Evolutionary Design of Reconfiguration Strategies to Reduce the Test Application Time

Jiří Šimáček, Lukáš Sekanina, Lukáš Stareček

Brno University of Technology, Faculty of Information Technology
Božetěchova 2, 612 66 Brno, Czech Republic
isimacek@fit.vutbr.cz, sekanina@fit.vutbr.cz, starecek@fit.vutbr.cz

Abstract. Recently, a method has been presented that allows a significant test application time reduction if some of gates of a digital circuit are reconfigured before test is applied. Selection of the gates for reconfiguration was performed using a very time consuming deterministic recursive search algorithm. In this paper, a new method is proposed for selection of the gates in order to reduce the test application time. The method utilizes an evolutionary algorithm which is able to discover very competitive reconfiguration strategies while the time of optimization is considerably reduced with respect to the original algorithm. Moreover, the user can easily balance the trade off between the number of test vectors and amount of logic that has to be reconfigured. Experimental results are reported for the ISCAS85 benchmark suite.

1 Introduction

One of the most significant properties of biological organisms is the ability to modify the conformation. Optimally-chosen conformation helps the biological organisms to perfectly manage elementary functions such as reproduction, sensing, communication, competition with others etc. Conformity between an organism and its environment constitutes what biologists call *adaptation* [1]. Reconfigurable architectures in fact implement the same concept in the world of electronic circuits. Different configurations of the same hardware can be activated to optimally perform quite different tasks, thus providing an obvious advantage to single-purpose circuits.

A circuit support for diagnostics and testing is one of the functionalities that are embedded in modern electronic chips. However, this functionality can be considered as time/area overhead because it is not directly utilized by end users. In a typical scenario, a singe purpose "user circuit" is equipped with a diagnosis subsystem that performs diagnostics and testing after fabrication (to identify faulty chips) and during lifetime of the system (such as BIST, on-line testing etc.). Unfortunately, costs of electronic chip testing have been growing steadily and typically amount to 40% of today's overall product cost [2]. In particular, the *test application time* which strongly depends on the number of *test vectors* needed to test a digital circuit significantly influences the product cost. Hence automatic test pattern generator tools have been used to reduce the number of

test vectors for a long time. High-quality test sequences in combination with full/partial scan techniques and other methods allowed designers to reduce the test application time significantly [3, 4, 5, 6, 7].

The reduction of test data volume is traditionally achieved by means of test data compaction [8, 9]. An unconventional approach to the test time reduction was introduced in [10]. The idea is to apply a smart circuit reconfiguration (i.e. to change the organism's conformance in biological terminology) before test is applied in order to reduce the number of test vectors. More precisely: It is known that automatic test pattern generator (ATPG) tool can generate k-vector test sequence T leading to $p\%$ fault coverage where k and p depends on structure and properties of a given circuit C, fault model used and user requirements. However, it was shown in [10, 11] that if logic function of some gates of C can be changed then a much shorter test T' can be generated, i.e. k can be significantly decreased (tens of percent, depending on circuit) simultaneously with having p almost unchanged. Therefore, the second circuit configuration is used only during test application to reduce the test application time. It is important to note that circuit topology remains unchanged during this reconfiguration. A method was proposed in [11] to find suitable gates for reconfiguration. However, the method is based on a deterministic recursive search which is very time consuming (days for mid-size circuits) and so impractical for designers and test engineers. The method was validated using only four benchmark circuits of the ISCAS85 benchmark suite [12].

The goal of this paper is to propose a new method for reduction of test vectors volume. The method should lead to a comparable quality of results wrt the previous approach; however, the time of computation has to be reduced. As there are many successful applications of evolutionary computing to hardware optimization and design [13, 14] the proposed method will be based on the evolutionary computing paradigm. Our goal is to generate a new circuit configuration, which differs from the original one as little as possible, but possesses better properties in terms of volume of required test vectors. The proposed algorithm utilizes a simple weight function to allow balancing the trade off between the number of test vectors and amount of logic that has to be reconfigured.

The rest of the paper is organized as follows. Previous work in the area of test time reduction using circuit reconfiguration is summarized in Section 2. Proposed method intended for selection of gates that will be reconfigured before test is applied is presented in Section 3. Section 4 gives an overview of experiments performed to evaluate the proposed method. It also compares the results with paper [11]. Section 5 is devoted to the analysis of obtained results. Conclusions are given in Section 6.

2 Previous Work

Conventional approaches to reduction of test application time are well covered in literature. This section surveys the method which is relevant for our research.

2.1 Reconfiguration Before Test Application

The principle of the method (which was initially proposed in [10]) is to identify gates of a circuit whose function has to be reconfigured before test is applied in order to reduce the number of test vectors. The reconfiguration should have the following properties: (i) The number of test vectors is reduced as much as possible. (ii) The number of reconfigured gates has to be minimized. (iii) The fault coverage is not influenced significantly (for a given fault model). (iv) Circuit connections remain unchanged. (v) Reconfiguration does not change the number of inputs and outputs of gates (for example, a two-input/one-output gate can be replaced only by a two-input/one-output gate).

After test is applied, the circuit is reconfigured back to its original configuration. Only the gates that have to be reconfigured will be implemented as reconfigurable; other gates remain implemented using a standard library.

2.2 Search Algorithm

Since the number of possible reconfigurations is n^r, where n is the number of gates of the circuit and r is the average number of possible replacements of a gate, an exhaustive search for an optimal configuration is intractable for real world circuits. Because the original method was based on enumeration, only the results for very small circuits (up to 13 gates) have been reported in [10]. In order to solve larger problem instances, a recursive search algorithm was proposed in [11]. This algorithm systematically reconfigures gate by gate, measures the resulting test length and fault coverage. When a particular gate reconfiguration leads to an improvement, the configuration is fixed and the algorithm is executed recursively from the next gate. Promising results have been reported for some of the ISCAS85 circuits even if the algorithm is terminated before the end of the complete search space exploration. In both cases the FlexTest tool was used to generate test vectors and calculate the fault coverage.

As discussed in [10, 11], the basic assumption of the proposed method is that gates are considered as black boxes and only the circuit structure is tested because it is expected that failures in components will propagate outside the component. A possible problem is that demanded reconfigurable two-function gates may be functional in one mode and damaged in the other. This could lead to undetectable faults or false alarms. However, that strongly depends on the implementation of reconfigurable gates. Recall that a 100% fault coverage is not nowadays achievable for complex real-world circuits. Hence some faults will always remain unrecognized. Although the method can leave some faults unrecognizable too, it allows reducing of the test vectors volume for a reasonable cost.

2.3 Example

Figure 1 shows a 3-input/8-output decoder (dec3to8) which consists of eleven gates (seven 3-input NOR gates, three inverters and a 3-input AND gate). FlexTest was utilized to derive the test with 100% fault coverage. A stuck-at-fault

model was considered for AMI 1.2 um technology. The resulting test contains eight vectors: 100, 000, 111, 110, 011, 101, 010 and 001, i.e. it is the trivial test.

Logic function of four gates of this circuit was modified as shown in Figure 1. Modified gates are shown in boxes. The three inverters were reconfigured to operate as simple wires (buffers) and the AND gate now operates as the NOR gate. Note that in the X/Y notation, X denotes the original function and Y denotes the modified function. Again, FlexTest was used to find a test with 100% fault coverage. The new test contains only four test vectors (100, 000, 010 and 001) which represents a 50% reduction. Other experiments are summarized in papers [10, 11].

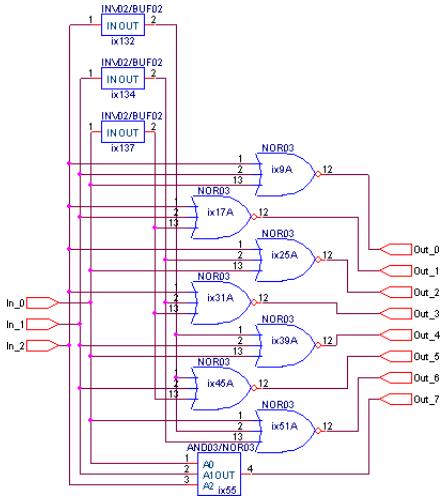

Fig. 1. Circuit dec3to8. Reconfigured gates are shown as boxes.

2.4 Possible Implementation Scenarios

An open problem (not addressed in this paper) is how to implement the reconfigurable gates. A straightforward approach is to employ multiplexing of the "user" and "test" function for selected gates (Fig. 2). This solution has a reasonable overhead, especially when the reconfigurable gate is optimized at the transistor level as shown in [15]. However, the select inputs of multiplexers are not considered during test pattern generation by ATPG. Hence it is necessary to use additional test vectors to test the select-inputs which increases the overall test time application. This solution is acceptable only in some cases.

Another solution could utilize so-called *polymorphic gates*. Polymorphic gates are unconventional circuit components that are not supported by existing synthesis tools. A polymorphic gate is capable of switching among two or more logic functions. However, the selection of the function is performed unconventionally.

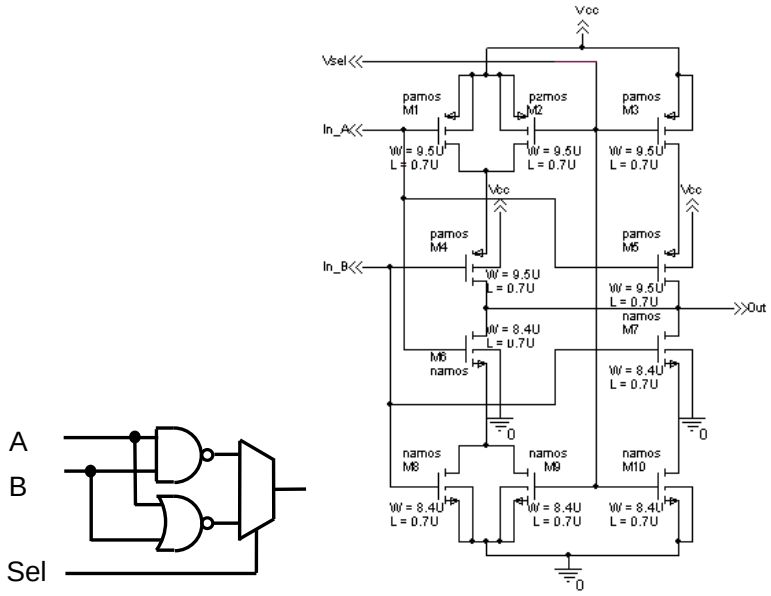

Fig. 2. Reconfigurable NAND/NOR gate based on a multiplexer and its optimized transistor-level implementation according to [15]

The logic function of a polymorphic gate depends on some external factors, e.g. on the level of the power supply voltage (V_{dd}) [16, 17, 18, 19]. Figure 3 shows the NAND/NOR gate controlled by V_{dd} which was fabricated using AMIS CMOS 0.7 micron technology. In case of reduction of test vectors volume it is assumed that the circuit will operate with a slightly different V_{dd} in test mode. Selected gates will then perform differently wrt the user mode and the test application time will be shorter (under our assumptions). As there are no select signals for polymorphic gates the problem with their testing does not exist. On the other hand it must be investigated whether all faults of the normal mode of the gate remain detectable in the test mode. There are other issues such as that a polymorphic circuit may have different timing parameters or power consumption during normal operation and test application.

3 Proposed Method

The proposed method utilizes a steady-state evolutionary algorithm (EA) which operates over chromosomes composed of n integers. New individuals are created using mutation applied on best-scored individuals of the population. Crossover is not utilized. The fitness function integrates the criteria given in Section 2.1 using weight coefficients.

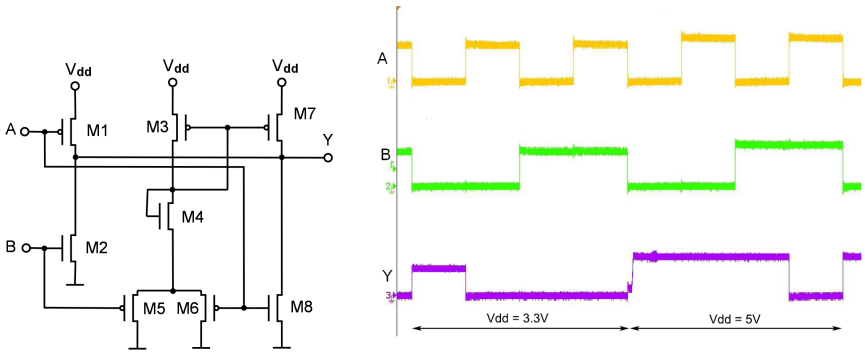

Fig. 3. Polymorphic NAND/NOR gate controlled by V_{dd} and its measured behavior according to [19]

3.1 Notation

Let G be a set of all gate types which can appear in the target design and $\mathcal{C}$ be a gate classification function such that $g_1, g_2 \in G$ belong to the same class if and only if they have the same number of inputs and outputs and g_1 can be replaced by g_2 (and vice versa) in the target circuit. We lift $\mathcal{C}$ to the sequence of gates such that

$$\mathcal{C}(g_1 \dots g_n) = \mathcal{C}(g_1) \dots \mathcal{C}(g_n).$$

Additionally, $len(s)$ denotes the number of symbols in a sequence s and $\delta(u, v)$ denotes the number of positions, where strings u and v differ.

3.2 Circuit Configuration

A digital circuit consists of the finite number of gates and an interconnection network. In this work, we only consider the sequence of gate types $g_1 \dots g_n$ (in the order given by a circuit's netlist) as a circuit configuration, because the interconnection network is always fixed. Each chromosome is then composed of just one circuit configuration.

3.3 Fitness Function

Our goal is to generate a new circuit, which differs from the original one as little as possible, but the test lenght is reduced. Fault coverage should not be modified significantly. Thus, we want to minimize the function

$$f(nc, oc) = A * (1 - tCov(nc)) + B * \frac{vc(nc)}{vc(oc)} + C * \frac{\delta(nc, oc)}{len(oc)},$$

where nc, oc denote the new and original configuration respectively, $tCov(x) \in \langle 0, 1 \rangle$ is a value which expresses the fault coverage of a given configuration, and $vc(x)$ denotes the volume of required test vectors. Coefficients A, B, C represent the weight of each property.

Algorithm 1: Evolutionary Algorithm

Input: input configuration c, population size s, and mutation probability p_{mut}
Output: output configuration x

```
   /* seeding phase */
1  P ← ∅;
2  while |P| < s do
3      P ← P ∪ {modify(c, p_mut)};
4  while ⟨terminating condition not satisfied⟩ do
       /* reproduction phase */
5      P' ← ∅;
6      while |P'| < s do
7          select x ∈ P randomly;
8          P' ← P' ∪ {modify(x, p_mut)};
       /* reduction phase */
9      P ← P ∪ P';
10     while |P| > s do
11         select x ∈ P such that f(x, c) ≥ f(y, c) for any y ∈ P;
12         P ← P \ {x};
13 return x ∈ p such that f(x, c) ≤ f(y, c) for any y ∈ P;
```

3.4 Mutation

Mutation takes an input configuration and flips each gate with the probability p_{mut}. A new gate is selected randomly from the set of all gates belonging to the same class (according to $\mathcal{C}$).

3.5 Evolutionary Algorithm

The evolutionary algorithm (Algorithm 1) starts with seeding of population P by randomly modified input configuration c. The modifications as well as mutations are performed by function *modify*. Then, it repeats reduction and reproduction phases. In the former the worst individuals wrt f are being iteratively removed until the size of the population meets the required criterion. The reproduction phase then generates new individuals by modifying configurations which are picked randomly from the original population. The condition which terminates the main loop of EA can be either the program running time or the number of generations being generated. As a result, the algorithm picks the best individual from the last population.

4 Experimental Results

As for experiments reported in [11], G contains 56 standard gates (with up to 4 inputs) which are also supported by the AMIS library. In addition, 7 gates (with

up to 8 inputs) were included to G to cover all the gates used in the ISCAS85 circuits. The FlexTest tool is used to generate test vectors and calculate the fault coverage. The results of proposed evolutionary algorithm are compared with the recursive search algorithm using the ISCAS85 benchmark suite. Main features of the ISCAS85 circuits (such as the number of gates, test length and fault coverage) are given in Table 2 (column 'Original circuit'). All experiments were permormed on a server with 2 x Dual Core AMD Opteron 2220.

Performing a single experiment is very time consuming because of using the FlexTest tool in the loop. Hence we have firstly investigated different settings of our EA on circuits c499 and c1355 and then performed a final set of experiments with all the benchmark circuits. For the first experiments we have used $A = 1000$, $B = 100$, $C = 10$, the probability of mutation $p_{mut} = 0.005$ and the population of 1000 individuals. Various modifications of EA were tested using the c499 circuit. Resulting values are given in Table 1. Figure 4 shows the relation between the number of test vectors and the number of reconfigured gates for the c499 circuit. It can be seen that EA produces various solutions and one can easily identify a Pareto front in the figure.

Table 1. Summary of experiments for c499 with basic setup: $A = 1000$, $B = 100$, $C = 10$, popsize = 1000, $p_{mut} = 0.005$, 10 independent runs, 1000 generations

Modification wrt	gates			test vectors			fault coverage [%]			Mean
basic setup	min.	max.	mean	min.	max.	mean	min.	max.	mean	t [h]
500 gen.	10	32	23.8	29	34	31.2	99	100	99.6	3.29
250 gen.	11	30	31.4	30	34	31.4	100	100	100	1.80
100 gen.	19	27	24	30	33	31.4	99.73	100	99.97	0.72
250 gen./$p_{mut} = 0.01$	24	45	33.9	28	32	29.6	99.33	100	99.91	1.75
250 gen./$p_{mut} = 0.0025$	14	21	17.5	30	34	32.3	99.47	100	99.82	1.72
250 gen./$A = 700$	14	40	24.3	29	34	30.8	98.94	100	99.77	1.87

Figure 5 shows the results for the c1355 circuit obtained from 10 independent runs. Note that applying the FlexTest on this circuit (which consists of 546 gates) leads to the 108-vector test sequence and 99.49% fault coverage. The fault coverage was slightly reduced after using the proposed method; however, the test length was significantly reduced to 27 – 37 test vectors when 39 – 70 gates are reconfigured. The average runtime is 4.2 hours for 500 generations.

Table 2 summarizes the results obtained for the complete set of ISCAS85 circuits using the proposed evolutionary algorithm and the recursive search. EA has been applied with the following setting: $A = 1000$, $B = 100$, $C = 10$, popsize = 1000, $p_{mut} = 0.005$, 1000 generations, a *single* run per circuit. The time of evolution depends on the complexity of a particular circuit. An example of EA run is given in Fig. 6 which shows the progress of fitness score and the number of test vectors for the best individual in case of the c7552 circuit. As the recursive algorithm presented in [11] is deterministic we allowed the algorithm to run (i)

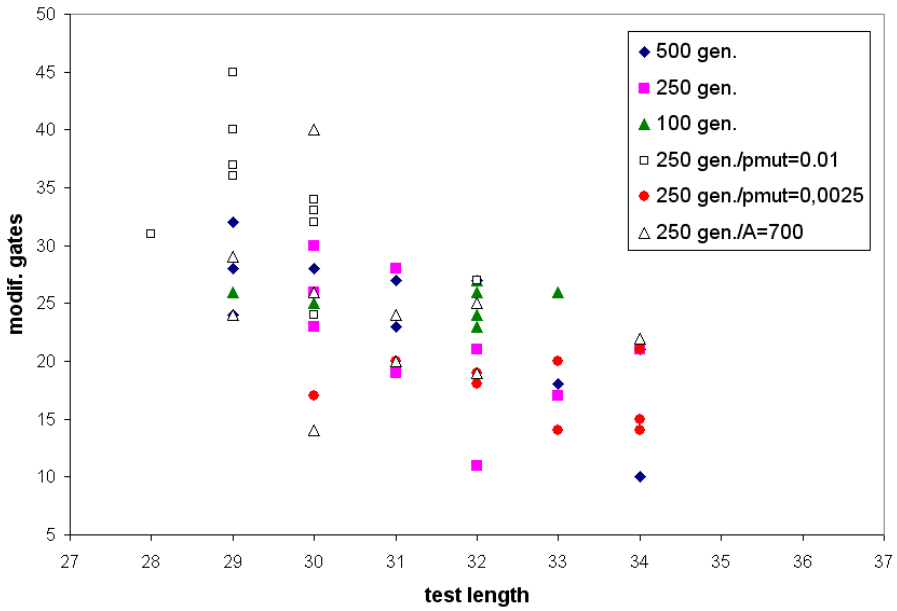

Fig. 4. Test length vs the number of modified gates for the c499 circuit (60 runs with different setting)

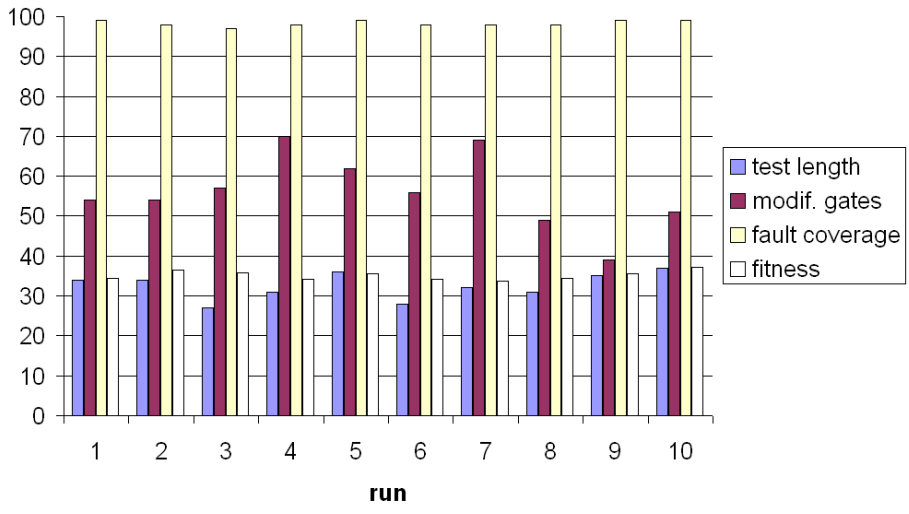

Fig. 5. Results of 10 independent runs for the c1355 circuit.

for the same time as EA and (ii) for the maximum limit of 96 hours. In most cases of (ii), the computation was not terminated within this time limit.

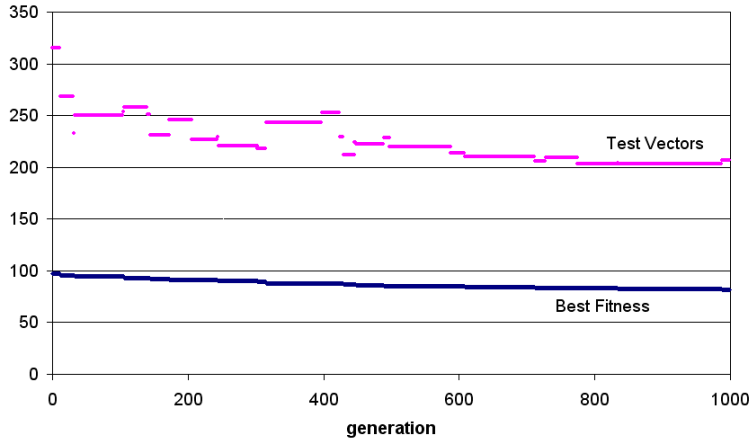

Fig. 6. The progress of fitness score and the number of test vectors for the best individual (the c7552 circuit).

5 Discussion

We can see from Table 2 that the proposed method has reduced the number of test vectors by 49.0% and reconfigured 6.4% of gates in average (all benchmarks counted). Note that only a single run was performed (9.5 hour per circuit in average). The recursive algorithm has achieved a reduction of 31.4% test vectors and reconfigured only 2.6% of gates (when 96 hours were allowed). The recursive algorithm has increased the fault coverage measure by 0.4% in average and the EA has decreased the same measure by 1.9%. wrt the original fault coverage. Some particular results are interesting: No result was discovered by the recursive algorithm for the c1355 circuit; however, the EA has found a significant reduction of test vectors volume. Both algorithms have achieved similar results for the c499 circuit. The recursive algorithm has produced much better results for the c6288 circuit (where only 10 gates have to be reconfigured (94 in case of EA) to get the same test length). A significant test volume reduction has been achieved for c7552 circuit using EA; however, the solution is not probably acceptable as fault coverage is decreased by 6.77%. We expect that better results will be obtained in case that EA is executed multiple times.

In summary, the proposed evolutionary algorithm has two main advantages in comparison with the deterministic recursive search algorithm. Firstly, it produces many different solutions which allow the designer to balance the trade off

Table 2. Parameters of the original ISCAS85 circuits and the circuits modified using the evolutionary algorithm and the recursive search (the time allowed as for EA vs 96 hours allowed). Notation: rg – the number of reconfigured gates, tl – test length, fc – fault coverage [%], t - runtime

	Original Circuit			Evol. Algorithm				Recursive (t[h])			Recursive (96h)			
	gates	tl	fc	rg	tl	fc	t[h]	rg	tl	fc	rg	tl	fc	t[h]
c17	6	9	100	4	5	100	3.5	3	5	100	3	5	100	0.1
c432	160	102	99.24	22	50	99.07	4.9	22	63	99.82	27	57	99.82	96.0
c499	202	67	98.94	31	30	100	4.9	34	33	100	36	28	100	96.0
c880a	383	104	100	63	49	99.50	5.0	24	74	100	30	66	100	96.0
c1355	546	108	99.49	67	31	98.52	7.7	0	108	99.49	0	108	99.49	1.9
c1908	880	163	99.52	90	48	95.54	7.0	26	119	99.74	30	114	99.79	96.0
c2670	1269	189	95.74	99	92	95.33	8.7	29	155	96.50	59	109	96.73	75.5
c3540	1669	252	96.00	103	171	94.32	9.8	22	214	96.48	57	196	97.20	96.0
c5315	2307	190	98.88	123	111	92.32	9.0	7	182	98.92	58	127	98.99	96.0
c6288	2416	46	99.56	94	35	98.31	9.5	7	39	99.56	10	36	99.56	31.5
c7552	3513	371	98.26	152	207	91.49	34.7	14	351	98.40	36	325	98.41	96.0

between the number of test vectors and amount of logic that has to be reconfigured. Secondly, the EA generates a reasonable solution for larger circuits much faster than the recursive algorithm. On the other hand, as EA does not strictly keep the fault coverage equal to or higher than the original value, the resulting solution can exhibit slightly lower fault coverage. However, this behavior can be eliminated by setting stronger requirements in the fitness function.

6 Conclusions

In this paper, we have presented an alternative method for selection of gates that have to be reconfigured before test is applied in order to reduce the test application time. We have shown on the ISCAS85 benchmark suite that the proposed method is able to achieve competitive results while the time of optimization is reduced wrt the deterministic search. In future, we plan to use a truly multi objective algorithm to easily discover the Pareto-optimal solutions.

Acknowledgments

This work was partially supported by the Czech Science Foundation under contract numbers GP103/10/1517 and GD102/09/H042, the BUT FIT grant FIT-10-S-1 and the research plan Security-Oriented Research in Information Technology, MSM 0021630528.

References

[1] Fisher, R.A.: The Genetical Theory of Natural Selection. Oxford: Clarendon Press (1930)

[2] Wang, L.T., Stroud, C.E., Touba, N.A.: System-on-Chip Test Architectures: Nanometer Design for Testability. Morgan Kaufmann (2007)
[3] Park, S.: A partial scan design unifying structural analysis and testabilities. Int. J. Electronics **88**(12) (2001) 1237–1245
[4] Xiang, D., Patel, J.H.: Partial scan design based on circuit state information and functional analysis. IEEE Trans. Computers **53**(3) (2004) 276–287
[5] Efthymiou, A., Bainbridge, J., Edwards, D.A.: Test pattern generation and partial-scan methodology for an asynchronous soc interconnect. IEEE Trans. VLSI Syst. **13**(12) (2005) 1384–1393
[6] Makris, Y., Orailoglu, A.: Property-based testability analysis for hierarchical rtl designs. In: Proceedings of IEEE ICECS99, 6th IEEE International Conference on Electronics, Circuits and Systems, IEEE Computer Society (1999) 1089–1092
[7] Lee, J., Touba, N.A.: Low power test data compression based on lfsr reseeding. In: 22nd IEEE International Conference on Computer Design: VLSI in Computers & Processors (ICCD 2004), IEEE Computer Society (2004) 180–185
[8] Das, S.R., Ramamoorthy, C.V., Assaf, M.H., Petriu, E.M., Wen-Ben, J., Sahinoglu, M.: Fault simulation and response compaction in full scan circuits using HOPE. IEEE Tran. on Instr. and Meas. **54**(6) (2005) 2310–2328
[9] Pomeranz, I., Reddy, S.M.: Static test compaction for multiple full-scan circuits. In: 21st International Conference on Computer Design (ICCD 2003), VLSI in Computers and Processors, IEEE Computer Society (2003) 393–396
[10] Sekanina, L., Starecek, L., Kotasek, Z., Gajda, Z.: Polymorphic gates in design and test of digital circuits. Int. J. of Unconventional Computing **4**(2) (2008) 125–142
[11] Starecek, L., Sekanina, L., Kotasek, Z.: Reduction of test vectors volume by means of gate-level reconfiguration. In: Proc. of 2008 IEEE Design and Diagnostics of Electronic Circuits and Systems Workshop, IEEE (2008) 255–258
[12] Brglez, F., Fujiwara, H.: A neutral netlist of 10 combinational benchmark circuits and a target simulator in Fortran. In: Proceedings International Symposium on Circuits and Systems (ISCAS), Kyoto, Japan (1985) 695–698
[13] Drechsler, R.: Evolutionary Algorithms for VLSI CAD. Kluwer Academic Publishers, Boston (1998)
[14] Zebulum, R., Pacheco, M., Vellasco, M.: Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. The CRC Press International Series on Computational Intelligence (2002)
[15] Starecek, L., Sekanina, L., Gajda, Z., Kotasek, Z., Prokop, R., Musil, V.: On properties and utilization of some polymorphic gates. In: Proc. of 6th Electronic Circuits and Systems Conference, FIIT STU, Bratislava (2007) 77–81
[16] Stoica, A., Zebulum, R.S., Keymeulen, D.: Polymorphic electronics. In: Proc. of Evolvable Systems: From Biology to Hardware Conference. Volume 2210 of LNCS., Springer (2001) 291–302
[17] Stoica, A., Zebulum, R.S., Keymeulen, D., Lohn, J.: On polymorphic circuits and their design using evolutionary algorithms. In: Proc. of IASTED International Conference on Applied Informatics AI2002, Insbruck, Austria (2002)
[18] Stoica, A., Zebulum, R., Guo, X., Keymeulen, D., Ferguson, I., Duong, V.: Taking Evolutionary Circuit Design From Experimentation to Implementation: Some Useful Techniques and a Silicon Demonstration. IEE Proc.-Comp. Digit. Tech. **151**(4) (2004) 295–300
[19] Ruzicka, R., Sekanina, L., Prokop, R.: Physical demonstration of polymorphic self-checking circuits. In: Proc. of 14th IEEE International On-Line Testing Symposium, IEEE (2008) 31–36