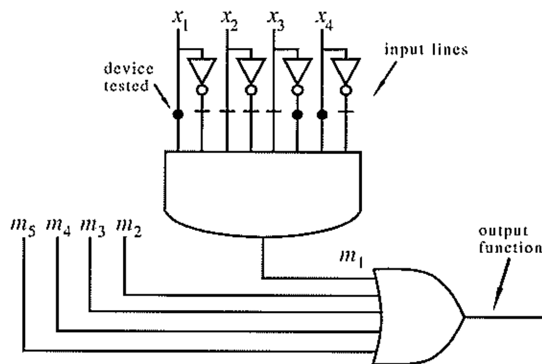

Figure 2: *Product Term Under Test*

For example, to detect a missing device at x1 of the implicant under consideration [1X01], logic 0 must be applied to the input x1, while the care values (at input x3 x4) re-main unchanged. Since the value of the literal was changed from 1 to 0, a term from this set could detect a fault in the uncomplemented bit-line. To detect a fault in the complemented bit-line the literal must be changed from 0 to 1.

We should be able to sensitize the fault (if it exists) at the output of the AND gate. The implicant being tested carries a 0 in the absence of a fault at the output

of the AND gate, while a value of 1 denotes a fault. For the sensitization of growth faults, a 0 must be generated on the product line. This is done by toggling the input value connected to the target growth fault bit-line.

B- Growth Test

A growth term stands for the set of extra minterms formed by a missing fault. Growth terms may have terms in the ON-set function (i. e., minterms) and in the OFF-set function (i.e., maxterms). The terms in the ON-set function fail to uniquely select the product line on which the target is located. Since it will also select the product terms that cover them, the fault cannot be propagated. This procedure can be carried out by computing the intersection (denoted by ($\cap$)) between the growth terms generated for each product with the complement function (OFF-set) (Wei & Sangiovanni-Vicentelli, 1986; Robinson & Rajski, 1988). One of dis-advantages in this approach is backtracking. This could occur when the test is chosen and fails to propagate.

PLAtestGA uses the conjunctive normal form (CNF) logical expression (equivalent to the complement's function) to derive the test set for growth faults. An expression of proposition is in conjunctive normal form (CNF) when it is a sequence of clauses joined by an AND relation. Each of these clauses is in the form of disjunction, the OR of literals. Use of the CNF is supported by the De Morgan's theorem (Mano & Kime, 2000).

The genetic algorithm for the CNF-satisfiability problem is applied for the generation of test vectors for growth faults.

Fitness Functions

An evaluation or fitness function is used to determine the fitness of each candidate solution. A fitness value is given to individuals in the current population during each iteration step, called a generation. The evaluation function is the link between the problem to be solved and the GA.

The problem is to determine whether there exists a truth assignment for the variables in the expression, so that the CNF expression evaluates them to TRUE. For example, the following CNF logical expression

$$(\neg x_1 \cup x_3 \cup \neg x_4) \cap (x_1 \cup \neg x_3 \cup x_4) \cap$$
$$(x_1 \cup \neg x_2 \cup \neg x_4) \cap (x_1 \cup x_2 \cup x_3) \cap$$
$$(x_1 \cup x_3 \cup \neg x_4)$$

has several truth assignments (a valid candidate test vector) for which the whole expression evaluates to TRUE, e. g., any assignment with x_3 = TRUE and x_4 = TRUE. The CNF expression of the example PLA is made up of five clauses that allow us to rank potential bit pattern solutions in the range of 0 to 5 (depending on the number of clauses that pattern satisfies). When a pattern has a fitness of 5, a maxterm of the function is evaluated. Thus, test pattern 0101 has a fitness of 1, 0110 has a fitness of 2, and 0111 has a fitness of 5 and is a solution.

Population

All genetic algorithms work on a constant population size that consists of several alter-native solutions to the given problem. Each individual in the population is called a string or chromosome, in analogy to chromosomes in natural systems, and is generated at random. The strings can be composed of characters or symbols analogically referred to as genes. The population size determines the amount of information stored by the GA. The GA population evolves over a number of generations.

C- Genetic Operators

The three basic steps performed during the generation of the test vectors are selection, crossover, and mutation.

Selection

Various selection schemes have been used in the past. We focused on the generational roulette wheel and steady state selections that are perhaps the best representatives of non-overlapping and overlapping populations, respectively. The generational reproduction or non-overlapping population such as the roulette wheel replaces the entire population at once, while overlapping populations such as the steady state replaces only a few members at a time.

Steady state selection or overlapping population is a technique used to reduce execution time by reducing fitness computation in the population (Rudnick, et. al, 1994).

Crossover

One-point crossover (1 CX), two-point crossover (2 CX), and the uniform crossover operators were used in this study for efficiency comparisons. Crossover rates of 0.75 and 1.0 were used, as well as mutation rates of 0.01 and 0.001. Experiments for the PLA ATPG

confirm that low mutations approximately equal to 0.001 reduce the CPU time. The mutation rate had a much greater effect on the results of the three PLA datasets than the crossover rates used.

In the simplest crossover operator, called one-point crossover, the parents swap bits from the crossover point. The first set of bits is taken randomly from one parent and the remaining bits (starting from the crossover point) are taken from the other parent. Figure 3 illustrates one-point crossover with parents '10101011' and '11111001,' and shows it after the third bit cut point.

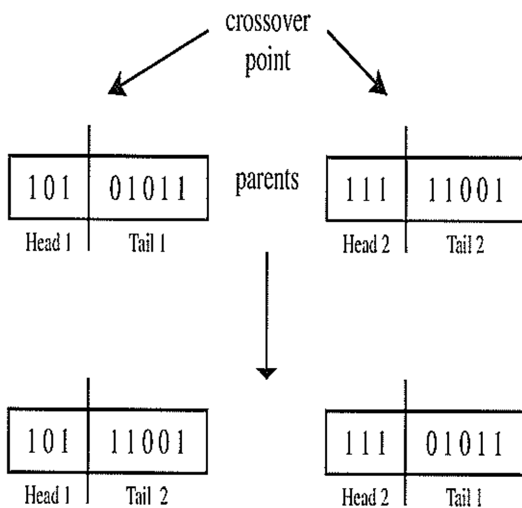

Figure 3: The Effect of the One-Point Crossover Operation

Like one point crossover, the second crossover operator is distinguished by two cut points rather than one, and is selected at random with chromosome material swapped between the two cut points. For example, in two-point crossover operators there are two cut points (marked with the vertical bars) and the substrings are swapped between the two points (see sample below).

Parent 1	1 0 1 1 | 0 0 1 | 0 0 1 1
Parent 2	0 1 0 1 | 1 0 1 | 1 1 0 1
Child 1	1 0 1 1 | 1 0 1 | 0 0 1 1
Child 2	0 1 0 1 | 0 0 1 | 1 1 0 1

Syswerda (1989, 1991) described the third crossover operator that was used in this study. He called this operator uniform crossover. According to Syswerda, two parents are selected which result in two children being produced. For each bit position on the two children, we choose randomly (with probability p) which parent contributes its bits value to each child using the template. For example, for p = 0.5 (0.5-uniform crossover), the strings

Parent 1	1 0 1 0 1 1 0 0
Parent 2	0 1 0 0 1 0 1 0
Template	2 2 1 1 2 1 1 2

may produce the following offspring:

Child 1	0 1 1 0 1 1 0 0
Child 2	1 0 0 0 1 0 1 0

The template contains a series of random numbers of 1s and 2s. The value of 1 in the template leaves the bit positions of the children the same as their parents. The value of 2 assigns to Child 1 the corresponding bit from Parent 2, while Child 2 gets the corresponding bit from Parent 1.

Mutation

Mutation is a unary operation involving the probabilistic alteration of components in a chromosome. In the circumstances where bit strings are used to represent chromosomes, mutation engages in a simple operation of flipping a bit from one (1) to zero (0) or vice versa. Normally, this is a process necessary for avoiding stagnation.

Mutation frequency has to be low in order to avoid the search from generating a random walk.

The routine mutation in C++ is shown below:

```
void computeMutation(struct sibling siblingArr[])
 {
 int mutantIndx, geneIndx;

 mutantIndx = rand()%MAX_POP;
 geneIndx = rand()%MAX_GEN;
 (siblingArr[mutantIndx].genes[geneIndx]
   ? siblingArr[mutantIndx].genes[geneIndx] = 0
   : siblingArr[mutantIndx].genes[geneIndx] = 1);
 return;
 } // end computeMutation
```

III- RESULTS

Tests were generated for three different PLA sizes (small, medium, and large) using the GA test generator. The PLA sizes range from 240 to 1920 crosspoints as follows: the small PLA size is of (12×20) 240 crosspoints, the medium PLA size is of (14×60) 840 crosspoints, the large PLA size is of (16×120) 1920 crosspoints.

Several simulations are necessary to average the

results, because GA-based ATPG is inherently a random process. This multiplies the number of experiments. In our work, results are based on averaging five runs each with different random seed for each set of independent variables.

- The headings of the Tables I and II have the following meaning:
- The population size is shown in the column under heading "Pop. Size".
- The crossover rate is shown in the column under heading "CX rate".
- The mutation rate is shown in the column under heading "Mut. rate".
- The number of detectable faults calculated is shown in the column under heading "DF".
- The number of undetectable faults identified by the deterministic procedure is shown in the column under heading "UF".
- The number of recovered irredundant faults is shown in the column under heading "RIF".
- The total number of possible faults including the redundant faults (same as the growth term list given for each PLA) is shown in the column under heading "TNPF".
- The number of test vectors is shown in the column under heading "NTV".
- All execution time is given in milliseconds (ms). It includes fault simulation time and it is shown in the column under heading "CPU time".
- The percentage of fault coverage of the pure crosspoint faults when not counting redundant faults is shown in the column under heading "FC".
- The percentage of fault effectiveness of the real crosspoint fault when counting redundant faults is shown in the column under heading "FE".
- The number of generations is shown in the column under heading "NG".
- The detectable faults (DF), undetectable faults (UF), recovered irredundant faults (RIF), CPU run time, fault coverage (FC), fault effectiveness (FE), and the number of generations (NG) are the dependent variables observed from the ATPG result process.

The independent variables are composed of circuit specific (inputs, product terms) and GA parameters. The GA parameters are the followings: population size, crossover and mutation rate.

The parameters or independent variables such as population size, crossover and mutation rate used to control GA-based ATPG can greatly affect test size, fault coverage, and CPU execution time. Knowing a priori what the optimal settings of the exogenous parameters or independent variables (mainly the mutation rate, the crossover probability, and the population size) are allows efficient testing procedures. When comparing different algorithms one must look at the dependent variables such as total execution time, the overall test size in terms of the number of test vectors, the fault coverage, and the fault efficiency.

A deterministic fault oriented test is still needed to identify the redundant faults in the list of undetectable faults (UF) that remain. This is due to a limitation of any simulation-based ATPG that makes it unable to generate tests for targeted faults, and to identify undetectable faults (UF). Undetectable faults were not pre-filtered in the PLA datasets used in our experiments. Good fault coverage (FC) may be achieved using only a simulation-based approach, but the fault efficiency (FE) of the test generator will not be better than the fault coverage (FC). The reason that a FC of 100% cannot be achieved is due to redundant faults. PLAtestGA is a hybrid deterministic/genetic test generator that helps improve fault effectiveness and reduce CPU run time. First, the GA is run until no more improvement is obtained; then the deterministic approach is used to target remaining undetectable faults. The integrated deterministic algorithm with the simulation-based algorithm can always achieve 100% efficiency as shown in Tables I and II.

A- Analysis of the Dependent and Independent Variables

Previous studies have shown that the steady state selection is susceptible to stagnation (Rudnick, et. al, 1994; Syswerda, 1991), when the population size is small. In this study, the results confirm that small populations with low mutation frequency make the steady state selection converge rapidly leaving some faults untested. Therefore, the fault cover-age (FC) can be unacceptable in most small population cases. An acceptable FC should be higher than 80%. Here is where the deterministic procedures can complement the simulation-based algorithm. The deterministic procedures contribute to test effectiveness by finding all the faults left untested by the simulation-based algorithm (RIF values) and identifying all the undetectable faults (UF values).

In this study, we present the results between these two selections using the one-point crossover operation (1 CX) as a point of reference for the Large PLA. However, similar results are found for the two-point operation (2 CX) and the uniform crossover operation. For brevity and due to the lack of space, we restrict our discussion only to the large PLA. Comparison of the results in Tables I and II demonstrates the advantages of the steady state selection over the roulette wheel in terms of CPU time and the number of generations (NG) to achieve the optimal test vectors. In general, the overhead associated with the non-overlapping population, as the with roulette wheel, is the time used for computing fitness of each new population. As far as speedup is concerned, the results for (1 CX) when comparing these two selections is the best for population size 16, as shown in Figure 4.

The speedup is calculated as follows:

$$Speedup = \frac{CPU\ time\ of\ Roulette\ Wheel}{CPU\ time\ of\ Steady\ State}$$

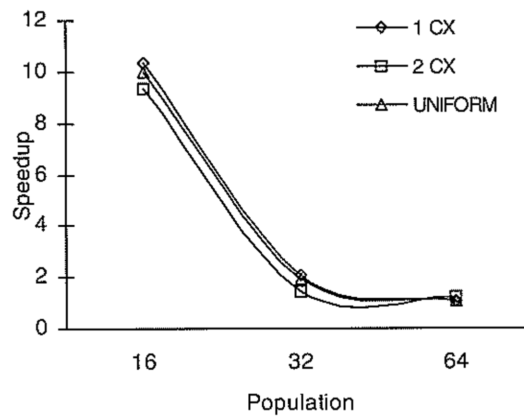

Figure 4: *Speedup Between the Overlapping vs. Non-overlapping Population with Mutation Rate 0.001, Crossover Rate 1.0 Using 1 CX, 2 CX, and the Uniform Operator for the Large PLA*

From Table I of the roulette wheel of the Large PLA we can observe that the column with heading DF is kept close to 1480. The column with heading UF shows that there are 16 undetectable faults. This

Table I. *Large PLA (16x120) with Roulette Wheel Selection and 1 CX*

Pop. Size	CX rate	Mut. rate	DF	UF	RIF	TNPF	NTV	CPU Time (ms)	FC (%)	FE (%)	NG
16	1.00	1/100	1480.0	16	0.0	1496	347.2	172055.4	98.93	100	22969.4
		1/1000	1479.2	16	0.8	1496	348.2	171921.2	98.93	100	24103.2
	0.75	1/100	1479.2	16	0.8	1496	348.2	184397.2	98.93	100	25079.0
		1/1000	1480.0	16	0.0	1496	347.6	187099.2	98.93	100	26958.0
32	1.00	1/100	1480.0	16	0.0	1496	347.6	191154.8	98.93	100	15376.8
		1/1000	1480.0	16	0.0	1496	347.2	185432.4	98.93	100	15675.6
	0.75	1/100	1480.0	16	0.0	1496	348	210068.0	98.93	100	16748.4
		1/1000	1480.0	16	0.0	1496	347.2	203915.2	98.93	100	17645.6
64	1.00	1/100	1480.0	16	0.0	1496	347.2	199963.6	98.93	100	9319.4
		1/1000	1480.0	16	0.0	1496	347	199527.0	98.93	100	9287.0
	0.75	1/100	1480.0	16	0.0	1496	347	226960.4	98.93	100	10790.2
		1/1000	1480.0	16	0.0	1496	347	226754.2	98.93	100	10808.2

demonstrates, once again, that the deterministic procedure always identifies all the undetectable faults. A FC of 98.93% is achieved. FE is 100% since the identification of the undetectable faults is feasible. The TNPF is equal to 1496. The result is very consistent with the sum of columns DF and UF; in this case, the sum of 1480 and 16 is equal to 1496. The results of the number of generations (NG) are also consistent, the higher the population sizes the lower the NG. For example, the number of generations falls below 10808.2 for population size 64, while for population size 32 the NG can be as high as 17645.6. For population 16, the lowest number of generations (NG) is 21327.8.

Table II shows that the NTV is high when population size 16 is used with mutation rate 0.001. In addition, notice that the RIF values is high (over 1262) when population size 16 and mutation rate 0.001 is used. The RIF column indicates that over 1262 test vectors were found using the deterministic procedure. Notice that the NTV generated by the GA averages 348 for the Large PLA. It can be observed in Table II when the RFI values are equal to zero. Since the column of the DF is equal to 1480, the density of the test vectors is 1486/348 = 4.25. This means that a test vector generated by the GA can detect an average of over 4 faults. However, when the deterministic procedures have an active participation the density of the test vectors are degraded.

Figure 5 shows the results of the comparison among the one-point crossover operation (1 CX), two-point crossover operation (2 CX), and uniform cross-over for the steady selection for a Large PLA. These results are compared using both crossover rates of 1.0 and 0.75 with a fixed mutation rate of 0.001.

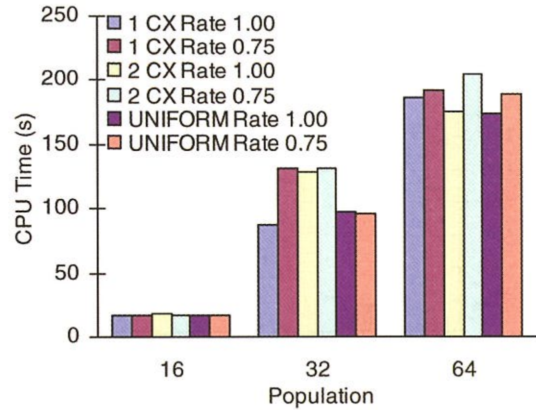

Figure 5: *Comparison Among the 1 CX, 2 CX, and Uniform Crossover Using a Mutation Rate of 1/1000 and the Steady State Selection for the Large PLA*

Table II: *Large PLA (16x120) with Steady State Selection and 1 CX*

Pop. Size	*CX rate*	*Mut. rate*	*DF*	*UF*	*RIF*	*TNPF*	*NTV*	*CPU Time (ms)*	*FC (%)*	*FE (%)*	*NG*
16	1.00	1/100	1441.6	16	38.4	1496	380.4	126317.8	98.93	100	22565.8
		1/1000	140.8	16	1339.2	1496	1372.2	16668.0	98.93	100	596.4
	0.75	1/100	1472.0	16	8.0	1496	360	148237.2	98.93	100	27559.6
		1/1000	120.0	16	1360.0	1496	1387.6	17547.4	98.93	100	907.4
32	1.00	1/100	1476.8	16	3.2	1496	352	149108.6	98.93	100	17400.8
		1/1000	1254.4	16	225.6	1496	527.6	87860.4	98.93	100	10822.2
	0.75	1/100	1480.0	16	0.0	1496	348.6	170255.0	98.93	100	20462.8
		1/1000	1376.8	16	103.2	1496	431.8	130888.2	98.93	100	16922.4
64	1.00	1/100	1479.2	16	0.8	1496	348.2	167757.2	98.93	100	11517.6
		1/1000	1480.0	16	0.0	1496	347.2	186820.8	98.93	100	13764.6
	0.75	1/100	1480.0	16	0.0	1496	347.6	204127.6	98.93	100	14589.4
		1/1000	1480.0	16	0.0	1496	347.2	192038.2	98.93	100	14246.6

Figure 6 shows the CPU time for the Large PLA at population sizes 16, 32, and 64. The crossover rate 0.01 for the one-point crossover operation (1 CX) is used to compare the effects of using mutation rate 0.01 and 0.001. Good results can also be obtained with different crossover rates for the (2 CX) and uniform crossover operations using same parameters.

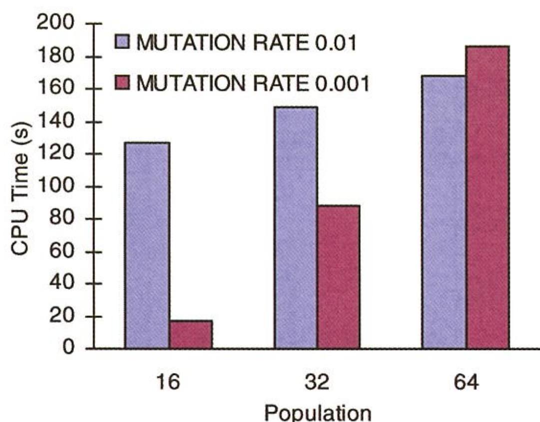

Figure 6: *Comparison Between Mutation Rates of 0.01 and 0.001 Using Steady State Selection, Crossover Rate 1.0 and 1 CX for the Large PLA*

IV- CONCLUSIONS

Fault coverage tends to improve with increasing population size. In this study, there is no need to increase the population size (which is computationally expensive) to improve fault coverage. The deterministic procedure finds all remaining undetected faults, and identifies all undetectable (redundant) faults, regardless of the population size. This approach clearly has advantages that help reduce CPU time, since fault coverage is independent of the population size.

Deterministic procedures assure a maximum achievement of FC, and an FE of 100% is guaranteed. This is an improvement over previous genetic algorithms in VLSI testing. The use of the GA followed by the deterministic procedures can achieve the best balance between run time and fault coverage. Realizing which algorithms are best suited for solving specific problems is an important issue to consider.

REFERENCES

[1] Abramovici, M., Breuer, M. A., and Friedman, A. D. (1990). Digital Systems Testing and Testable Design. Computer Science Press.

[2] Carbine, A., and Feltham, D. (1998, September). Pentium Pro Processor Design for Test and Debug. IEEE Design & Test of Computers, 15(3), 77-82.

[3] Hua, K. E., Jou, JingYang, Abraham, J. A. (1984, June). Built-In Tests for VLSI Finite State Machines. Dig. of Papers 14th International Conference on Fault Tolerant Computing, 292-297.

[4] Kapur, R., Hay, and Williams, T. (2000, September). The Mutation Metric for Benchmarking. IEEE Design & Test for Computer, 18-21.

[5] Miranda, J. M. (1997, September). A BIST and Boundary-Scan Economics Framework. IEEE Design & Test of Computers, 14(3), 17-23.

[6] Needham, W. (1999, November). Nanometer Technology Challenges for Test and Test Equipment. IEEE computer, 32 (11), 52-57.

[7] Rajsuman, R. (2001, June). Design and Test for Large Memories: An Overview. IEEE Design and Test of Computers, 18(3), 16-27.

[8] Robinson, M. and Rajski. (1988). An Algorithm Branch and Bound Method For PLATest Pattern Generation. International Test Conference, 66-74.

[9] Rudnick, E. M., Patel, J. H., Greenstein, G. S., and Nierman, T. M. (1994, June). Sequential Circuit Test Generation in a Genetic Algorithm Framework. Proc Design Automation Conf., 698-704.

[10] Syswerda, G. (1989). Uniform Crossover in Genetic Algorithms. Third International Conference on Genetic Algorithms, ed. By Schaffer, Morgan Kaufman Publishers, San Mateo, CA, 2-9.

[11] Syswerda, G. (1991). A Study of Reproduction in Generational and Steady State Genetic Algorithms. In Foundations of Genetic Algorithms. ed. G. Rawling, San Francisco: Morgan Kauffman, 94-101.

[12] Thompson, K. (1996). Intel and the Myths of Test. IEEE Design & Test of Computers, 79-81.

[13] Wei, R. S., and Sangiovanni-Vicentelli. (1986, October). PLAtypus: A PLA Test Generation Tool. Trans. on Computer Aided Design, 5.

[14] Xu, X., and Breuer, M. (1988, August). Analysis of Testable PLA Designs. IEEE Design & Test of Computers, 14-28.