World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:9, No:8, 2015

An Analysis of Genetic Algorithm Based Test Data Compression Using Modified PRL Coding

K. S. Neelukumari, K. B. Jayanthi

***Abstract*—**In this paper genetic based test data compression is targeted for improving the compression ratio and for reducing the computation time. The genetic algorithm is based on extended pattern run-length coding. The test set contains a large number of X value that can be effectively exploited to improve the test data compression. In this coding method, a reference pattern is set and its compatibility is checked. For this process, a genetic algorithm is proposed to reduce the computation time of encoding algorithm. This coding technique encodes the 2n compatible pattern or the inversely compatible pattern into a single test data segment or multiple test data segment. The experimental result shows that the compression ratio and computation time is reduced.

***Keywords*—**Backtracking, test data compression (TDC), x-filling, x-propagating and genetic algorithm.

I. Introduction

TODAY'S system-on-chip (SoC) test data volume has become huge due to the factors of design complexity and the integration of embedded cores which in turn has resulted in longer testing time and tester memory requirement. The percentage of x-bit in SoCs test set is quite high and filling these x-bit in test data will improve the compression ratio. Therefore, the number of specified bit in the test data will influence the efficiency of encoding bit as referred to (1). So the x-values in the pattern should be replaced with 1's and 0's.

$$E = \frac{number\ of\ specified\ bits}{number\ of\ bits\ fed\ into\ decompressor} \quad (1)$$

With the advancement in semiconductor manufacturing technology, a Very-Large-Scale-Integration (VLSI) device can contain tens to hundreds of millions of transistors which has led to many challenges in the manufacturing tests. In the digital VLSI circuit design, power dissipation and testing speed has become critical design concerns in the recent years which is driven by the emergence of portable devices in mobile applications. This is applicable not only to design power but also for testing power.

In the testing process, if test vector sets are not optimized for power, even the low power circuits dissipate more than twice the power under test, than at normal operating condition. This is because the large and complex chips require a huge amount of test data and dissipate a substantial amount of power during the test. The reason is that the consecutive input test vectors are statistically independent which results in increased switching activity in the circuit during testing. There are many test parameters that should be improved in order to reduce the test cost. These parameters include the test power, test length (test application time), test fault coverage, and test hardware area overhead.

Minimization of test power, test length (test application time), test fault coverage, and test hardware area overhead in testing of VLSI circuits is a challenging problem for the researchers. Thus, the above factors motivated the researcher to do research in this area. Most of these techniques are used to enhance the design of the conventional LFSR method (or other forms of TPGs such as cellular automata) to reduce the transitions in the primary inputs of the Circuit Under Test (CUT) for test-per-clock BIST (Built in Self Test) or inside the scan-chain for scan-based BIST.

II. Related Work

Various compression algorithms are employed to minimize the area of the test pattern and also to improve efficiency which is discussed. The efficiency equation is given by (1). In [1], compression is achieved by encoding the most frequently occurring blocks with short code words and the less frequently occurring ones with long code words. Therefore, the efficiency depends on the number of distinct blocks that are encoded as well as on the block size. In [2], selective encoding method is used to reduce test data volume and time for scan testing for IP cores. This method encodes the slice of test data that are fed at every clock to the scan chain. The proposed coding scheme in [3] is based on Huffman coding, a statistical data-coding method which reduces the code length for a set of pattern. In this pattern, variable length is used as input to the Huffman algorithm which allows an efficient manipulation of test sets. In paper [4] PRL coding is stored in automatic test equipment (ATE) which transfers it to each core in SOC. The x's value present in data set after decompression reduces the efficiency. The paper [5] uses exactly nine code words which are based on fixed length blocks of input test data and variable length blocks. The algorithm proposes to minimize the volume of data set. In [6], the test data contains do not care bits which make the test data compression more complex. The entropy of unspecified test data information and greedy algorithm of fixed length symbol is used to fill the x- values. Reference [7] proposes a new pattern run – length compression method the encoding of which is $2^{|n|}$compatible and inversely compatible. The compression ratio is also improved and it is given by CR% =$[(|TD| - |TE|)/|TD|]X\ 100$, where |TD| is the size of a test set and |TE| is the size of the compressed test set.

K.S. Neelukumari is with the P. A. college of Engineering & Technology, Pollachi, Tamilnadu, India (e-mail: neeluvijay@gmail.com).

K.B.Jayanthi is with the K. S. Rangasamy College of Technology, Tiruchengode, Tamilnadu, India.(e-mail: jayanthikb@gmail.com).

Reference [8] extends the pattern run – length coding which presents a novel strategy that propagates the x's reference pattern. This technique improves the compression effectively. These papers discuss the encoding algorithm which consumes larger energy because it takes all the test pattern from the pattern generator and then applies encoding algorithm. A genetic algorithm is developed to reduce the time consumption. The fitness function is calculated based on the reference pattern which is used in the encoding algorithm. Reference [9] proposes a genetic algorithm based on which the test pattern is selected for FPGA based on fault coverage. This maintains dynamic global record table (DGRT) containing a list of test patterns with respective to the set of detected faults. This algorithm reduces the simulation time. This paper provides an idea on genetic algorithm that is used in the application-dependent testing of SEUs in SRAM-FPGAs, that takes into account SEUs in configuration bits of the FPGA.

Reference [10] proposes a new genetic algorithm by a multi population pattern searching algorithm (MuPPetS), which uses some of the messy GA ideas like coding and operators. References [11] and [12] propose different methods to reduce the time consumption for testing the SOC cores. In [13] a genetic algorithm for test pattern generator design is presented which composes flip – flops. The optimization includes the search for the optimal combination of register cells type; the presence of inverters at inputs and outputs, the test patterns order in the generated test sequence and the bit order of test patterns. References [14]–[17] discuss the different encoding schemes that are used in test data compression. These papers provide an idea on how genetic algorithm can be employed for various testing domains.

III. Proposed Methodology

A. PRL Coding Technique

The original test set TD is partitioned into a set of patterns of length k, and the last pattern is appended by x's if its length is less than k. The compression starts by simply retaining the first pattern as the common pattern and using "10" or "11" to code the next pattern that is equal to (E-case) or a complement of (C-case), the common pattern. A"0" is used to terminate a round when it encounters a neither equal nor complementary pattern (N-case), which is then directly used as the next common pattern and runs in the same way as described before. When neither an equal nor a complementary pattern appears in the test sequence, the process terminates and new cycle starts.

In the proposed PRL approach, the original test set TD is partitioned in the same way as used in the PRL coding. Similarly, the first pattern is used as the reference pattern. In the E-case and the C-case, "0" and "10" are respectively used to encode the next pattern. An N-case is indicated by "11". In both E-case and C-case, the reference pattern is updated, and then, a back-tracing should probably be done to fill some x's evenly in all the previously coded patterns. Then, the reference pattern:1) does not change in the E-case; 2) is inverted in the C-case; and 3) is XOR-ed with the currently coded pattern in the N-case to attain a new reference pattern, which is used to repeat the aforementioned procedure, until all the patterns in TD are encoded.

Consider the following test set TD of length 20. Let's consider the length of k as 4.

$$TD = x1x10xx1x0x01x01x111.$$

For the k value as 4, the given bits can be divided as:

$$\begin{array}{lccccc} TD = & x1x1 & 0xx1 & x0x0 & 1x01 & x111 \\ & P1 & P2 & P3 & P4 & P5 \end{array}$$

By default, the first four bits (p1) specify the reference pattern x1x1 which is compared with the second pattern, 0xx1. They are equal if the leftmost x of the upper row is replaced with "0"and the leftmost x of the lower row with "1", so that the second pattern is coded as "0" (E-case). Then, some x's in the reference pattern is filled and the updated reference pattern 01x1 is obtained, which is compared with the third pattern x0x0.

01x1 and x0x0 are compared and they are complementary if the leftmost x of the lower row is assigned to "1", and the third pattern is coded as "10"(C-case). Because the upper row does not change when it is updated, the back-tracing skips. Next, the reference pattern is inverted and the resulting pattern is compared with the fourth pattern 1x01.

10x0 and 1x01 are compared and they are neither equal nor complementary no matter how the x's in the two patterns are assigned, so that the fourth pattern is coded as "11"followed by p4 (N-case). Then, the reference pattern is XOR-ed with the fourth pattern to generate a new reference pattern. Here the number of x's in the resulting pattern is increased by 1. The XOR operation brings all the x's in the two operands to the new pattern because any bit that is XOR-ed with x will result in x bit, which is referred to as "x-propagation". Now the new pattern is compared with the last pattern 0111. Hence, 0xx1 and x111 are equal if some x's of the two rows are assigned properly, so that the last pattern is coded as "0" (E-case). The upper pattern is then updated to 0111, which results in a back-tracing x-filling that updates the fourth pattern to 1101, and the previous reference pattern to 1010, followed by updating other previously encoded patterns until the first pattern is updated to 0101. In this manner, the test data is compressed and then the compressed bits are stored in the test core. Later the encoded data is given as the input to the decoding unit which is the reverse process of the encoding technique.

B. Proposed Genetic Algorithm in 2n-PRL Technique

Instead of using normal ATPG, the genetic algorithm is used to overcome the drawbacks in normal ATPG. In the genetic algorithm, a better solution is evolved by using a population of candidate solutions (called individuals, creatures, or phenotypes). Each candidate solution has a set of properties which can be mutated and altered; traditionally, solutions are represented in binary as strings of 0s and 1s, but other encodings are also possible. The evolution usually starts from a population of randomly generated individuals, and is

an iterative process, where the population in each iteration is called as generation.

In each generation, the fitness of every individual in the population is evaluated; the fitness is usually the value of the objective function in the optimization problem being solved. The more fit individuals are stochastically selected from the current population, and each individual's gene is modified to form a new generation. The new generation of candidate solutions is then used in the next iteration of the algorithm. Commonly, the algorithm terminates when either a maximum number of generations has been produced, or a satisfactory fitness level has been reached for the population.

A typical genetic algorithm requires:

- A genetic representation of the solution domain,
- A fitness function to evaluate the solution domain.

A standard representation of each candidate solution is as an array of bits. Arrays of other types and structures can be used in essentially the same way. The main property that makes these genetic representations convenient is that their parts are easily aligned due to their fixed size, which facilitates simple crossover operations.

1. Initialization of Genetic Algorithm

Initially many individual solutions are (usually) randomly generated to form an initial population. The population size depends on the nature of the problem, but typically contains several hundreds or thousands of possible solutions. Traditionally, the population is generated randomly, allowing the entire range of possible solutions. In this case, all the test pattern is considered for the population.

2.Selection

During each successive generation, a proportion of the existing population is selected to breed a new generation. Individual solutions are selected through a fitness-based process, where fitter solutions are typically more likely to be selected. Certain selection methods rate the fitness of each solution and preferentially select the best solutions. Other methods rate only a random sample of the population, as the former process may be very time-consuming. The fitness function is defined over the genetic representation and it measures the quality of the represented solution. The fitness function is always problem dependent. Here the fitness function is considered as the pattern of the generated code words. The code word which will match the given pattern is considered as having higher fitness value.

3. The Genetic Operators and Parameters

Crossover is the main genetic operator. It consists in splitting two chromosomes in two or more sub-sequences and obtaining two new chromosomes by exchanging gene sub-sequences between the two original chromosomes as shown in Fig. 1. The place where a sub-sequence starts is called a cut-point. More specifically, a single-point crossover is adopted by choosing a non-uniform cut-point for each parent and generating the descendants by swapping the segments containing the ending clock cycles as shown in Tables I and II.

TABLE I
SINGLE POINT CROSSOVER EXAMPLE

Parent	Gene sequence
Parent 1	1 0 0 1 0 0 1 0 1 0
Parent 2	0 0 1 0 1 1 0 1 1 1
Child 1	1 0 0 0 1 1 0 1 1 1
Child 2	0 0 1 1 0 0 1 0 1 0

TABLE II
DOUBLE POINT CROSSOVER EXAMPLE

Parent	Gene sequence
Parent 1	1 1 0 1 0 0 1 0 0 1 0 1 1
Parent 2	0 1 0 1 1 0 0 0 1 0 1 0 1
Child 1	1 1 0 1 0 0 0 0 1 0 0 1 1
Child 2	0 1 0 1 1 0 1 0 0 1 1 0 1

The rationale for this choice is summarized in the following considerations. With sequential logic, the output of a circuit depends on both the current input values and the previous inputs, starting from the initial state. Therefore, in order to take advantage of the added benefit of a gene sequence, in terms of number of recognized faults, the state of the circuit, should be taken into account which is a result of all previous inputs, i.e., the previous gene sequence. Hence, it is generally more efficient to have a new generation chromosome which can retain a large fraction of the previous sequence. In order to achieve this behaviour, the following criterion is added in the crossover operation: Random cut-points are generated via the probability density function of an exponential distribution, i.e $f(x;\lambda)=\lambda e^{-\lambda x}$, where x is the distance of the cut point from the end of the sequence. This distribution implies that a large initial segment is kept unchanged from parent to child. Consequently, the end segments that are swapped are relatively short. The level of exploitation of the previous gene sequences can be adjusted via parameter k.

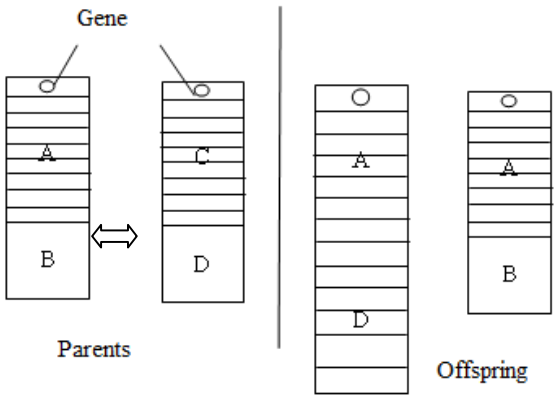

Fig. 1 Crossover Process

The fraction of chromosomes to undergo the crossover operation is the crossover rate (pc). The crossover operator is applied with a probability pc on the selected pair of individuals. When the operator is not applied, the offspring is a pair of identical copies, or clones, of the parents. A higher crossover rate allows a better exploration of the space of solutions which are shown in Fig. 1. However, too high a

crossover rate causes unpromising regions of the search space to be explored.

Mutation is an operator that produces a random alteration in a single bit of a gene. Mutation is randomly applied. The mutation rate, pl, is defined as the probability that an arbitrary bit of an arbitrary gene is complemented. If it is too low, many genes that would have been useful are never discovered, but if it is too high, there will be much random perturbation, the off-springs lose their resemblance to the parents, and the GA loses the efficiency in learning from the search history.

Child 1	1 1 0 1 0 0 0 0 1 0 0 1 1
After mutation	1 1 0 1 1 0 0 0 1 0 0 1 1

Typical values of pl are in the order of 10^{-2}. The mutation operator is controlled by a dynamic pl which decreases linearly between an initial value pl, and a value pl at the final generation. With a linearly decreasing pl, the early generations have a high probability of mutation and solutions are spread all over the solutions space, so that most of them have a chance to be tried. Later generations have a lower mutation probability, so that the search is focused on the regions of the solution space where fitter individuals are found.

4.Selection Process

A selection operator chooses a subset of chromosomes from the current population. Various stochastic selection techniques are available. With this method, an individual is selected with a probability that is directly proportional to its fitness.

5. The Fitness Function

The fitness function measures the quality of the solution, and is always problem dependent. Here the problem is considered as the placement of the test bits. So the fitness function is considered as the placement of test bits. So the selection process is dependent on the fitness function and hence the selected pattern will obey the condition which will be easily compressed by the 2n PRL compression process.

The genetic algorithm will stop when the required number of children produced by the algorithm in the fitness test. Now the obtained code-word is given to the decoder unit which will generate the efficient test sequence and the test sequence is applied to the various benchmark circuits. By using the genetic algorithm, around 35 patterns are generated out of which 5 patterns which are compatible for the compression are selected.

IV. Results and Discussion

The simulations are performed in Xilinx software. The C880 benchmark circuit is used to evaluate the algorithm. Table III is the comparison table between PRL technique and backtracking algorithm. The backtracking algorithm gives a better compression ratio for the various benchmark circuits. The compression ratio is calculated by CR% = $\frac{|Td|-|Te|}{Td}$*100, where $|Td|$ and $|Te|$ are sizes of input data and compressed data. Table IV shows the compression algorithm and the basic PRL algorithm [4] under optimum k-values.

Table V shows the comparison between the other compression algorithms. The backtracking algorithm shows the better compression ratio. If the value of 'k' is increased, the compression ratio is also increased accordingly. If the input value is increased, the value of 'k' is also increased to have better compression ratio. The comparison of compression ratio with various algorithms is shown in Fig. 2.

TABLE III
COMPARISON TABLE FOR C- CIRCUITS

S. No.	Circuit	Input Bits	Compressed bit		Compression ratio %	
			Back tracking	PRL technique	Back tracking	PRL technique
01	C432	36	12	21	67	42
02	C499	41	13	28	68	32
03	C880	60	16	35	73	42
04	C1355	73	21	49	71	32
05	C1908	33	11	21	67	37

TABLE IV
COMPARISON TABLE FOR S- CIRCUITS

S. No.	Circuit	Input Bits	Compressed bit		Compression ratio %	
			Back tracking	PRL technique [4]	Back tracking	PRL technique [4]
01	S9234	39273	15890	18053	59.54	54.03
02	S15850	76986	20800	25760	72.98	66.54
03	S38417	164736	73134	77014	55.61	53.25
04	S38584	199104	55760	65923	71.99	66.89
05	S5378	23754	9080	10696	61.77	54.97

TABLE V
COMPARISON TABLE OF VARIOUS ALGORITHMS

Name of the technique	Input bits	Compressed bits	Compression ratio	%
Back tracking technique	84	37	0.559	55.95
2^n–PRL technique	84	49	0.416	41.6
Back tracking with pattern length of 7	125	68	0.674	67.4
XOR Based Compression	126	63	0.50	50

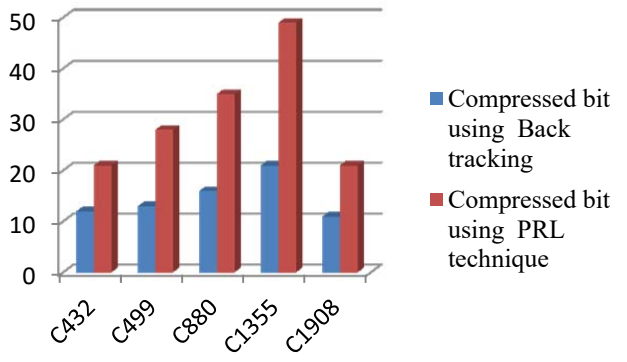

Fig. 2 Comparisons between compressed bit and CUT

Time reduction=> By PRL technique=1000200 ps and by Genetic algorithm=1000000ps

Table IV shows the results for genetic algorithm. It shows that the computational time is also reduced by using the genetic algorithm. The computation time of 200ps is reduced on an average when the test patterns are selected by genetic algorithm and then the proposed backtracking algorithm is applied. By comparing the processing time between the PRL technique and genetic algorithm, the genetic algorithm processing time is less.

V. Conclusion

The genetic algorithm is proposed to improve the compression ratio based on PRL algorithm and backtracking. This algorithm propagates the x- value and guarantee for the decoding. This genetic based tool for automatic test pattern generation is used instead of pattern generation by applying the normal TPG. Test patterns generated by the proposed tool can be used for generating the patterns for the specified compressing technique. Here the generated patterns are given to the 2^n- PRL decoder which results in providing a high efficiency data decoder. The time requirement for the execution of the algorithm is successfully reduced and the efficiency of the algorithm is increased by selecting the most compatible sequence.

TABLE IV
Selected Patterns for Various Circuits

Circuit	Compression ratio%			
	SC [15]	FDR [16]	VIHC [17]	Ours (genetic)
S5378	43.64	48.02	51.8	61.38
S9234	40.04	43.59	47.2	62.49
S38417	45.15	43.26	53.4	66.65
S15850	58.84	66.22	67.9	69.11
S38584	55.24	60.91	62.2	69.72

References

[1] X.Kavousianos, E. Kalligerous and D. Nikolos, "Optimal selective Huffman coding for test-data compression", IEEE Trans. Comput., vol.56, no. 8, pp. 1146 – 1152, Aug 2007.

[2] Zhanglei Wang, Member, IEEE, and KrishnenduChakrabarty, "Test Data Compression Using SelectiveEncoding of Scan Slices", IEEE Trans. on VLSI system, vol. 16, no. 11, Nov. 2008.

[3] P.T.Gonciari, B. Al-Hashimi, and N.Nicolici, "Variable – length input Huffman coding for system-on-a-chip test", IEEE Trans. Comput. Aided Des.Integr. Circuits Syst., vol. 22, no. 6, pp. 783-796, Jun 2003.

[4] X.Raun and R.Katti, "A efficient data – independent technique for compression test vectors in systems –on – a – chip", in Proc. IEEE comput. Soc. Annu. Symp.Emerging VLSI Technol. Archit., Washington, DC 2006, pp153-158.

[5] M.Tehranipoor, M.Nourani and K.Chakrabarty, "Nine-coded compression technique for testing embedded cores in SoCs," IEEE Trans. VLSI Syst., vol.13, no.6, pp. 719-731, Jun.2005.

[6] Kedarnath J. Balakrishnan and NurA.Touba, "Relationship between Entropy and Test Data Compression", in IEEE trans. on CAD of Integrated circuits and system, vol. 26, no.2, Feb 2007.

[7] Lung – Jen Lee, Wang – Dauh Tseng, Rung – Bin Lin, and Cheng – Ho Chang, "2^n Pattern Run-Length for Test Data Compression", in IEEE trans. on CAD of Integrated circuits and system, vol. 31, no.4, April 2012.

[8] Maoxiang Yi, Huaguo Liang, Lei Zhang and Wenfa Zhan, "A Novel X-ploiting Strategy for Improving Performance of Test Data Compression", in IEEE Trans. on VLSI system, vol. 18, no. 02, Feb. 2010.

[9] CinziaBernardeschi, Luca Cassano, Mario G.C.A. Cimino, Andrea Domenci, "GABES: A Genetic Algorithm Based Environment for SEU Testing in SRAM-FPGAs", in journal of system architecture, pp 1243 – 1254, 2013.

[10] HalinaKwasnicka and Michal Przewozniczek, "Multi population Pattern Searching Algorithm: A New Evolutionary Method Based on the Idea of Messy Algorithm", IEEE Trans. on evolution computation vol. 15 No. 5, pp 715 – 734, Oct. 2011.

[11] Jervan, G. Eles, P., ZeboPeng ,Ubar, R., and Jenihhin, M., "Test time minimization for hybrid BIST of core-based systems", VLSI test symposium, 2014.

[12] Jervan, G. Eles, P., ZeboPeng ,Ubar, R., and Jenihhin, M., "Hybrid BIST time minimization for core-based systems with STUMPS architecture", VLSI test symposium, 2014.

[13] T. Garbolino, G. Papa, Genetic algorithm for test pattern generator design, Applied Intelligence 32 (2) (2010) 193–204.

[14] A. Jas, J. Ghosh-Dastidar, and N. A. Touba, "Scan vector compression / decompression using statistical coding," in *Proc. IEEE VLSI Test Symp.*, Apr. 1999, pp. 114–121.

[15] A. Chandra and K. Chakrabarty, "System-on-a-chip test data compression and decompression architectures based on Golomb codes," *IEEE Trans. Computer-Aided Design*, vol. 20, pp. 113–120, Mar. 2001.

[16] "Frequency-Directed Run-Length (FDR) codes with application to system-on-a-chip test data compression," in *Proc. IEEE VLSI Test Symp.*, Apr. 2001, pp. 114–121.

[17] P. T. Gonciari, B. M. Al-Hashimi, and N. Nicolici, "Variable-length input Huffman coding for system-on-a-chip test," *IEEE Trans. Comput.-Aided Design Integr. Circuit Syst.*, vol. 22, no. 6, pp. 783–796, Jun. 2003.