The Hardware Evolution Based on New ne-TCGA Algorithm

Jianwei Mi, Xiaoli Fang, and Libin Fan

***Abstract*—Aiming at typical shortcomings including large memory occupation of GA(Genetic Algorithm) in evolvable hardware and poor searching capabilities of CGA(Compact Genetic Algorithm) in solving complex problems, this paper presents an improved compact genetic algorithm named ne-TCGA(none-persistent elitism Compact Genetic Algorithm with Tendency). This algorithm is combined with the analysis of convergent trend on the basis of TCGA (Compact Genetic Algorithm with Tendency) and it adopts the strategy of temporary elitist preservation, which both ensure the adequate selection pressure, and maintains the diversity of the population in the evolutionary process. According to the experiment, it can be concluded that the ne-TCGA applied in evolvable hardware has better computational efficiency than other random search algorithms.**

***Index Terms*—evolvable hardware, genetic algorithm, ne-TCGA System control, temporary elitist preservation**

I. Introduction

WITH the development of fixed functional hardware and the reconfigurable hardware, the self-configurable and evolvable hardware will act the mainstream in near future. They are endowed to solve large-scale and complex problems by using the evolution patterns of biology [1]-[4], and can make the system automatically adjust internal structures in real time for adapting changes of internal and external environments [5]. Therefore, evolvable hardware will have a wide application prospect and considerable industrial, commercial value in circuit design [6], [7], fault-tolerant systems [8], pattern recognition [9], artificial intelligence [10], [11], and other fields. However, the configuration of logic circuits in evolvable hardware evolves according to the evolutionary algorithm, so the evolvable result and efficiency of the evolutionary algorithm to great extent decide whether the evolvable hardware is able to achieve the desired function or not. At present, the evolvable hardware technology is only able to accomplish simple small-scale circuits. So there still exist several problems in the process of its development, such as the low speed of evolution, the low efficiency of evolution, the weak robustness of evolvable circuit [12], [13]. In order to tackle these problems, it is of particularly importance to analyze the basic theoretical knowledge thoroughly and propose an algorithm with favorable effect of convergence and high efficiency of execution. Therefore, this paper elaborates an improved genetic algorithm with tendency named ne-TCGA (none-persistent elitism Compact Genetic Algorithm with Tendency), which improves the convergent efficiency of algorithm and achieves satisfactory performance of application in hardware evolution.

Manuscript received November 26, 2016; revised May 15, 2017. This work is partially supported by National Natural Science Foundation of China under Grant Nos. 51490660 and 51405362.

Jianwei Mi is a Associate Professor in Key Laboratory of Electronic Equipment Structure Design, Ministry of Education of China, Xidian University, Xi'an, Shaanxi, China(e-mail: jwmi@xidian.edu.cn).

Xiaoli Fang is a M.Sc. candidate at School of Mechanical and Electrical Engineering, Xidian University, Xi'an, Shaanxi, China(e-mail: xiaoli_fang@yeah.net).

Libin Fan is a M.Sc. candidate at School of Mechanical and Electrical Engineering, Xidian University, Xi'an, Shaanxi, China(e-mail: 15129563686@163.com).

II. The Fundamental Concept of ne-TCGA

The key technologies of evolvable hardware include programmable logic device and evolutionary algorithm. It is required that the corresponding devices can be configured repeatedly because of the randomness of evolutionary process and more evolutionary frequency. The dominant advantage of FPGA is programmable on-line. Therefore, FPGA currently becomes an ideal implementation device. GA (Genetic Algorithm) which is a random search algorithm for the simulation of natural evolution in biosphere by using a series of encoded bit string to describe the population of candidate solutions of the problem, is one of the most commonly used algorithms in evolutionary algorithms. However, because it needs to store large amounts of individual information in a population, the required storage space is proportional to $L \bullet N$ (where L represents the length of the chromosome bit string, N represents the population size). That is to say that the GA requires the storage unit of $O(N)$ order of magnitude, thus consuming a large amount of hardware resources and having huge calculation in dealing with complex problems [14]. At present, there are many ways and technologies to improve the application of GA in FPGA [12], such as the CGA (Compact Genetic Algorithm) [15]. The CGA uses the probability vector to describe the population. Its evolutionary process is that each generation can randomly produce two mutually independent chromosomes according to the probability vector and calculate fitness for them to update value of the probability vector. The evolution process terminates and makes the obtained probability vector as an optimal solution for problems until every bit of probability vector converges to "0" or "1." Therefore, CGA is very effective in application of limited memory, such as evolvable hardware [12].

A. The Advent of ne-TCGA

Although CGA is more successful than GA in terms of storage, and it has an explicit termination criterion, which is

when every bit of probability vector converges to "0" or "1," the evolution terminates. However, because the CGA achieves less information in the evaluation of chromosome [16] and has weak ability to search and insufficient ability to extract information of excellent individual, which straightforward lead to the loss of excellent individual and premature, and it can only be used to tackle certain simple problems of first order. The implementation results often fail to meet the actual application requirements for complex problems and the execution speed is low. Aiming at these shortcomings, the TCGA (Compact Genetic Algorithm with Tendency) is proposed afterward. TCGA includes extra judgment for trend from the current solution toward the optimal solution in the algorithm, and introduces the strategy of elitist preservation. But excessive elitist preservation may lead to premature convergence, and high selection pressure results in rapid convergence of the group. Thus it tends to fall into local optima at the expense of diversity of the population. So this paper proposes ne-TCGA, and introduces the temporary strategy of elitist preservation and parameters α which represents the maximum algebra of elitist preservation. The elite individuals can be inherited to the next generation within α-generations, but it will regenerate two chromosomes by the probability vector more than α-generations. The detailed process of ne-TCGA will be described as follows.

B. Detailed Process of ne-TCGA

1) Construct a probability vector P which has the same encoding length L as chromosome and let each bit of probability vector be equal to 0.5. Then generate two chromosomes randomly according to the probability vector. Each bit value of the probability vector represents the probability of the corresponding bit in the newly generated chromosome is equal to 1.

2) Calculate the fitness value of two generated chromosomes separately and compare these two fitness values. The chromosome with larger fitness value is treated as "winner." On the contrary, the chromosome with smaller fitness value is treated as "loser." Then compare each bit of the "winner" and "loser," if the two corresponding numbers are not equal to each other, then jump to step 3). Otherwise, continue to compare the next bit.

3) Invert the bit of the "winner," and then, judge and compare the fitness value between the inverted individual and the original individual. If the inverted fitness increases, one should judge the value of inverted bit. If its value is "1," update corresponding bit value of probability vector by adding the $1/N$ step length. If its value is "0," update corresponding bit value of probability vector by reducing the $1/N$ step length. The computational cost and storage in the algorithm is dependent of N, which is defined as the population size.

4) Determine whether it reaches the convergent conditions. If it reached, then terminate the process. If it did not, then proceed to the next step.

5) Perform the mutation operations on chromosomes. Judge whether each bit of probability vector is greater than 0.5, if it is greater than 0.5, then continue to determine the corresponding bit of the "winner" chromosome. If its value is "1," then remain constant. If its value is "0," then invert the bit; Otherwise, if the value of the corresponding bit of "winner" is "1," then invert the bit. If it is "0," then remain constant. Judge and compare the fitness value of generated chromosome with the fitness value of original "winner" chromosome, and make the chromosome with big fitness value as new "winner."

6) Judge the generation number of elitist preserving. If it is not more than α, it will generate a new chromosome by the probability vector and turn to step 2). If it is greater than α, it will generate two new chromosomes by the probability vector and turn to step 2).

The pseudo-code of ne-TCGA algorithm is as follows:

```
Step.1   for i = 1 to L do P[i]=0.5;
         a=generate(P); b=generate(P);
Step.2   if fitness(a)>fitness(b) then
           winner=a; loser=b;
         else winner=b; loser=a;
         fwn=fitness(winner);
Step.3   for i = 1 to L do
         if winner[i] != lose[i] then
           if winner[i]==1 then
           { winner[i]=0;fw=fitness(winner);
             if fw>fwn then P[i]=P[i]-1/N;
             else P[i]=P[i]+1/N;}
           else{ winner[i]=1;fw=fitness(winner);
                 if fw>fwn then P[i]= P[i]+1/N;
                 else P[i]=P[i]-1/N;}
Step.4   for i = 1 to L do
         if P[i]>0&&P[i]<1 then goto Step.5
         else end all
Step.5   c=mutate(winner);
         if fitness(c)>fitness(winner) then winner=c;
Step.6   if z<α a=winner; b=generate(P);z++;
         else a=generate(P); b=generate(P);z=0;
```

III. Comparison of Experimental Results of ne-TCGA, CGA, and TCGA

The ne-TCGA inherits the advantage of less storage space of CGA, and has real-time judgments for trend toward the optimal solution. It is therefore able to increase the search ability of algorithm and quickly make the chromosome individual converge to optimal solution. Then, in order to get the better chromosome individual with maximum probability, the improved mutation operation is introduced. Finally, by controlling the generation number of elitist preserving, we introduce the temporary elitist preserving strategy, so that it could ensure the adequate selection pressure, and maintain the diversity of the population because of the reasonable control of elitist preservation at the same time. It has a considerable advantage of successful control in premature convergence. This paper employs three algorithms, including CGA, TCGA, and ne-TCGA, to separately seek the maximums of the following two functions and averages values in each population size of every algorithm after running 10 times. The curves of these functions are shown in Fig. 1 and Fig. 2 respectively. The results of comparison on performance and required numbers of evolution are shown in Fig. 3, Fig. 4, Fig. 5, and Fig. 6.

$$y=\frac{0.6-\sin\left(x^2\right)}{\left(3+0.002x^2\right)^2}+1\ ,\quad -100\le x\le 100 \qquad (1)$$

$$z=3(1-x)^2\,e^{-x^2-(y+1)^2}-10\left(\frac{x}{5}-x^3-y^5\right)e^{-x^2-y^2}\ , \qquad (2)$$

$$-3\le x\le 3\ ,\quad -3\le y\le 3$$

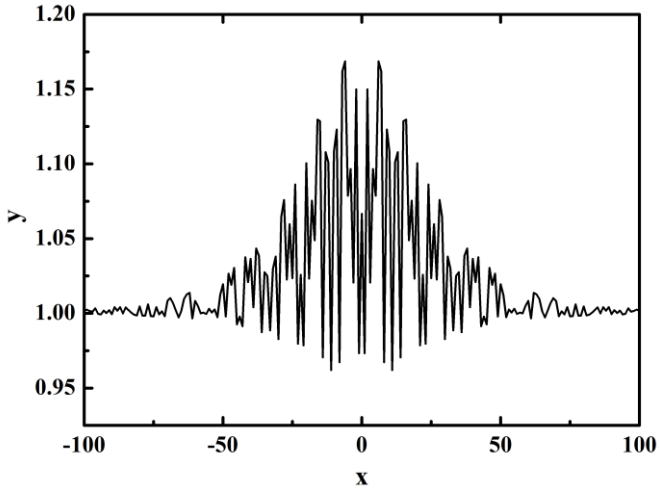


Fig. 1. The curve of function（1）

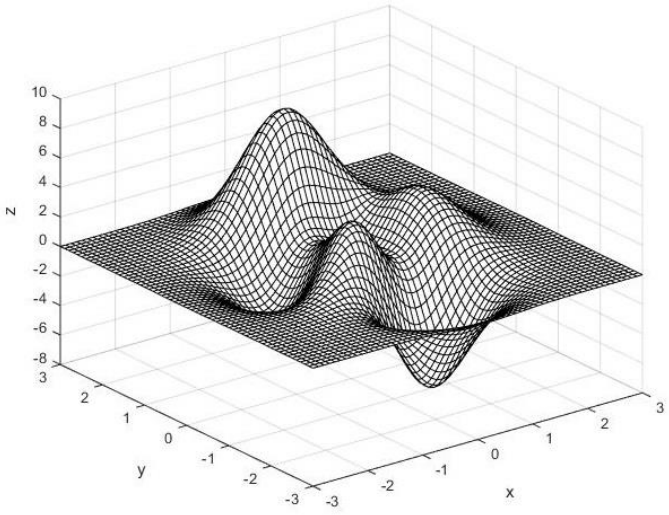


Fig. 2. The curve of function（2）

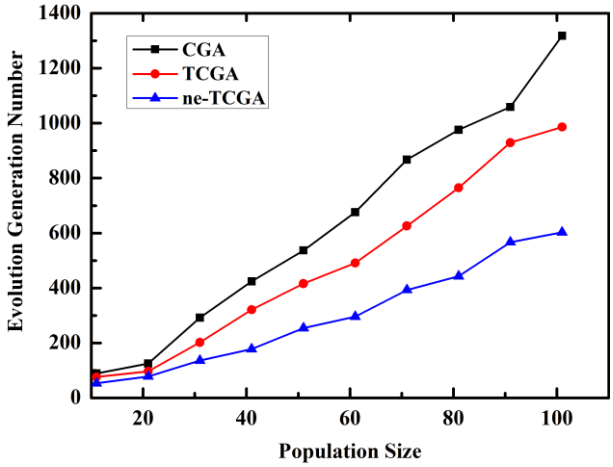


Fig. 3. The evolution generation number of function (1)

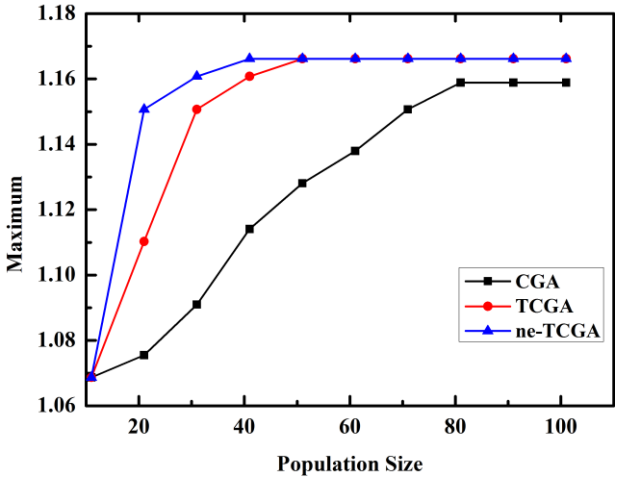


Fig. 4. The maximum value of function (1)

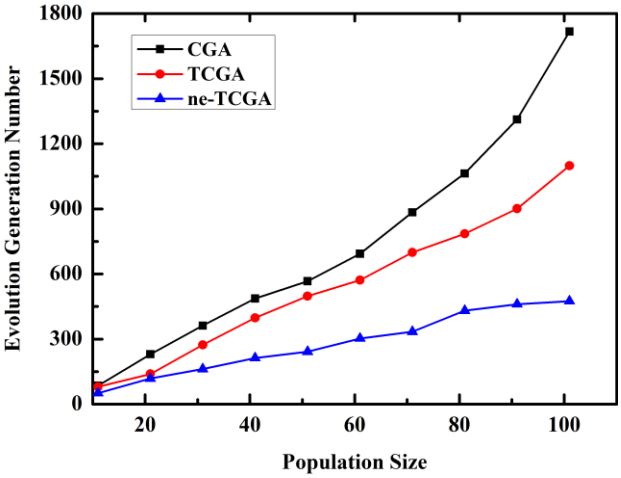


Fig. 5. The evolution generation number of function (2)

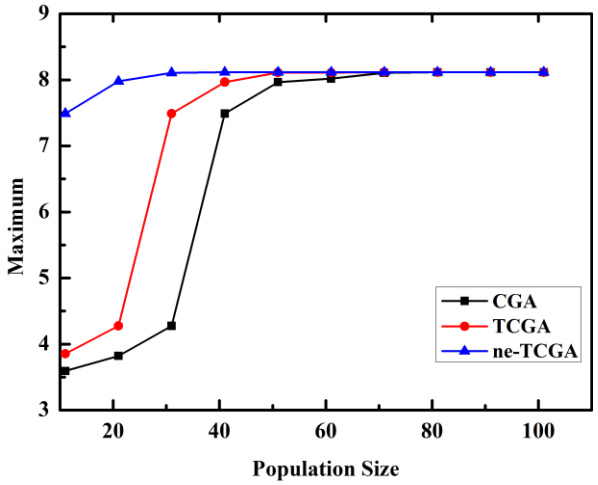


Fig. 6. The maximum value of function (2)

As shown in these figures, the numbers of evolution of the three algorithms are increasing when the population size increases. But the ne-TCGA algorithm has the minimum number of evolution and the best performance of solution in the case of the same population size. Its evolutionary generation number reduces by 30%-40% than TCGA. Besides, it has the highest solving speed when the elitist generation α is 50% of the total population size. When seeking the maximum of function (1), although CGA once ever achieved the maximum in domain of definition in the

evolutionary process, the results always converged to local optima and failed to get the maximum. Moreover, the final results obtained from TCGA are not always the global optimal solution. So we can conclude that the ne-TCGA algorithm has more advantages over CGA and TCGA algorithms in problem of seeking the maximum of functions.

IV. The Application of ne-TCGA in the Evolvable Hardware

The evolvable hardware is conducted by encoding structures and parameters of circuit into chromosomes and executing evolutionary operation. The classical genetic algorithm occupies large memory and has large calculations for extensive range searching. Therefore, in order to reduce the storage and improve the efficiency of searching, we develop the ne-TCGA algorithm proposed above. This paper uses ne-TCGA, TCGA, and CGA algorithms and regards a full adder as the evolutionary target. We chose the FPGA development board which is EP2S30F484I4 in Altera's Stratix II family to design the self-evolutionary system, and compare the results in verification of the three algorithms. At present, the implementation of evolvable hardware based on FPGA can be divided into the evolution between boards, the evolution of inter-chip on board-level and the evolution on chip-level. And the evolution on chip-level which saves the data communication time and is also the development trend of evolutionary system, can implement the evolutionary process on a single chip. So we select this method in this paper, and its implementation procedure of hardware platform is shown in Fig. 7.

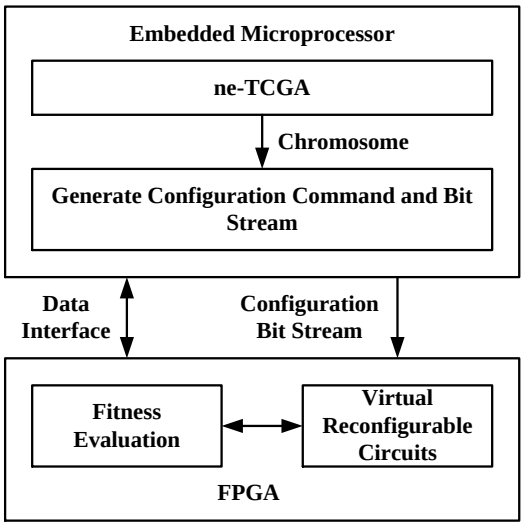


Fig. 7. The implementation of evolvable hardware platform

A. Hardware Design of Self-evolutionary System

The newly designed self-evolutionary system contains the virtual reconfigurable circuit-based IP core. The evolvable IP core is actually an evolutionary circuit which is controlled by evolutionary algorithm and is able to operate on the Nios II soft-core. The IP core designed in this paper is a cell array consisting of 40 cells [17], as shown in Fig. 8. The cell array has 8 external inputs and 8 outputs.

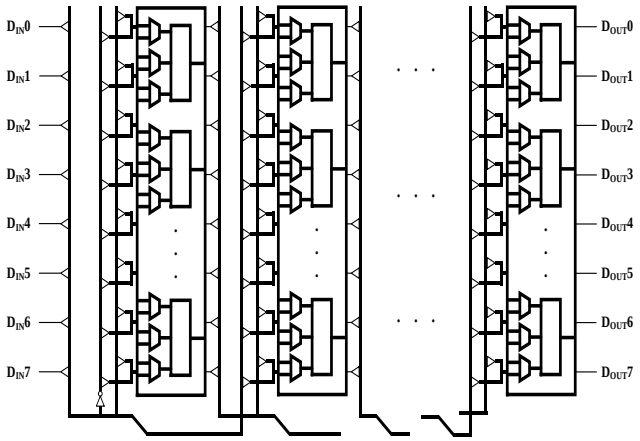


Fig. 8. Schematic diagram of the cell array

The basic logic configuration cell designed in this paper is shown in Fig. 9, which is composed of the look-up table (LUT) and multiplexer. The LUT with 3 inputs is selected because the evolvable circuit model of 3-input LUT has the top efficiency [18]. Although the cell array has 8-bit external input, the inputs of 8 cells in the first column are made up of the external 8 inputs and the inversion of the same 8 inputs. The inputs of 8 cells in the second column are constituted by the external 8 inputs and 8 outputs of cells in the first column. The inputs of cells in the third column and the subsequent columns are made up of the outputs of the immediately previous two columns. So every cell unit has 3 selectors of 16-to-1 and the output of selector is further considered as control bit to determine the input of look-up table (LUT) behind. Because each selector of 16-to-1 is configured by 4 chromosomes and LUT needs 8 chromosomes, it requires 20 chromosomes totally to configure a single cell. Since cell array contains 40 cell units, it thus needs 800 chromosomes to configure a cell array.

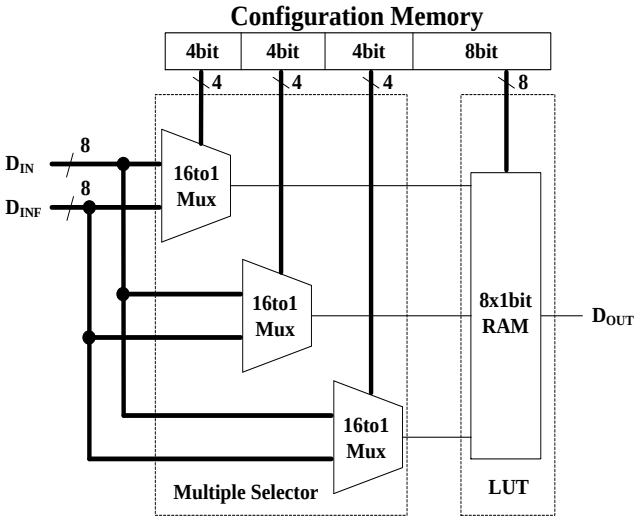


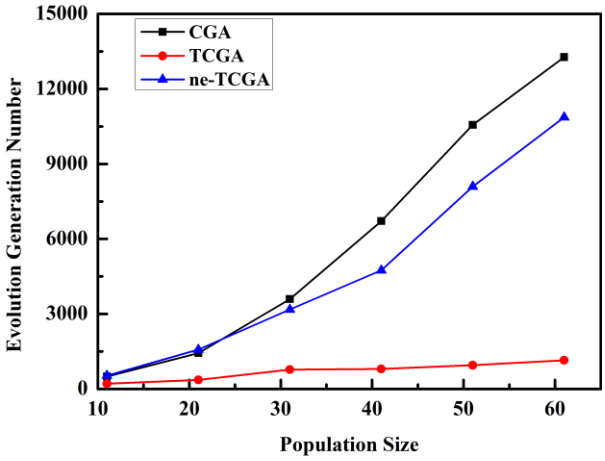
Fig. 9. Schematic diagram of internal structure in cell unit

The evolvable IP core has 8 external inputs and 8 outputs, but a full adder only has 3 inputs and 2 outputs. So we just take 8 combinations of low three places in 8 external inputs as the inputs of full adder in this paper. This operation will be achieved in the Nios II IDE by C language. After inputting the test vectors, it operates the outputs of cell array "XNOR" the true table of 8 combinations and achieves the matched signals.

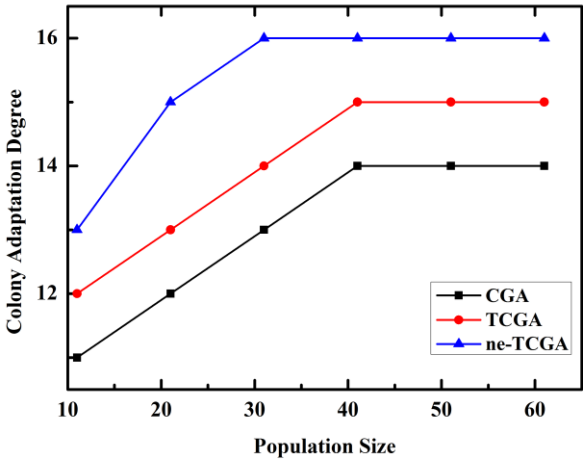
Furthermore, it designs a MASK register used to shield the higher 6 bits of matched signals, because the full adder only has 2 outputs. It puts the 0x3 into MASK register and operates it "AND" matched signals. Then it outputs match-data and the number of "1" in match-data is the fitness value. The maximum of fitness value is 16, which represents an evolved correctly full adder.

B. The Verification of Self-evolutionary System

The ne-TCGA, TCGA, and CGA algorithms used in this paper are respectively programmed using C language in Nios II soft-core. Then the program is downloaded to the development board by the JTAG download cable to evolve a full adder. The evolution results are shown in Fig. 10 and the average evolution results of 10 runs in each population size of every algorithm are shown in Table I. It can be seen from the results that the TCGA and CGA algorithms both fall into local optima, and only the ne-TCGA algorithm successfully evolves a full adder in limited evolving generation number.



(a)The evolution generation number



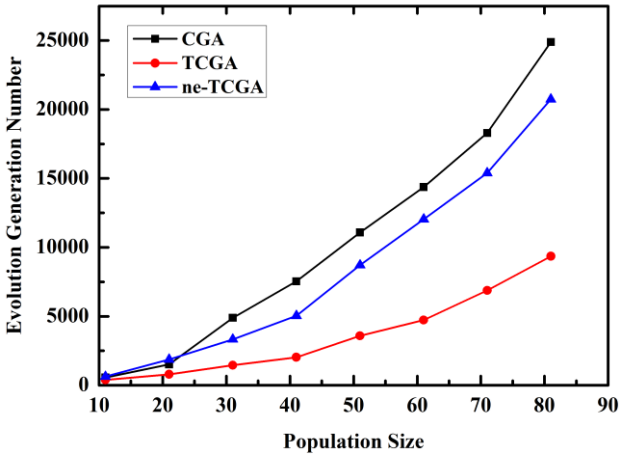
(b)The colony adaptation degree

Fig. 10. The evolution results of 1-bit full adder

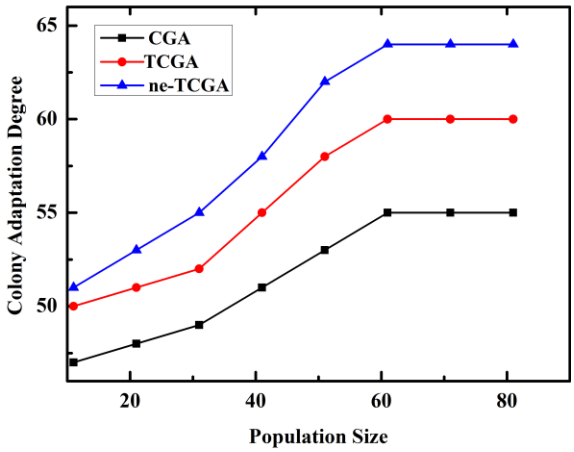
TABLE I
THE AVERAGE EVOLUTION RESULTS OF 1-BIT FULL ADDER

Population Size		11	21	31	41	51	61
Evolution Generation Number	CGA	498	1437	3596	6716	10563	13272
	TCGA	211	362	775	800	951	1146
	ne-TCGA	536	1574	3178	4746	8093	10867
Colony Adaptation Degree	CGA	11	12	13	14	14	14
	TCGA	12	13	14	15	15	15
	ne-TCGA	13	15	16	16	16	16

The following is an example of the evolution of 2-bit multiplier circuits to further validate the application of ne-TCGA, TCGA and CGA algorithms in self-evolutionary system. The designs of 2-bit multiplier and 1-bit full adder are similar. Because 2-bit multiplier has 4 inputs and 4 outputs, we just take 16 combinations of low 4 places in 8 external inputs of the evolvable IP core as the inputs of 2-bit multiplier, then we operate the outputs of cell array "XNOR" the true table to achieve the matched signals, and put the 0xF into MASK register used to shield the higher 4 bits of matched signals. The maximum of fitness value is 64. The evolution results of evolving 2-bit multiplier using the ne-TCGA, TCGA, and CGA algorithms are shown in Fig. 11 and the average evolution results of 10 runs in each population size of every algorithm are shown in Table II.



(a)The evolution generation number



(b)The colony adaptation degree

Fig. 11. The evolution results of 2-bit multiplier

TABLE II
THE AVERAGE EVOLUTION RESULTS OF 2-BIT MULTIPLIER

Population Size		11	21	31	41	51	61
Evolution Generation Number	CGA	560	1510	4890	7535	11077	14375
	TCGA	389	795	1456	2033	3589	4728
	ne-TCGA	630	1868	3329	5040	8711	12040
Colony Adaptation Degree	CGA	47	48	49	51	53	55
	TCGA	50	51	52	55	58	60
	ne-TCGA	51	53	55	58	62	64

As seen from Fig. 10, Fig. 11, Table I, and Table II, the evolution results of evolving 1-bit full adder and 2-bit multiplier using the ne-TCGA, TCGA, and CGA algorithms are similar. CGA has the lowest evolution efficiency relative

to the other two algorithms. Although TCGA has the least evolving algebra, it fails to evolve a required circuit. The ne-TCGA algorithm can successfully evolve 1-bit full adder and 2-bit multiplier in limited evolving generation number, which shows that the ne-TCGA algorithm is efficient in applications of hardware evolution.

V. Conclusion

This paper proposes the ne-TCGA algorithm which adds the temporary elitist preserving strategy on the basis of TCGA algorithm, thereby in the case of inheriting advantages of TCGA algorithm, such as the less storage space occupied, the definite condition of convergence termination, the strong searching capabilities. In addition, the algorithm also provides some new features:

1) It reduces the possibility of falling into local optimal solution and displays better convergence and searching capability than TCGA. At the same time, its running results are more accurate and the evolutionary generation number reduces by 30%-40% than TCGA.

2) By applying the ne-TCGA, TCGA, and CGA algorithms to the self-evolutionary system in order to evolve a full adder and 2-bit multiplier, by comparison one can conclude that the ne-TCGA algorithm in circuit evolution is effective, and its occupied resources are greatly reduced.

The ne-TCGA algorithm will have considerable potential in hardware evolution by its simple implementation methods and advantage of less storage space occupied in hardware.

Acknowledgment

The authors gratefully acknowledge the helpful comments and suggestions of the reviewers, who have improved the presentation.

References

[1] A. N. Al-Rabadi and M. A. Barghash, “Fuzzy-PID Control via Genetic Algorithm-Based Settings for the Intelligent DC-to-DC Step-Down Buck Regulation,” Engineering Letters, vol.20, no.2, pp. 176-195, 2012.

[2] X. Tian, K. Benkrid, and X. Gu, “High Performance Monte-Carlo Based Option Pricing on FPGAs,” Engineering Letters, vol.16, no.3, pp. 434-442, 2008.

[3] G. Mohammed, J. El-Miloud, and M. El-Bekkaye, “A Genetic Algorithm Approach for an Equitable Treatment of Objective Functions in Multi-objective Optimization Problems,” IAENG International Journal of Computer Science, vol.41, no.2, pp.102-111, 2014.

[4] J. Lohn and G. Hornby, “Evolvable hardware using evolutionary computation to design and optimize hardware systems,” IEEE Computational Intelligence Magazine, vol.1, no.1, pp.19-27, 2006.

[5] T. A. El-Mihoub, A. A. Hopgood, L. Nolle, A. Battersby, “Hybrid Genetic Algorithms: A Review,” Engineering Letters, vol.13, no.2, pp. 124-137, 2006.

[6] V. Parthasarathy, P. Anandakumar, V. Rajamani, “Design, Simulation and FPGA Implementation of a Novel Router for Bulk Flow TCP in Optical IP Networks,” IAENG International Journal of Computer Science, vol.38, no.4, pp.343-349, 2011.

[7] H. Liu, J. Miller, and A. Tyrrel, “Intrinsic evolvable hardware implementation of a robust biological development model for digital systems,” Proc. of 2005 NASA/DoD Conference on Evolvable Hardware, Washington DC, USA, pp.87-92, 2005.

[8] R. J. Terrile, H. Aghazarian, M. I. Ferguson, W. Fink, “Evolutionary computation technologies for the automated design of space systems,” Proc. of the 2005 NASA/DoD Conference of Evolution Hardware, Washington DC,USA, pp. 131-138,2005.

[9] J. Wang, B. B. Tang, C. H. Piao, G. H. Lei, “Statistical method-based evolvable character recognition system,” Proc. of the IEEE International Symposium Industrial Electronics, Seoul, Korea, pp.804-808, 2009.

[10] H. D. Garis, M. Korkin, F. Gers, M. Hough, “ATR’s artificial brain (CAM-brain) project: a sample of what individual CoDi-1 Bit model evolved neural net modules can do,” Proc. of the IEEE Congress on Evolutionary Computation (CEC), Washington DC, USA, pp.1979-1987, 1999.

[11] N. Feamster, L. gao, and J. Rexford, “How to lease the Internet in your spare time,” Acm Sigcomm Computer Communications Review, vol.37, no.1, pp.61-64, 2007.

[12] P. R. Fernando, S. Katkoori, and D. Keymeulen, “Customizable FPGA IP core implementation of a general-purpose genetic algorithm engine,” IEEE Trans. on Evol. Comput., vol.14, no.1, pp.133-149, 2010.

[13] J. Yutana and C. Prabhas, “A parallel genetic algorithm for adaptive hardware and its application to ECG signal classification,” Neural Comput & Applic, vol. 22, no.7, pp.1609-1626, 2013,

[14] M. A. Moreno-Armendáriz, N. Cruz-Cortés, C. A. Duchanoy, A. León-Javier, R. Quintero, “Hardware implementation of the elitist compact Genetic Algorithm using Cellular Automata pseudo-random number generator,” Computers and Electrical Engineering, vol.39, no.4, pp.1367-1379, 2013.

[15] G. R. Harik, “The compact genetic algorithm,” IEEE Trans. on Evol. Comput., vol.3, no.4, pp.287-297, 1999.

[16] F. Cupertino, E. Mininno, and D. Naso, “Compact genetic algorithms for the optimization of induction motor cascaded control,” Electric machines & drives conference, Antalya, Turkey, pp.82-87, 2007.

[17] J. L. Liu and R. Yao, “The implementation of self-evolutionary hardware,” Journal of Jiamusi University, vol.30, no.2, pp.109-212, 2012.

[18] H. X. Bu, L. G. Chen, and J. M. Lai, “A LUT-based VRC model for random logic function evolution,” Proc. of IEEE International Conference on ASIC, Changsha, China, pp.1-5, 2009.