DOI 10.31114/2078-7707-2019-1-2-6

Automated Evolutionary Design of Fault-Tolerant Logic Circuits

S.V. Gavrilov, D.V. Telpukhov

Institute for Design Problems in Microelectronics of RAS, Moscow, nofrost@inbox.ru

***Abstract* — Currently, fault tolerance of electronic equipment requires special attention. Application field of integrated circuits expands; at the same time, permissible limits of destabilizing effects, which increase vulnerability of integrated circuits, grow. It is often necessary to take into account requirements for fault tolerance and apply various methods and tools for developing the most stable circuits as early as at the design stage. As the result, there is a large demand for design automation tools for failure- and fault-tolerant integrated circuits.**

This paper presents a method for combinational circuits synthesis based on general principles of evolutionary algorithms. This method allows synthesizing comparatively small logic circuits that are resistant to random failures induced by hits of heavy charged particles.

***Keywords* — evolutionary synthesis, fault tolerance, combinational circuits, genetic algorithms.**

I. INTRODUCTION

The problem of building fault-tolerant combinational circuits at logical level formulated as synthesis of reliable circuits from unreliable components was first posed in the fundamental work of Von Neumann [1] and was further developed in works of W.H. Pierce, J.G. Tryon, N. Pippenger [2-5]. Within the framework of this problem, a large number of different majority approaches were developed for protecting circuits at architectural level, such as logic circuits with cascade triple redundancy [2], with fourfold redundancy [3,4], randomly interlaced logic [5], etc. However, despite the large number of scientific publications, in practice, archaic methods of triple redundancy are still used to protect combinational circuits. This is because there are no clear criteria for evaluating effectiveness, and the methods are not enough studied for a large range of test circuits for real technology libraries.

Within the proposed approach, we suggest to generalize the problem of improving fault tolerance of a given logic circuit to the problem of fault-tolerant combinational circuit synthesis. The problem of logical synthesis of circuits optimized for given parameters is, in fact, the task of selecting solution that is optimal or close to optimal from the whole space of possible solutions.

If we have a logical function defined by expression or truth table, it is obvious that there is infinite set of logic circuits that implement this function. The task is to find a circuit that is satisfactory according to given criteria and belongs to the whole space of possible solutions.

Conventional methods of improving fault tolerance transform original circuit without changing its logical function and implement one specific solution for each circuit. Exhaustive search is impossible; therefore, various heuristics are often used in problems of logical synthesis (Espresso, MIS II, ...).

In this paper we propose using genetic methods [6,7] to search for circuits that are near-optimal for criteria considering fault tolerance and architectural redundancy.

II. DATA REPRESENTATION FOR GENETIC ALGORITHM

Let us define basic data structures for combinational circuit representation in the form of phenotype and genotype; and define basic operators on these structures. We define phenotype as immediate netlist of combinational circuit. Its primary purpose is getting output vector when input stimulus are fed. Genotype backs genetic procedures such as crossover, mutation, etc.

A. Phenotype

Circuit is represented as the list of input labels, list of output labels and dictionary of gates. Here dictionary is associative array with keyed access similar to that in Python, where key is logic gate label and array element is structure containing gate type and labels of gates connected to its outputs. Consider ISCAS testbench circuit C17 as an example of phenotype description (Fig. 1).

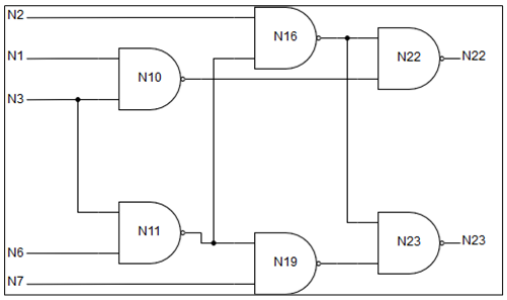

Fig. 1. Testbench circuit ISCAS'85 C17

This circuit is represented as the following structure (Fig. 2).

```
Inputs: ['N1', 'N2', 'N3', 'N6', 'N7']
Outputs: ['N22', 'N23']
Elements: {   'N10': ('NAND', ['N1', 'N3']),
              'N11': ('NAND', ['N3', 'N6']),
              'N16': ('NAND', ['N2', 'N11']),
              'N19': ('NAND', ['N11', 'N7']),
              'N22': ('NAND', ['N10', 'N16']),
              'N23': ('NAND', ['N16', 'N19'])   }
```

Fig. 2. Phenotype of a circuit ISCAS'85 C17

Simulation function is the base class method for circuit phenotype; it is the basis of other procedures related to fault tolerance evaluation, etc. Its input is binary vector of input stimulus and binary error vector, length of which is the number of gates: if the gate is faulty, the corresponding element of error vector takes value "1", if there is no error, value is "0". Function output is the circuit reaction to the impact.

B. Genotype

To present circuit in the form of chromosome we have to implement some one-dimensional representation that is unique for given circuit structure. Ordered linear representation of the combinational circuit was developed for this purpose. Similar to phenotype, basic element of this representation is structure containing gate type and references to gates linked to the outputs, except that gate position in the array of gates is given instead of gate label. Logic gates in chromosome are sorted according to their topological order. In addition, positions of gates connected to circuit outputs, as well as characteristic of number of inputs, are given. Genotype of a circuit C17 from ISCAS testbench is shown in Fig. 3.

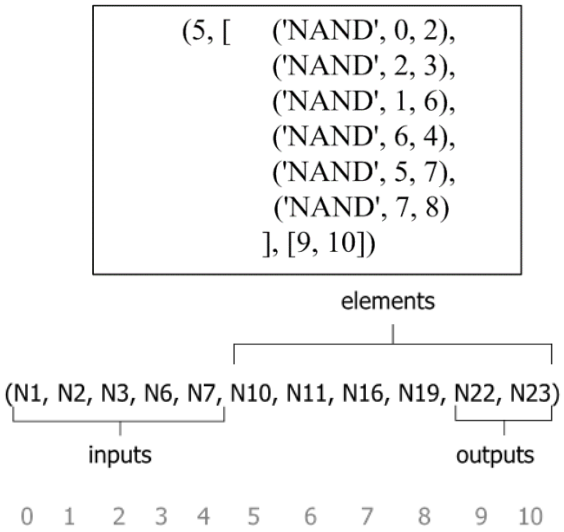

Fig. 3. Genotype of a circuit ISCAS'85 C17

This representation, as opposed to phenotype, complicates even the easiest operators concerning circuits, such as merging circuits or replacing subcircuits, etc., because permanent re-numbering and re-sorting is needed; however, it facilitates actions related to genetic operators. The proposed chromosome representation has several advantages. First, it allows encoding circuits with arbitrary number of gates, which is not typical for evolutionary synthesis methods [8]. Second, this representation allows easy implementation of main genetic operators, such as mutation and crossover, even with varying chromosome lengths. Note that in this context, crossover means not only exchange of subcircuits, but also implicit interconnections exchange.

III. BASIC GENETIC OPERATORS

Further we consider basic genetic operators required for efficient algorithm implementation.

Initial population creation. To create initial population, we implemented generator of arbitrary circuits. This stage begins with getting the number of inputs and outputs. The number of logic gates in the circuit is determined by selecting a random integer from the normal distribution with expectation $\mu = (m + n)$ and standard deviation σ, the value of which is transferred to the generation function. Next comes the process of generating of logic gates. For each gate, its type is selected randomly from the standard library. Then, input labels are selected for this element, indicating the circuit inputs or the outputs of already existing elements. The process of creating such schemes continues until the population is completely filled. The rate of convergence and efficiency of the entire algorithm depends on the quality of the initial population.

In some cases it is possible to effectively introduce parts of circuits synthesized from the reference function by traditional methods. However, this often leads to rapid reproduction of exactly these synthesized circuits.

Fitness function. Fitness function of the proposed genetic algorithm is as follows:

$$f = \varepsilon + \lfloor \varepsilon \rfloor \cdot \alpha \qquad (1)$$

where f is fitness function, ε – degree of proximity of the individual function with the reference function, α – coefficient of logic sensitivity to random failures [9,10].

This metric is based on Von Neumann probabilistic error model [1], which implies that output value of any gate can be inverted independently of other gates with some fixed probability. If we define binary vector $\bar{X}$ as input vector and vector $\bar{e}$ with logic "1" at positions for faulty gates as error vector, coefficient of logic sensitivity to random failures is defined as follows:

$$\alpha = \frac{1}{2^N} \sum_{\bar{X},\bar{e},|\bar{e}|=1} E(\bar{X},\bar{e}) \quad , \qquad (2)$$

where N is the number of inputs and $E(\bar{X},\bar{e})$ denotes characteristic function of set of vector pairs:

$$E(\bar{X},\bar{e}) = \begin{cases} 1, if\ set\ (\bar{X},\bar{e})\ results\ in\ error \\ 0, otherwise \end{cases}$$

Coefficient of logic sensitivity for a circuit is the sum of gate observabilities $= \sum_{i\in\Omega} o_i$; it characterizes average number of faulty gates, that is, the gates for which error propagates to circuit outputs. Gate observability is the probability that failure at this gate is not masked and affects circuit output given that no errors occurred at the other gates. Observability is calculated by the formula:

$$o_i = \frac{1}{2^N}\sum_{\bar{X}} E(\bar{X}, \bar{e}_k),$$

where vector $\bar{e}_k$ has single "1" at position e_i, while all the other values $e_{j\neq i} = 0$.

Computational complexity of this metric is linear with respect to the number of gates, which, together with the methods of bit-parallel modeling and Monte-Carlo methods, allows using this metric for relatively large circuits. The proposed coefficient does not depend on the probability of gate failure. Thus, it can be used at early stages of fault-tolerant circuits design. This metric is effective for comparing fault-tolerant methods when element base and operating conditions of the circuit are not known. For most practical applications when probability of gate failure tends to zero — this approximation is the most accurate, being tangent to the graph of error polynomial at point zero.

Proximity value ε varies from 0 to 1 and can be calculated in various ways. The simplest method is simulation and counting deal of coincidences with reference. It is more efficient to count deal of coincidences for each output of the circuit separately. That provides more information for genetic algorithm and speeds up the process of convergence of evolutionary search.

As we can see from formula (1), fault-tolerance characteristics of the circuit start to effect fitness function only when ε=1, in other words, when circuit becomes equal to reference function. That provides smooth evolutionary search: first, circuits that implement the required function; then, circuits with better fault tolerance.

Selection of parents. There are two popular methods to apply when performing selection in genetic algorithms, roulette wheel selection and tournament selection. Both use probability to create bias in choosing fitter chromosomes to serve as the parents. We implemented both in our study, but they showed almost the same properties as applied to our problem.

Crossover. Crossover is crossing two parent chromosomes and producing two offspring. This is done by selecting two cut points on the parent chromosomes, followed by exchange of sections. For this, two individuals need to be randomly selected from the parent pool. After that, a random real number is generated in the range from 0 to 1. This number is compared with the crossover probability, which is transmitted as a parameter of the crossover operator. If the resulting number does not exceed the probability, a crossing occurs. With 2-point crossover, the parent chromosomes are divided into three sections, the positions of the two separators d1 and d2 are determined randomly. After this, descendants are created by concatenating sections of the parent chromosomes. The created descendants are placed in the population, and the parents participating in the crossing are removed from the population. Since this process is random, there may be a situation in which crossing does not occur. In this case, the parents remain in the population. The following is an example of 2-point crossover (Fig. 4):

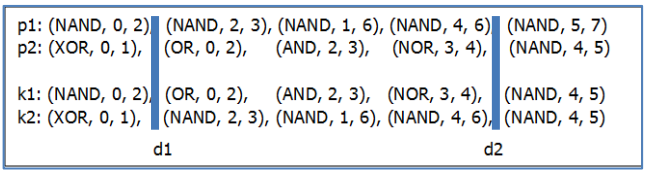

Fig. 4. The example of 2-point crossover

As the result, it is possible to exchange not only element types, but also some of their interrelations. Note that such cross operation can be applied to chromosomes, lengths of which differ. Crossover with two cut points can be implemented in similar way.

Mutation. Individual mutation is distortion of some chromosome genes, that is, alteration of several element types or interconnects. The mutation operation is applied to all individuals in the population, obtained after crossover. For each allele, a random real number is generated from 0 to 1, if this number does not exceed the probability of mutation, the current allele mutates, forming a new genotype pattern. At this stage, we create an alternative population, which consists of mutant individuals and individuals that have not undergone mutations. The probability of a mutation is determined by the parameter that is passed to the mutation function. The following is an example mutation process (Fig. 5):

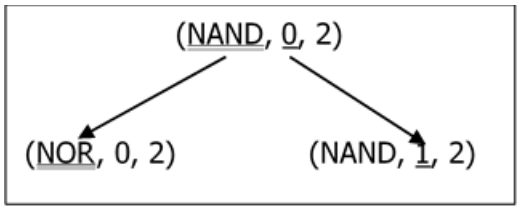

Fig. 5. The example of mutation process

It is proposed to use adaptive mutation method, which implies variable probability of gene mutation depending on the current rate of convergence of evolutionary process. In case of long stagnation, probability of mutation for each gene gradually increases until genetic algorithm leaves the local optimum.

Selection. Selection is the process of selecting individuals for new population. As part of selection, we can use asymmetric roulette wheel method, race method, or simple ranking method, when n best adapted individuals are selected. For a number of cases the so-called "elitism" is effective, when the most adapted individual is guaranteed to enter new population. This technique makes the process of evolutionary search more predictable, but can lead to premature convergence to a local optimum.

IV. Basic algorithm structure

Fig. 6 presents basic flow of fault-tolerant combinational circuit synthesis with the use of genetic algorithm.

At the first stage initial population is generated. After that, fitness function for each individual is estimated. At the next stage parents for crossing are selected. This choice is made either by asymmetric roulette wheel method or by race method. After that, crossover and mutation of a certain fraction of descendants are realized. After that, initial population and population of descendants are combined and the stage of selection into new population is implemented. After this, a new cycle of genetic algorithm is started.

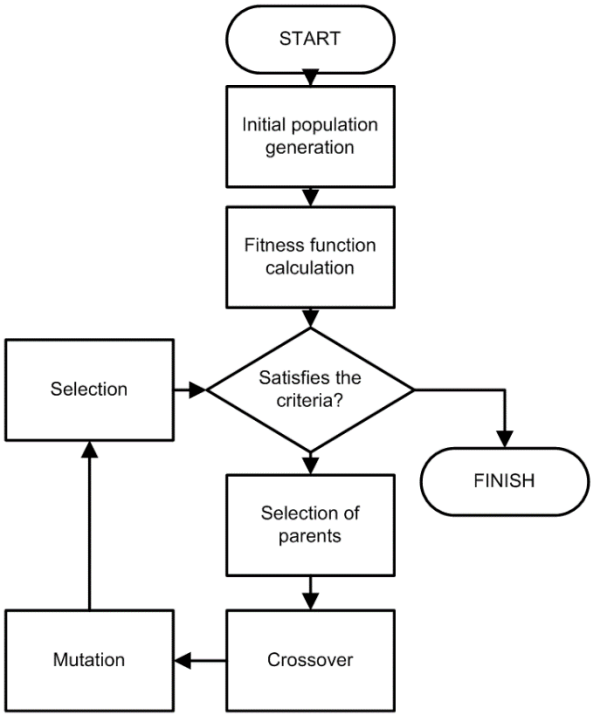

Fig. 6. Basic structure of genetic algorithm

To speed up the process of convergence and reduce the problems associated with premature convergence to local extremes, a number of methods related to the use of dynamic fitness function have been proposed, as well as methods of adaptive mutation. Also, we proposed that fitness function of genetic algorithm should include entropy parameter of combinational circuits alongside with such parameters as proximity to reference function and logical sensitivity to failures. This allows us to cut off trivial cases where structurally very simple solution immediately turns out to be quite close to reference function, but cannot get closer to it by any gradual changes. The decisions we made resulted in the development of software package in which we could implement synthesis of small combinational circuits with improved fault tolerance parameters.

V. Experimental results

In the course of adjusting of genetic algorithm parameters, large number of computational experiments were carried out. During this process, we built a number of combinational circuits with improved fault tolerance characteristics.

We estimated average time spent for our evolutionary synthesis. It differs greatly depending on number of inputs and especially outputs.

Table 1

Average runtime for evolutionary synthesis

	2 inputs		3 inputs		4 inputs	
	1 out	2 out	1 out	2 out	1 out	2 out
Time, sec	0.14	4.29	0.275	5.242	0.223	7.045

Fig. 6 shows an example of evolutionary synthesis of full adder circuit. Average value of fitness function across the population is shown in blue, red is the maximum value, and yellow is average value of entropy. Synthesis process required less than 300 generations.

Fig. 7. Genetic synthesis of full adder

VI. Summary

The paper presents a method for synthesis of fault-tolerant combinational circuits based on genetic algorithm. Basic structure of the algorithm is described, as well as some details of data representation and main genetic operators. Failure metric of combinational circuit is embedded into fitness function of the algorithm. This metric characterizes the average number of unreliable gates, that is, the gates whose failures affect circuit output.

The method shows good convergence for small circuits; first, selection is carried out with respect to proximity of the function to the reference one; then, fault tolerance metric is taken into account.

We should point out a drawback of the method, namely, the algorithm can appear to be understable. Currently, it can happen that the appropriate combinational circuit is not found or the algorithm works too long because of falling into a local optimum. Such behavior

additionally constrains possible application scope of this method.

Further work will be focused on improving predictability of the algorithm, increasing convergence rate and extending the range of applicability to medium-sized logic circuits.

REFERENCES

[1] J. von Neumann, “Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components”, Automata Studies, C.E. Shannon and J. McCarthy, eds., Princeton Univ. Press, 1956, pp. 43-98.

[2] W.H. Pierce, Failure-Tolerant Computer Design, Academic Press, 1965.

[3] J.G. Tryon, “Quadded Logic”, Redundancy Techniques for Computing Systems, R.H. Wilcox and W.C. Mann, eds., Spartan Books, 1962, pp. 205-228.

[4] P.A. Jensen, Quadded NOR Logic, IEEE Trans. Reliability, vol. 12, no. 3, Sept. 1963, pp. 22-31.

[5] N. Pippenger, Developments in ‘The Synthesis of Reliable Organisms from Unreliable Components’. Proc. Symposia in Mathematics, American Mathematical Society, 1990, pp. 311-324.

[6] Gladkov L.A., Kurejchik V.V., Kurejchik V.M. Geneticheskie algoritmy - Genetic Algorithms, Moscow, 2010. (2-nd edition) (in Russian)

[7] Kurejchik V.V., Kurejchik V.M., Rodzin S.I. Teorija jevoljucionnyh vychislenij - Theory of evolutionary computation, Moscow, 2012 (in Russian).

[8] Coello, C.A., Christiansen, A.D., Aguirre, A.H. Use of Evolutionary Techniques to Automate the Design of Combinational Circuits. International Journal of Smart Engineering System Design, 2000.

[9] Tel'puhov D.V., Solov'ev R.A., Mjachikov M.V. Razrabotka prakticheskih metrik dlja ocenki metodov povyshenija sboeustojchivosti kombinacionnyh shem. Informacionnye tehnologii i matematicheskoe modelirovanie sistem, 2015, Trudy mezhdunarodnoj nauchno-tehnicheskoj konferencii, 2015, pp. 79-81 (in Russian).

[10] Stempkovskiy A.L., Telpukhov D.V., Solov'ev R.A., Mjachikov M.V. Povyshenie otkazoustojchivosti logicheskih shem s ispol'zovaniem nestandartnyh mazhoritarnyh jelementov. Informacionnye tehnologii, 2015, Vol. 21, No.10, pp. 749-756 (in Russian).