Toward Self-adaptive Embedded Systems: Multi-objective Hardware Evolution

Paul Kaufmann and Marco Platzner

University of Paderborn

Abstract. Evolutionary hardware design reveals the potential to provide autonomous systems with self-adaptation properties. We first outline an architectural concept for an intrinsically evolvable embedded system that adapts to slow changes in the environment by simulated evolution, and to rapid changes in available resources by switching to preevolved alternative circuits. In the main part of the paper, we treat evolutionary circuit design as a multi-objective optimization problem and compare two multi-objective optimizers with a reference genetic algorithm. In our experiments, the best results were achieved with TSPEA2, an optimizer that prefers a single objective while trying to maintain diversity.

1 Introduction

In the last decades, *natural computing* methods which take problem solving principles from nature have gained popularity. Among others, natural computing includes evolutionary computing. Evolutionary computing covers population-based, stochastic search algorithms inspired by principles from evolution theory. An evolutionary algorithm tries to solve a problem by keeping a set (population) of candidate solutions (individuals) in parallel and improving the quality (fitness) of the individuals over a number of iterations (generations). To form a new generation, genetically-inspired operators such as crossover and mutation are applied to the individuals. A fitness-based selection process steers the population towards better candidates.

Evolvable hardware denotes the combination of evolutionary algorithms with reconfigurable hardware technology to construct self-adaptive and self-optimizing hardware systems. The term evolvable hardware was coined by de Garis [1] and Higuchi [2] in 1993. In the last years, evolutionary techniques have generated astonishing circuits that are totally different from classically engineered circuits, and sometimes even superior, as presented by Thompson and Layzell [3]. Moreover, for specifications varying over time, evolutionary techniques achieved very promising results indicating their potential to construct self-adapting systems. Higuchi and Kajihara [4] presented case studies on evolved controllers for prosthetic hands and robot navigation. However, several problems remain to be solved, the major ones being the scalability and the robustness of evolved hardware.

Our long-term goal is the development of autonomous embedded systems that implement hardware functions (circuits) characterized by their functional quality and resource demand. We plan to leverage on three concepts to achieve a flexible adaptation: First, an *intrinsic evolutionary search process* adapts the system to slow changes in the environment. Second, radical changes in available resources are compensated for by replacing the operational circuit with a *preevolved alternative* which meets the new resource constraints. To this end, we store at any time an approximated Pareto front of circuit implementations. Third, a *reconfigurable system on chip* platform is the technology allowing for the replacement of circuits during runtime and for the implementation of an intrinsically evolvable system.

The main contribution of this paper is the development and comparison of multi-objective evolutionary techniques for hardware design. In Section 2, we first outline our architecture concept and then we review the few approaches that treat evolutionary hardware design as a multi-objective optimization problem. Section 3 presents our basic hardware representation model and a baseline genetic algorithm as well as two multi-objective evolutionary optimizers for hardware design. Experiments and results are discussed in Section 4. Finally, Section 5 summarizes the paper and outlines further work.

2 Architecture Concept

2.1 Autonomous Subsystem and Fitness Evaluation

Fig. 1 shows the envisioned architecture concept for an intrinsically evolved subsystem. The currently instantiated circuit reads input signals from sensors, computes its function, and writes output signals to actuators. The instantiated solution has to meet area and speed constraints. The intrinsic evolutionary algorithm (EA) applies the genetic operators selection, crossover and mutation to the candidate solutions stored in the population data structure. The fitness evaluation is based on test vectors which are also stored in the subsystem. Depending on the application and system resources, the EA can run continuously or from time to time. At any time, however, the subsystem maintains a set of approximated Pareto points for the required circuit. Specifically, whenever a new solution is found with better quality than the currently instantiated one (while still meeting area and speed constraints), the subsystem's controller can replace the instantiated solution with the new one. In case of a rapid change in available resources, the controller selects one of the circuits from the Pareto set that meets the new constraints.

The fitness evaluation is highly application-dependent. For any reasonably sized circuit, we will not be able to store all possible input vectors as test vectors. A full test coverage is, however, only necessary for functions that reveal a binary correctness property. The prime example are arithmetic functions, where we typically accept nothing less than 100% correctness. Much of the recent work in evolvable hardware has been concerned with the design of arithmetic circuits. We

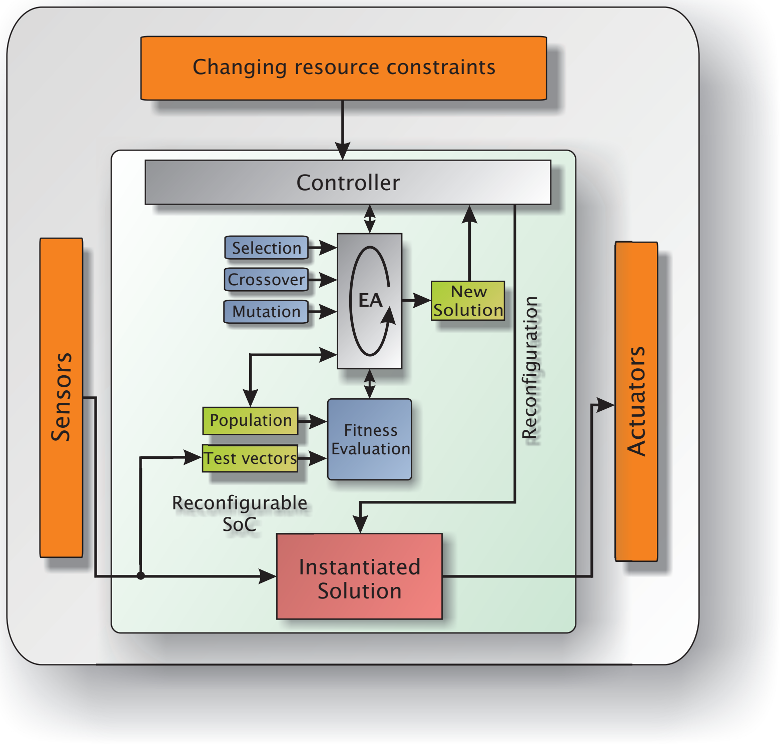


Fig. 1. Architecture concept for intrinsic evolution.

use arithmetic functions as test functions for algorithm design and evaluation, but do not view them as main candidates for autonomous evolution.

The ideal candidates for autonomous evolution are functions that are rated by a quality metrics rather than a binary correctness. Damiani et al. [6] presented the example of a hashing function where quality is measured by the ability to distribute the input keys evenly. Other examples include image compression where the quality is expressed by the compression rate and prosthetic hand control of Higuchi and Kajihara [4], where the quality is given by the percentage of correct classifications. In all these applications, the optimal circuit depends on input data which varies with time. For these functions it suffices to store a certain amount of test vectors that can either be static or being sampled during runtime. For example, Keymeulen et al. [7] designed an adaptive robot control with the objectives to avoid obstacles and reduce the distance to a given target. The robot acquires spatial information about its environment, building a model of it. New robot controllers are evolved and evaluated using this environment model without necessity of making a real-world test run.

2.2 Multi-objective Hardware Evolution

A central issue in our work is hardware evolution with multiple objectives, e.g., functional quality, area and speed. While multi-objective evolutionary optimizers have been successfully used in system-level synthesis and synthesis of analog circuits, there are only few projects dealing with multi-objective evolution of digital circuits. Kalganova and Miller [8] used a multi-stage fitness function to optimize for circuit correctness and hardware area. They evolved arithmetic circuits on a two-dimensional array of simple gates with an interconnect restricted to feed-forward wires. The fitness F of an individual is defined as:

$$F = \begin{cases} c & \text{if } c < \ 100\%, \\ c + \gamma & \text{else} \end{cases} \tag{1}$$

The parameter c denotes the percentage of the correct output bits of the circuit and γ is the number of gates in the array that are not used. As long as the circuit is incorrect, the selection process bases solely on the functional quality. Area is taken into account as soon as correctness is ensured. Coello et al. [9] address the same problem with a multi-objective search algorithm. The initial single correctness objective is redefined in a way that treats the function of each circuit output as a separate objective. The evolutionary search algorithm has to first meet all these objectives, and then area is taken into account. This approach actually turns constraints into objectives but still uses a multi-stage fitness function to optimize for area.

In contrast to related work, we use a multi-objective EA to optimize for several objectives simultaneously. We are most interested in functions without correctness property. Hence, we do not have to turn (correctness) constraints into objectives. Resource constraints are satisfied by the system's controller that selects a proper circuit for instantiation. Research in multi-objective evolutionary algorithms has identified two key issues subsumed by Zitzler et al. [10]: minimizing the distance between the approximated and the real Pareto front, and maintaining a diverse population to avoid premature convergence to a single objective. The remaining part of this paper presents our work in multi-objective optimizers for evolving digital hardware. This is the central algorithmic challenge in building the autonomous system outlined in Fig. 1.

3 Evolutionary Hardware Design

We use the Cartesian Genetic Programming (CGP) introduced by Miller and Thomson [11] in our work. CGP is a structural hardware model where a circuit is formed by combinational logic blocks arranged in a two-dimensional array and an interconnect (wires) between the blocks. The array consists of $n_c \times n_r$ combinational blocks, n_i primary inputs, and n_o primary outputs. The primary inputs can be connected to the inputs of any logic block in the array. A logic block in column c has n_n inputs that can be connected to the columns $c - l, \ldots, c - 1$ of the array and to the primary inputs, respectively. This ensures that no

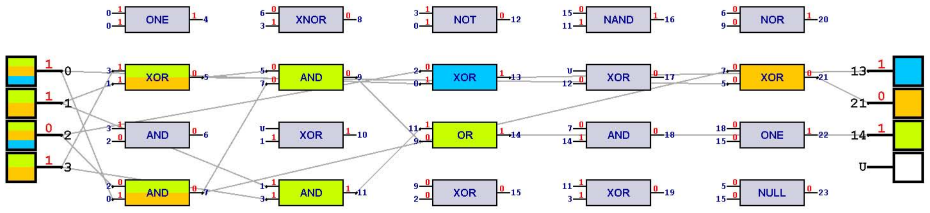


Fig. 2. 2×2 bit-adder evolved on the CGP model.

combinational feedback loops are generated. A combinational block implements one out of n_f different logic functions of its inputs. An individual is defined by its chromosome (genotype) with a length of $n_c \cdot n_r(n_n + 1) + n_o$.

Fig. 2 presents an example of a successfully evolved 2×2 bit-adder on a GCP model instance with $n_i = 4, n_o = 4, n_c = 5, n_r = 4, n_n = 2, n_f = 9$, and $l = 4$. The nine possible logic block functions have been chosen as `AND, ONE, XOR, NULL, NAND, NOT, NOR, OR,` and `XNOR`.

In the following, we outline the three algorithms used for evolving circuits:

Reference Algorithm GA is a standard single-objective genetic algorithm. The parameters are set as follows: The top 5% of the individuals are selected and transferred without any modification to the next generation. The recombination probability is chosen to be 90%. The individuals are recombined uniformly. We choose the mutation rate such that only one combinational block or wire is mutated each time the mutation operator is applied. In our implementation each recombined child is mutated once. These parameter settings have been used for the experiments described in the following section.

SPEA2 is a recent multi-objective evolutionary optimizer introduced by Zitzler et al. [10] with a structure shown in Fig. 3. SPEA2 maintains two sets of individuals: an archive that contains non-dominated individuals and a breeding population. In each generation, the two sets are merged and the fitness of the individuals is evaluated. The non-dominated individuals are then copied to the new archive. If the archive exceeds a predefined maximum size, SPEA2 applies a nearest neighbor density estimation technique to thin out clusters on the Pareto front. The fitness assigned to an individual considers the number of individuals it dominates - the dominance count, the number of individuals that are dominators - the dominance rank, and a density estimate based on the k-th nearest neighbor method. All individuals undergo a binary tournament selection which selects parents for the recombination and mutation.

TSPEA2 is an algorithm we have devised to put an increased selection pressure on one objective while trying to keep diversity. This should be beneficial for evolving circuits with a correctness property. Compared to SPEA2, we expect degraded fitness values for the other objectives. Both SPEA2 and TSPEA2 use an archive and a breeding population and a selection scheme based on Pareto dominance ranking. TSPEA2, however, checks as a first selection rule in a binary tournament whether one of the two individuals dominates the other regarding the main objective. TSPEA2 has been motivated by an earlier algorithm MO-Turtle GA presented by Trefzer et al. [12], that preferred a main and several random objectives during the evolution of analog circuits.

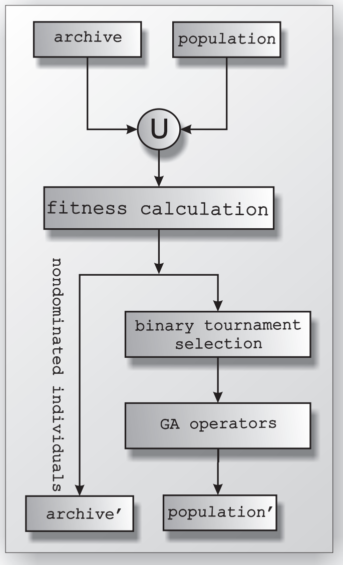


Fig. 3. Structure of the SPEA2 and TSPEA2 optimizers.

4 Experiments and Results

We have evolved several test functions with GA, SPEA2, and TSPEA2. In this section, we report on typical results for a 6-parity function and a hashing function. While the 6-parity function is an example for a function with a correctness property, the hashing function is rated by a non-binary quality metrics. The functional set available for the logic blocks in the CGP model comprises the 9 functions shown in Fig. 2. The parameters for crossover and mutation used in SPEA2 and TSPEA2 are set as described in Section 3. The tournament selection operator is configured to execute two tournaments before selecting one of the competitors as a parent. For all evolutionary algorithms, we conducted 10 optimization runs with a maximum of 100.000 generations.

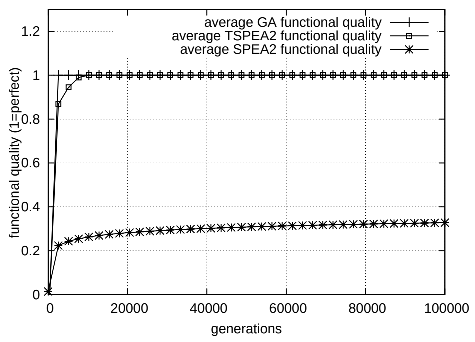
The delay of a circuit is in the range $\{0, \ldots, n_c + 1\}$. The fitness with respect to speed is determined as:

$$speed(c) = 1 - \frac{delay(c)}{n_c + 1} \tag{2}$$

The speed equals 1 for the fastest possible circuit and 0 for a circuit that has no connection at all from primary inputs to primary outputs. The number of logic blocks used by a circuit, denoted as $used\_blocks(c)$, is in the range $\{0, \ldots, n_c \cdot n_r\}$. Based on this number, the fitness with respect to area is defined as:

$$area(c) = 1 - \frac{used\_blocks(c)}{n_c \cdot n_r} \tag{3}$$

A circuit with minimal area gets an area value of 1, a circuit that utilizes all available logic blocks has an area value of 0.



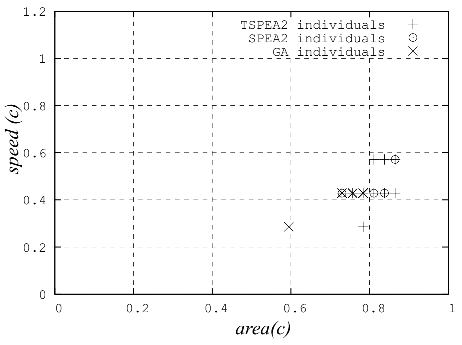


Fig. 4. Evolving the 6-parity function. Data from 10 experiments is shown.

4.1 6-parity

The used parameters for the CGP model are $n_c = n_r = n_i = 6$, $n_o = 1$, $n_n = 2$, $l = \frac{n_c}{2}$. For the parity function, a circuit's c fitness with respect to functional quality is defined as follows:

$$f(c) = \frac{1}{1 + \sum_{i \in \mathbb{B}^6} (\text{parity}(i) - c(i))^2}. \tag{4}$$

Thus, a correct parity function has a functional quality of 1. It is an easy task for a conventional GA to evolve a correct circuit for the 6-parity function. Using a population of size 100, only 69 generations were needed on average to evolve a fully functional circuit. In contrast to the GA, SPEA2 with an archive and population size of 100 evolved only four correct solutions overall and needed more than 30000 generations on average. With TSPEA2 preferring the functional quality, the search process converged faster with, on average, 903 generations to evolve a correct circuit. Fig. 4 shows the development of the average functional qualities for the three algorithms, and the speed and area parameters for correctly evolved circuits. Both SPEA2 and TSPEA2 found the same dominant solution. Moreover, TSPEA2 managed to discover a more diverse solution set compared to SPEA2. The conventional single-objective GA evolved correct circuits with inferior area and speed.

4.2 Hashing function

The hashing function has been evolved previously by Tettamanzi et al. [6]. To be able to compare their experiments with ours, we used the same CGP-parameters: $n_c = 8$, $n_r = 8$, $l = 8$ and $n_n = 4$. The difference to our work is that Tettamanzi et al. restricted wires to connect only to logic blocks in the same row. We have relaxed this constraint which leads to an improvement using a conventional GA.

The problem statement is as follows: Find a function $h : \mathbb{B}^{16} \rightarrow \mathbb{B}^8$ which maps a set M of 2^{12} keys to a set N of 2^8 indices in the most uniform way

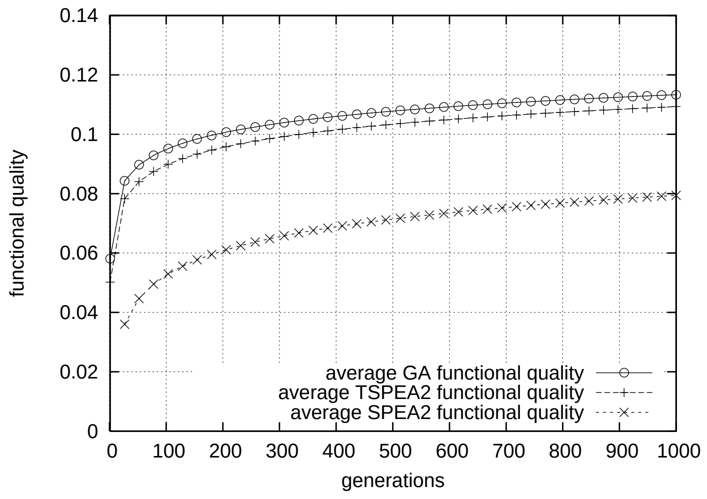


Fig. 5. Evolving the hashing function. Average fitness development over 10 experiments for GA, SPEA2, and TSPEA2.

possible. The fitness function is defined as:

$$f(c) = \frac{1}{1 + \frac{1}{|N|}\sum_{i=1}^{|N|}(|\{j|j \in M,\ c(j) = i\}| - \frac{|M|}{|N|})^2} \tag{5}$$

Tettamanzi et al. [6] evolved the best individual with a fitness value of 0.097785 after 257 generations. On our less constrained CGP model, the single-objective GA reached easily an average fitness beyond 0.1, as is shown in Fig. 5. After 257 generations, the best individual showed a fitness of 0.116469. Fig. 5 shows the fitness development for GA, SPEA2, and TSPEA2. As expected, TS-PEA2 performed close to GA while SPEA2 lagged behind. Fig. 6 displays the functional quality vs. area and speed. The figure shows two-dimensional projections of the Pareto front after 1000 generations. As expected, our experiments confirmed that a conventional GA optimizes the functional quality faster than SPEA2 and TSPEA2. Although TSPEA2 is close to GA measured in number of simulated generations, we have to note that simulating one generation in TSPEA2 takes about an order of magnitude longer than for GA. SPEA2 and TSPEA2 excel, however, in evolving solutions with improved area and speed. Table 1 lists the resulting functional qualities (best, worst and average case) after iterating for 1000 generations.

Comparing SPEA2 with TSPEA2, we note that SPEA2 did not evolve individuals with better area or speed. In fact, all individuals found by SPEA2 are dominated by individuals generated by TSPEA2. This is an interesting observation, as one would expect that TSPEA2, which prefers the functional quality over the other objectives, leads to a somewhat deteriorated Pareto front. This result has been consistent over all simulation runs with the hashing function.

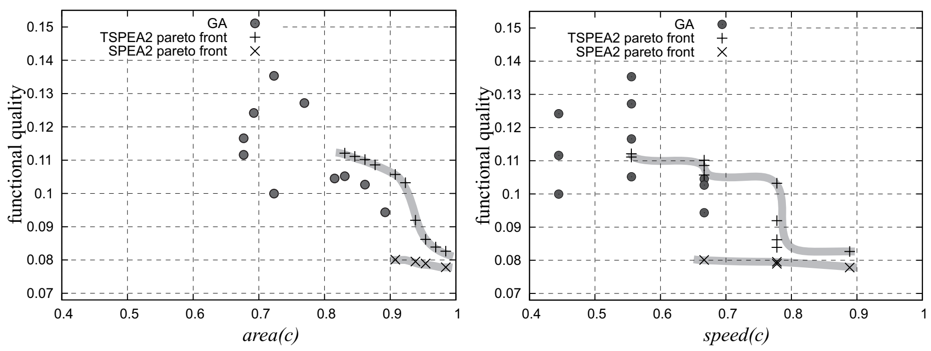


Fig. 6. Evolving the hashing function. 2D projections of the Pareto front for two typical populations. Also the objectives of the best individuals found by GA during the 10 experiments are plotted.

Table 1. Evolving the hashing function. Reached functional qualities after 1000 generations.

	GA	SPEA2	TSPEA2
best	0.135	0.084	0.125
worst	0.094	0.075	0.092
average	0.114	0.079	0.110

A possible explanation is that in our experiments the objectives are not necessarily conflicting. Driving the evolution towards functional quality will then also improve area and/or speed. However, this may not be generalized as design experience shows that for many circuits the functional quality, speed and area are indeed conflicting.

5 Summary and Further Work

In this paper, we have outlined a novel architectural approach for self-adaptive autonomous embedded systems. Simulated evolution is used to adapt to slow changes in the environment; switching to preevolved alternatives is the proper reaction to drastic changes in the available resources. We have then focused on multi-objective evolutionary optimizers and compared the known algorithm SPEA2 with the newly devised technique TSPEA2 and a baseline GA. We have presented comparisons of these algorithms for two test functions.

An implementation of the overall system shown in Fig. 1 is ongoing. Further work will focus on the scalability problem and investigate variants of the CGP model with more coarse-granular building blocks. Moreover, we will validate our observations on larger test functions.

6 Acknowledgement

This work was supported by the German Research Foundation under project number PL 471/1-1 within the priority program *Organic Computing*.

References

1. de Garis, H.: Evolvable Hardware – Genetic Programming of a Darwin Machine. In: Proceedings International Conference on Artificial Neural Networks and Genetic Algorithms (ICANNGA), Springer (1993)
2. Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., Furuya, T.: Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In: Proceedings 2nd International Conference on Simulation of Adaptive Behavior (SAB), MIT Press (1993) 417–424
3. Thompson, A., Layzell, P.: Analysis of Unconventional Evolved Electronics. Communications of the ACM **42** (1999) 71–79 ACM Press.
4. Higuchi, T., Kajihara, N.: Evolvable Hardware Chips for Industrial Applications. Communications of the ACM **42** (1999) 60–66 ACM Press.
5. Walker, J., Garrett, S., Wilson, M.: Evolving Controllers for Real Robots: A Survey of the Literature. Adaptive Behavior **11** (2003) 179–203 MIT Press.
6. Damiani, E., Liberali, V., Tettamanzi, A.: Evolutionary Design of Hashing Function Circuits Using an FPGA. In: International Conference on Evolvable Systems (ICES), Springer (1998) 36–46
7. Keymeulen, D., Konaka, K., Iwata, M., Kuniyoshi, Y., Higuchi, T.: Robot Learning Using Gate-Level Evolvable Hardware. In: EWLR-6: Proceedings of the 6th European Workshop on Learning Robots, London, UK, Springer-Verlag (1998) 173
8. Kalganova, T., Miller, J.: Evolving More Efficient Digital Circuits by Allowing Circuit Layout Evolution and Multi-Objective Fitness. In: The First NASA/DoD Workshop on Evolvable Hardware, Pasadena, California, IEEE Computer Society (1999) 54–63
9. Coello Coello, C.A.: Treating Constraints as Objectives for Single-Objective Evolutionary Optimization. In: Engineering Optimization. Volume 32., Taylor and Francis (2000) 275–308
10. Zitzler, E., Laumanns, M., Thiele, L.: SPEA2: Improving the Strength Pareto Evolutionary Algorithm. Technical Report 103, Gloriastrasse 35, CH-8092 Zurich, Switzerland (2001)
11. Miller, J.F., Thomson, P.: Cartesian Genetic Programming. In: Proceedings of the European Conference on Genetic Programming, London, UK, Springer-Verlag (2000) 121–132
12. Trefzer, M., Langeheine, J., Meier, K., Schemmel, J.: Operational Amplifiers: An Example for Multi-objective Optimization on an Analog Evolvable Hardware Platform. In: International Conference on Evolvable Systems (ICES), Springer (2005) 86–97