Combining Multiple Evolved Analog Circuits for Robust Evolvable Hardware

Kyung-Joong Kim[1] and Sung-Bae Cho[2]

[1] Department of Computer Engineering, Sejong University, Seoul, South Korea
kimkj@sejong.ac.kr
[2] Department of Computer Science, Yonsei University, Seoul, South Korea
sbcho@cs.yonsei.ac.kr

Abstract. Redundancy is one of the most important concepts when it comes to designing fault-tolerant systems. For example, if component failures occur, other redundant components can replace the functions of broken parts and the system can still work. The failure of electronic hardware presents a critical threat to the completion of modern aircraft, spacecraft, and robot missions. Compared to digital circuits, designing analog circuits is a difficult and knowledge-intensive task. In this paper, we used evolutionary computation to generate multiple analog circuits automatically and then we combined the solutions to generate robust outputs. Because evolutionary computation refers to a population-based search, multiple, redundant solutions can be maintained. Experimental results on the evolution of the lowpass filter show that the combination of multiple evolved analog circuits produces results that are more robust than those of the best single circuit.

1 Introduction

The presence of robustness is vital when working with analog circuits because there is usually a possibility of component failure. Physical damage, manufacturing faults, aging, radiation, temperature changes and power surges are possible reasons for such failures. If the system is not fault-tolerant, there is a high possibility of radical performance degradation. The purpose of fault-tolerant systems is to maintain functioning even when these kinds of component failures are experienced. Redundancy is the key concept of these systems because, in general, redundant parts can replace original damaged parts. Since it is necessary to prepare redundant parts with different architectures before system manufacturing, it is not common for original parts and redundant parts to fail simultaneously.

Designing analog circuits is a difficult, knowledge-oriented process which is not possible without training or experience. Furthermore, it is also difficult for novice users to design redundant circuits with the same functions but with different architectures. Koza *et al*. attempted to evolve analog circuits using genetic programming (GP) based on minimal information about problems such as the number of inputs and outputs [1]. Although these researchers showed the possibility of automatic circuit synthesis using evolutionary computation, their work did not deal with fault-tolerance issues.

E. Corchado and H. Yin (Eds.): IDEAL 2009, LNCS 5788, pp. 359–367, 2009.

Recently, Hu *et al*. evolved fault-tolerant filters using genetic programming (GP). These filters were robust to the changing parameters of each component [2]. Hollinger *et al*. also evolved fault-tolerant circuits for controlling robots using a genetic algorithm (GA) [3]. These researchers focused on using the most fault-tolerant individual in their evolutionary computation and ignored other redundant candidates in the population.

In this paper, we propose a method of exploiting multiple solutions from evolutionary computation in order to create robustness of analog circuits. The basic idea is to combine different, multiple analog circuits with the same function, using weighted summing circuits (averaging the outputs of multiple circuits), having an output which is the average of the outputs of each circuit. Because of these redundant circuits, a fault in one circuit can be recovered from the other circuits' normal outputs. In this method, the key point is to prepare diverse circuits with different architectures. If the members of the combined circuits are identical, they can suffer from similar failures and the system cannot realize the benefits of synergism. The easiest way to maintain diversity in the population is to use tournament-based selection method. Following the recent work [4], we have used evolutionary computation to evolve a lowpass filter, and multiple circuits were combined for robustness. The fault-tolerance levels of both evolved and combined circuits were also tested by removing each component.

2 Related Works

Koza *et al*. evolved eight different analog circuits including a lowpass filter, a crossover (woofer and tweeter) filter, a source identification filter, an amplifier, a computational circuit, a time-optimal controller circuit, a temperature-sensing circuit, and a voltage reference circuit [1]. These researchers used GP with automatically defined functions. Each function of the trees in GP encodes the operations that are applied to the embryonic circuits. These researchers also used parallel GP with 64 80-MHz PowerPC processors and each circuit was evaluated using a modified SPICE simulator.

There are implicit fault-tolerance and explicit fault-tolerance levels that exist in evolvable hardware [5][6]. With implicit fault-tolerance levels, it is assumed that the faults are previously known. At first, the evolution process continues without fault until successful candidates emerge and they are tested with the fault. The worst fault is selected to be incorporated in the fitness evaluation of the next generation. Because of the "worst" fault, the fitness degrades. It continues evolution until the successful candidates emerge again. And then, the faults are applied to these candidates and the worst one is selected again. This process is repeated.

Thompson *et al*. defined population fault tolerance as the potential for a population of an evolved circuit to contain an individual who can adequately perform a task in the presence of a fault that renders the previously-best individual useless [7]. In explicit fault-tolerance, given a circuit with n components, the circuit has to be simulated n times by removing each component. This requires high computational costs because the simulation process has to be repeated n times [2][3][6]. Keymeulen *et al*. compared both approaches and concluded that the implicit technique based on population dynamics outperformed the explicit one [8].

Hu *et al.* evolved fault-tolerant lowpass and highpass filters using GP when considering robustness [2]. They compared evolution-without-robustness, evolution-with-robustness and fine-tuning using a GA and solutions from the evolution-without-robustness process. The evolution-without-robustness process is just a GP version for evolving filters. The evolution-with-robustness process refers to explicit fault-tolerance and it uses an average performance of multiple simulations by changing the parameters of each component. Fine-tuning refers to using the architecture of an evolved circuit without considering robustness. The parameters of evolved circuits can be reconfigured using GA with the explicit fault tolerance scheme. These researchers considered the change of parameters as faults. They reported that the evolution-with-robustness process outperformed the other two processes.

In summary, if the faults are previously known, the implicit fault-tolerance level is better than the explicit level. However, the explicit level requires high computation costs because of the need for multiple runs. All of these processes generally consider only a single circuit for robustness and they focus the endowment of the fault-tolerance into the circuit. In [9], fault-tolerant designs with redundant structures were classified as 'static redundancy,' 'hot standby' and 'cold standby.' In 'static redundancy,' three or more active parallel modules with the same input signals were used. Their outputs were connected to a voter, who compared the signals and decided by majority vote which signal value was the correct one. The other two designs required a fault detection module and they were difficult to perform without experience. In this paper, we used multiple redundant circuits and a weighted summing circuit in order to combine the outputs. The weighted summing circuit was a relatively simple one and it was possible to adjust the weight by changing the resistors.

3 Proposed Method

The basic idea of the proposed method involved using multiple redundant circuits with the summing circuit. Each circuit performed the same function and all were active. The circuits were all evolved using evolutionary computation and it was assumed that they all had different architectures. Figure 1 shows an overview of the proposed method. Important issues included the number of evolved circuits for the ensemble system, the way of evolving each member circuit, and the implementation of the summing circuit. In this paper, we selected the best n circuits from the last generation (in this work, n is the half of the population size).

Among all possible ensembles of the best n circuits, the one with the highest fault-tolerance was selected. The summing circuit was implemented by using an op amp component. In this paper, there are 20 evolved circuits in the last generation of the evolution. Based on their fitness, 10 circuits are chosen. If we choose 3 circuits from them for the ensemble, there are ${}_{10}C_3$ (120) possibilities. The best ensemble was chosen from the exhaustive search of the 120 possibilities based on the fault-tolerance level of each ensemble.

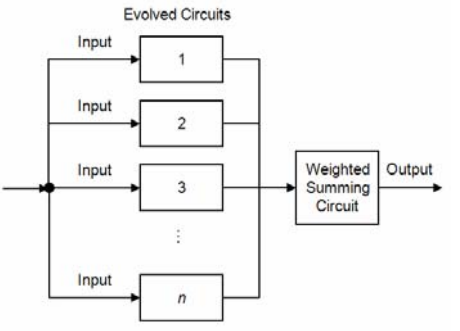

Fig. 1. Overview of the proposed method

3.1 Evolving Analog Circuits

Step 1) Initialization: The embryonic circuit refers to the starting point of the circuit development and it has voltage source and fixed resistors. Each modifiable wire is replaced with one new component.

Step 2) Mutations (new *P* offspring): One component is randomly selected and one of eight different mutations is applied to the component. The mutations are parameter change, type change, parallel addition of a different type component, serial addition of a different type component, component deletion, ground setting, replacement, and adding a component. *P* is a population size. Mutations are as follows [4].

- Parameter change: The component's value is assigned as a new randomly chosen value.
- Type change: The component type is swapped to a different one randomly.
- Parallel addition of a different type component: A new component (with a different type) is added in parallel configuration to the component. The type and value of the new component is randomly chosen.
- Serial addition of a different type component: Same as above except the addition in serial configuration.
- Component deletion: The component is removed from the circuit.
- Ground setting: The component is connected to the ground.
- Replacement: The component is replaced with a new component (possibly of the same type).
- Adding a component: A new component bridges between two randomly chosen wires (not identical wire).

Step 3) Circuit simplification: It combines identical components in a serial or parallel configuration into a single component. It maintains the circuit size as small as possible.

Step 4) Fitness Evaluation with a Spice Simulator

Step 5) Selection (P individuals from 2*P* pool): The best *P* circuits are selected from 2*P* individuals (parents + offspring).

Step 6) Termination: It stops when the number of generation is larger than the maximum number of generation.

Step 7) Ensemble: It searches for the best fault-tolerant ensembles of circuits from the last generation. Among the best n circuits, we choose 3 circuits for an ensemble and there are ${}_nC_3$ possibilities. The exhaustive search was used to find the best fault-tolerant ensemble from them.

3.2 Combination

The circuits were combined using weighted summing circuits. Figure 2 shows a weighted summing circuit [11]. Input voltages were defined as v_1, v_2, …, v_n. The output voltage v_o was defined as follows.

$$v_o = -\left(\frac{R_f}{R_1}v_1 + \frac{R_f}{R_2}v_2 + \cdots + \frac{R_f}{R_n}v_n\right) \tag{1}$$

The weights of each input voltage were adjusted using the resistors. If the resistor for each input voltage was the same as R_f, the weight was 1. If the weight was 1 for all input voltages, it was the average of the input voltages. In this work, the average method was used.

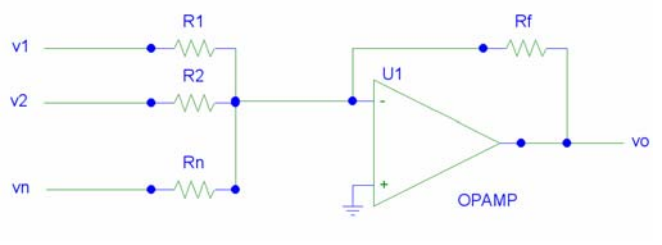

Fig. 2. A weighted summing circuit

3.3 Lowpass Filter Evolution

The initial circuits were generated randomly and their structures were dependent on the architecture of the embryonic circuit. In this paper, the embryonic circuit contained two modifiable wires and fitness was evaluated using a SPICE simulator. The developed circuit was converted into NETLIST, an input file for the SPICE program.

The target of evolution was a lowpass filter with a one-input, one-output circuit composed of capacitors and inductors that passed all frequencies below 1 kHz and suppressed all frequencies above 2 kHz. Figure 3 shows an embryonic circuit for the lowpass filter. In the diagram, VSOURCE, RSOURCE, and RLOAD are fixed, the Z0 and Z1 wires are modifiable.

The voltage source was 2V, and RSOURCE and RLOAD were 1 kΩ. In the lowpass filter problem, the frequency area below 1 kHz was a passband and the frequency area above 2 kHz was a stopband. The “don’t care” band was between 1 kHz and 2 kHz. Any voltage levels lower than 970 mV in the passband, and any voltage levels above 1 mV in the stopband, were regarded as unacceptable. From 1Hz to 100 kHz,

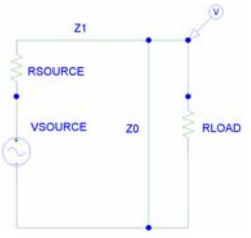

Fig. 3. An embryonic circuit

the SPICE simulator performed an AC small signal analysis. The frequency area was divided into five decades and each decade was further divided into 20 parts (on a logarithmic scale). Finally, the simulator checked the voltage of 101 points (61 points below 1 kHz, 5 points between 1 kHz and 2 kHz, and 35 points above 2 kHz). The fitness was calculated as follows.

$$Fitness = \frac{1}{Error} \qquad Error = \sum_{i=0}^{100} W(d(f_i), f_i) \times d(f_i) \tag{2}$$

The fitness value was summed over 101 points. In the above equation, f_i represents the frequency of the ith point, d represents the difference between the target and observed values at the frequency f_i, and W represents the weight for the difference at the frequency f_i (based on [1]). If circuits could not be simulated in the SPICE program, the fitness of these circuits was 0.

4 Experimental Results

WinSpice (version 1.05.07) by OuseTech [10] was used and interfaced using file I/O functions in C++. The population size was 20 and the maximum generation is 300. Maximum node number is 10. We repeated 5 runs.

Figure 4 shows the error of the best individual at each generation. Figure 5 shows the circuit diagram of the best circuit and its output response. The circuit has eight components. We selected the best 10 circuits from the last generation. The diversity

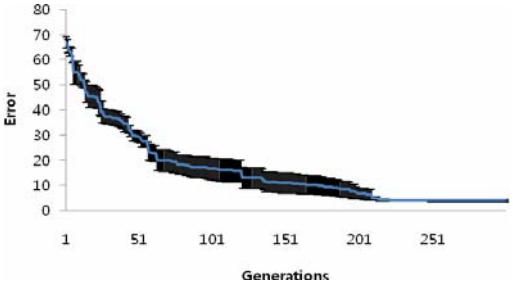

Fig. 4. The error of the best circuits over generation (averaged over 5 runs, standard error bar was included)

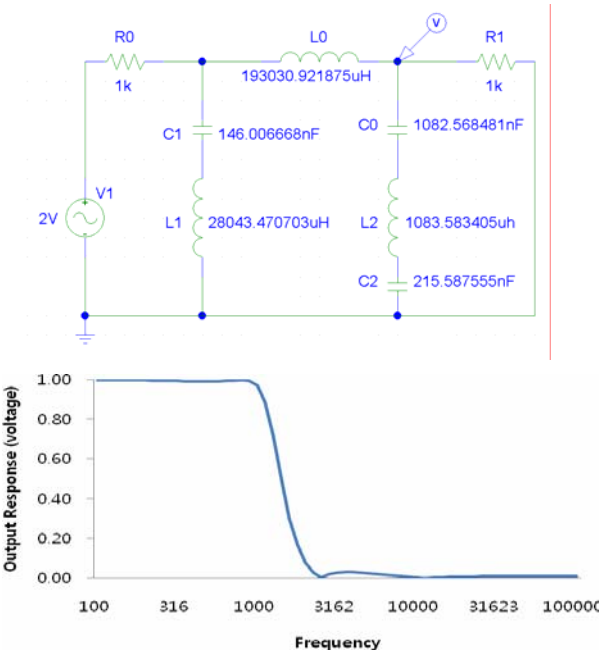

Fig. 5. The circuit diagram and output response of the best circuit

Table 1. Summary of statistics (average of 5 runs, standard error)

	Error (No damage)			*Fault-Tolerance*		
	Fault-Tolerance Evolution	Single Circuit	Ensemble of three circuits	Fault-Tolerance Evolution	Single Circuit	Ensemble of three circuits
Best	344	4.016 ± 0.297	4.017 ± 0.297	0.16	1.19 ± 0.08	3.11 ± 0.23
Avg	652 ± 94	4.022 ± 0.296	4.023 ± 0.296	0.16 ± 0.0	1.08 ± 0.09	2.93 ± 0.22

(the number of unique circuits) of the 10 circuits is 8.2 ± 1.11 (std. error). The average number of components of the 10 circuits is 10.34 ± 0.24 (std. error).

The fault-tolerance level of each circuit was represented by *ft*. The number of components (except the fixed resistors and voltage source) of the circuit was defined as *M*. The error of the circuit by removing the *i*th component was defined as *error*(*i*). The result of removing a component is an open circuit.

$$ft = \frac{1}{\frac{1}{M}\sum_{i=1}^{M} error(i)} \times 100 \tag{3}$$

Table 1 summarizes the error and fault-tolerance of both approaches. The ensemble showed an acceptable fitness performance and the best for the fault-tolerance

measure. The fault-tolerance level showed significant improvement compared to the single circuit. With regard to the fault-tolerance level, every ensemble with three circuits performed better than the single circuit. In the fault-tolerance evolution, the fault-tolerance level of each circuit was used as a fitness function.

5 Conclusions and Future Works

In this work, we have proposed an ensemble of evolved circuits with a weighted summing circuit. It was shown that the combination of the best circuits performed well when removing components. Furthermore, the ensemble also performed quite well when there was no component failure. In general, this will allow designers to make fault-tolerant redundant circuits. For diversity, tournament selection was used. However, it is more convenient and natural to use speciation and niching methods for evolution. For this purpose, in future work we will devise a method for measuring the similarities between two circuits in the phenotype and genotype levels. Also, the weights of each redundant circuit can be adjusted according to characteristic of the circuit.

Acknowledgements

This work was supported by Prof. Hod Lipson (Cornell University). Co-author Kyung-Joong Kim was supported by Korea Health 21 R&D Project, Ministry for Health, Welfare and Family Affairs (A040163).

References

1. Koza, J.R., Bennett III, F.H., Andre, D., Keane, M.A., Dunlap, F.: Automated Synthesis of Analog Electrical Circuits by Means of Genetic Programming. IEEE Trans. on Evol. Comp. 1(2), 109–128 (1997)
2. Hu, J., Zhong, X., Goodman, E.D.: Open-ended Robust Design of Analog Filters using Genetic Programming. In: Proceedings of the 2005 Conference on Genetic and Evolutionary Computation, vol. 2, pp. 1619–1626 (2005)
3. Hollinger, G.A., Gwaltney, D.A.: Evolutionary Design of Fault-tolerant Analog Control for a Piezoelectric Pipe-crawling Robot. In: Proceedings of the 2006 Conference on Genetic and Evolutionary Computation, pp. 761–768 (2006)
4. Kim, K.J., Wong, A., Lipson, H.: Automated Synthesis of Resilient and Tamper-evident Analog Circuits without a Single Point of Failure. Genetic Programming and Evolvable Machines (2009), doi:10.1007/s10710-009-9085-2
5. Zebulum, R.S., Pacheco, M.A.C., Vellasco, M.M.B.R.: Evolutionary Electronics: Automatic Design of Electronic Circuits and Systems by Genetic Algorithms. CRC Press, Boca Raton (2002)
6. Thompson, A.: Evolutionary Techniques for Fault Tolerance. In: Proceedings of UKACC International Conference on Control, pp. 693–688 (1996)

7. Layzell, P., Thompson, A.: Understanding Inherent Qualities of Evolved Circuits: Evolutionary History as a Predictor of Fault Tolerance. In: Miller, J.F., Thompson, A., Thompson, P., Fogarty, T.C. (eds.) ICES 2000. LNCS, vol. 1801, pp. 133–144. Springer, Heidelberg (2000)
8. Keymeulen, D., Zebulum, R.S., Jin, Y., Stoica, A.: Fault-tolerant Evolvable Hardware using Field-programmable Transistor Arrays. IEEE Transactions on Reliability 49(3), 305–316 (2000)
9. Isermann, R.: Fault-Diagnosis Systems: An Introduction from Fault Detection to Fault Tolerance. Springer, Heidelberg (2006)
10. WinSpice, `http://www.winspice.com/`
11. Sedra, A.S., Smith, K.C.: Microelectronic Circuits, 5th edn. Oxford University Press, Oxford (2004)