Evolutionary hardware design

Lukas Sekanina[a]

[a]Brno University of Technology, Faculty of Information Technology
Bozetechova 2, 61266 Brno, Czech Republic

ABSTRACT

Since the early 1990's researchers have begun to apply evolutionary algorithms to synthesize electronic circuits. Nowadays it is evident that the evolutionary design approach can automatically create efficient electronic circuits in many domains. This paper surveys fundamental concepts of evolutionary hardware design. It introduces relevant search algorithms such as Cartesian genetic programming (CGP). Several case studies are presented demonstrating strength and weakness of the method. Target domains are combinational circuit synthesis where the goal is to minimize the number of gates, image filter design intended for field programmable gate arrays (FPGAs) where the goal is to obtain the quality of filtering of conventional methods for a significantly lower cost on a chip and evolution of benchmark circuits for evaluation of testability analysis methods. Evolved circuits are compared with the best-known conventional designs. FPGAs are presented as accelerators for evolutionary circuit design and circuit adaptation.

Keywords: evolvable hardware, genetic programming, field programmable gate array, logic design, image filter

1. INTRODUCTION

Evolvable hardware (EHW), introduced at the beginning of nineties,[1,2] is now considered as a subfield of natural computing. In evolvable hardware, bio-inspired algorithms are combined with reconfigurable devices to either automatically design hardware or adapt hardware. Among the bio-inspired algorithms, evolutionary algorithms (EAs) play the most important role. While evolutionary hardware design is the use of EAs for creating innovative (and sometimes patentable) physical designs, the goal of adaptive hardware is to endow physical systems with some adaptive and fault tolerant characteristics. It should be noted that EAs are here primarily utilized to *design* new solutions rather than *optimize* existing solutions. A more detailed introduction to evolvable hardware can be found in monographs[3–9] or survey articles.[10–12]

We will only deal with evolutionary hardware design in this paper. John Koza has reported tens of human-competitive results in various areas of science and technology obtained automatically using evolutionary design techniques, in particular using genetic programming (GP).[13] The GP-based approach has been adopted by analog circuit design community.[14–16] In case of digital circuits, the evolutionary approach has been utilized for logic synthesis[17–20] as well as for design of various application-specific circuits such as filters, classifiers and predictors.[21–23]

The aim of this paper is to survey basic principles of the evolutionary circuit design and present several case studies where the evolutionary circuit design has achieved new and useful solutions. We will deal with digital circuits only. We will explain why these new solutions could be obtained and what role is played by field programmable gate arrays (FPGA) in current research on evolvable hardware.

The rest of the paper is organized as follows. After introducing the concepts of evolvable hardware and evolutionary design in Section 2, Cartesian genetic programming (CGP) will be presented in Section 3 as one of the most prominent search algorithms for digital circuit evolution. Section 4 gives three case studies where different types of problems are presented that were successfully solved by evolutionary design method. Approaches to FPGA-based acceleration of evolutionary circuit design are outlined in Section 5. Finally, conclusions are given in Section 6.

E-mail: sekanina@fit.vutbr.cz

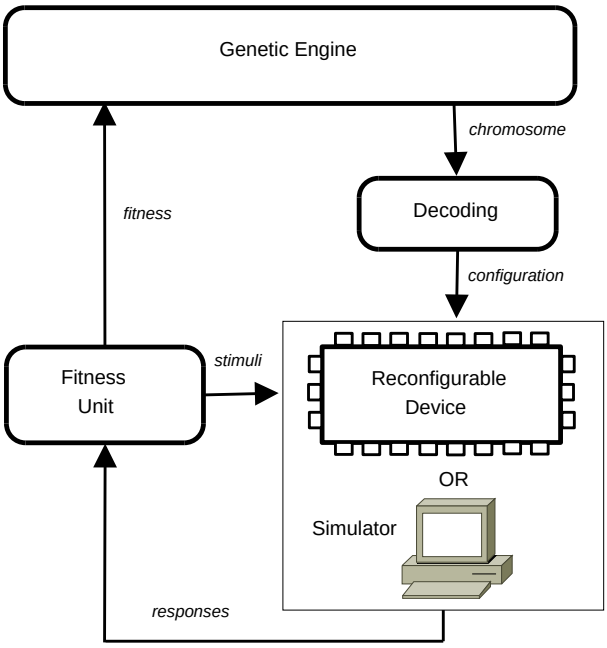

Figure 1. High-level description of the evolvable hardware approach

2. EVOLVABLE HARDWARE

2.1 The concept

Figure 1 explains the concept of evolvable hardware: electronic circuits that are encoded as strings of symbols are constructed and optimized by the evolutionary algorithm in order to obtain a circuit implementation satisfying the specification given by designer. In order to evaluate a candidate circuit, reconfigurable circuit (or its simulator) is reconfigured using a new configuration created on the basis of the chromosome (i.e. configuration) content. The configured device is then stimulated by a chosen set of input vectors; its responses are collected and compared with target responses. Finally, the fitness score is calculated reflecting to what extend a candidate circuit satisfies the specification. This process is repeated for every individual generated by EA. New individuals are created by applying genetic operators (such as mutation and crossover) on existing circuit configurations. High-scored candidate circuits have got a higher probability that their genetic material will be selected for creating next generations. The process of evolution is terminated when a perfect solution is obtained or when a certain number of generations is evaluated.

If all candidate circuits are evaluated in a physical reconfigurable device then the approach is called *intrinsic evolution*. If the evolution is performed using a circuit simulator and only a resulting circuit is uploaded to a reconfigurable device the approach is called *extrinsic evolution*. Adrian Thompson has discovered using the XC6216 FPGA chip that performing evolution directly in hardware can lead to solutions that exploit a reconfigurable device and external environment in a new way. Evolution can utilize non-modeled features of the platform to create a solution which is unreachable by convention model-based design methods.[24] Hence it is important to distinguish between unconstrained intrinsic evolution and extrinsic evolution whose result is always reconstructible in a device simulator.

The main motivation for circuit evolution can be seen in the fact that the evolutionary approach can produce innovative designs and increase the level of design automation. In the context of evolutionary circuit design, the term *innovative* means that a resulting circuit exhibits a better quality in some aspects with respect to existing designs of the same category. For example, the solution would occupy a smaller area on a chip, compute faster, provide a better precision, reduce the energy consumption, increase the reliability etc. The main contribution of the evolutionary hardware design can be seen in the areas where it is impossible to provide a perfect specification for target implementations and conventional design methods are based on experience and intuition rather than

on a fully automated methodology. Then, the EA can explore the "dark corners" of design spaces which humans have left unexplored.[11]

2.2 Scalability issues

The *scalability problem* has been identified as one of the most difficult problems the researchers are faced with in the evolvable hardware field. In this context, the scalability problem means such situation in which the evolutionary algorithm is able to provide a solution to a small problem instance; however, only unsatisfactory solutions can be generated for larger problem instances.

All the approaches proposed to overcome the scalability problem utilize a kind of domain knowledge which helps in focusing the search algorithm on promising areas of the search space and reducing the computation overhead. The scalability problem can primarily be seen from two perspectives: representation and fitness evaluation.

From the viewpoint of the *scalability of representation*, the problem is that long chromosomes which are usually required to represent complex solutions imply large search spaces that are typically difficult to search. In order to evolve large designs and simultaneously keep the size of chromosome small, various techniques have been proposed, including functional level evolution,[25] incremental evolution,[26] development,[27] modularization[28] and their combinations.[19,23]

Another issue is the scalability of *fitness evaluation*, the problem that complex candidate solutions require a lot of time to be evaluated. In case of the combinational circuit evolution, the evaluation time of a candidate circuit grows exponentially with the increasing number of inputs (assuming that all possible input combinations are tested in the fitness function). In order to reduce the time of evaluation, various techniques can be adopted, including partial evaluation, fitness estimation and formal verification.[20,29]

2.3 Types of applications in digital circuit evolution

In case of digital circuit evolution, two main application classes can be identified. In the first class, the goal is to obtain a circuit which responds *perfectly* for all requested assignments to the inputs and which exhibits a kind of innovation such as close-to optimum number of gates, small delay or low power consumption. The fitness function is usually constructed in such a way that all requested assignments are applied to the inputs of a candidate circuit and the fitness value is defined as the number of bits that the candidate circuit computes correctly (additional criteria can be incorporated as well). The evolution of arithmetic circuits is a typical example of that class. Because the evaluation time depends exponentially on the number of inputs, the evaluation is tractable only for small circuits. Only relatively small and simultaneously innovative designs have been evolved in this domain, for example area-efficient parallel multipliers with up to 8 inputs.[17,30] Nevertheless, for some very specific cases, such as linear transforms synthesis,[31] the scalability problem can effectively be eliminated.

In the second class, it is sufficient to evolve a circuit which responds correctly for a reasonable *subset* of all possible input vectors because the specification is in principle incomplete; design of filters, classifiers and predictors are typical examples.[21–23] The fitness value is usually calculated on the basis of circuit response obtained for a carefully chosen training set, a subset of all possible input vectors. The design problem can be considered as a symbolic regression problem. Behavior of evolved solution has to be validated using a test set at the end of evolution, i.e. using vectors unseen during the evolution.

3. SEARCH ALGORITHMS

It can be seen that the design process is transformed to a search process in case of evolvable hardware. The *search space* is a space that contains all possible considered solutions to the problem, and a point in that space defines a solution. Instead of working with one solution at a time, evolutionary algorithms operate with the *population* of *candidate solutions* (individuals). Every new population is formed using genetically inspired operators (like *crossover* and *mutation*) and through a selection pressure, which guides the evolution towards better areas of the search space. In order to propose an efficient search algorithm, it is particularly important to find a suitable problem representation, genetic operators and fitness function. Next subsections describe Cartesian Programming (CGP)[32] – a method that has been successfully used in various applications of digital circuit evolution.

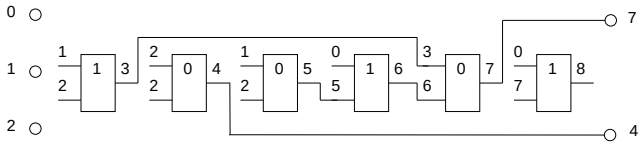

Figure 2. Example of a candidate circuit in CGP with parameters: $n_c = 6$, $n_r = 1$, $n_i = 3$, $n_o = 2$, $l = 6$, $n_a = 2$, $\Gamma =$ {NOR (0), NAND (1)}. Chromosome: 1, 2, **1**, 2, 2, **0**, 1, 2, **0**, 0, 5, **1**, 3, 6, **0**, 0, 7, **1**, 7, 4. (Gate function is typed in bold.)

3.1 Problem Representation

To model a general combinational circuit a candidate circuit is represented as an array of n_c (columns) $\times$ n_r (rows) of programmable gates. The number of inputs, n_i, and outputs, n_o, is fixed. Each gate can be connected either to the output of a gate placed in previous l columns or to one of circuit inputs. Setting of the l parameter allows to control the maximum circuit delay. Feedback is not allowed. Each gate is programmed to perform one of n_a-input functions defined in the set Γ (n_f denotes $|\Gamma|$). Each gate is encoded using $n_a + 1$ integers where values $1 \ldots n_a$ are indexes of the input connections and the last value is the function code. Every circuit is encoded using $n_c.n_r.(n_a+1)+n_o$ integers. Figure 2 shows an example of a candidate circuit and its chromosome.

3.2 Search Strategy

The initial population is constructed either randomly or by a heuristic procedure. Every new population consists of the best individual of the previous population and its λ offspring individuals. The offspring individuals are created using a point mutation operator which modifies h randomly selected genes of the chromosome, where h is a user-defined value. In order to keep a genetic diversity a new parent individual is selected from those highest-scored individuals who have not served as parent in the previous population.[33]

3.3 Fitness Function

In case of small combinational circuit evolution, the fitness value of a candidate circuit is defined as:[34]

$$f = \begin{cases} b & when \quad b < n_o 2^{n_i}, \\ b + (n_c n_r - z) & otherwise, \end{cases} \tag{1}$$

where b is the number of correct output bits obtained as response for all possible assignments to the inputs, z denotes the number of gates utilized in a particular candidate circuit and $n_c.n_r$ is the total number of available gates. It can be seen that the last term $n_c n_r - z$ is considered only if the circuit behavior is perfect, i.e. $b = b_{max} = n_o 2^{n_i}$. Similarly, delay or power consumption may be optimized. A multi-objective formulation of circuit evolution problem was also proposed.

For symbolic regression problems, such as filter design, the goal is usually to minimize the mean absolute error of a candidate circuit response Y and target response T. The fitness function may be defined as

$$f = \sum_{j=1}^{k} |Y_j - T_j| \tag{2}$$

where k is the number of sampled points.

4. CASE STUDIES

We will present three case studies where digital circuit evolution has provided new solutions with respect to conventional human designs. While target applications are completely different, evolutionary method remains almost unchanged (expect the fitness function formulation).

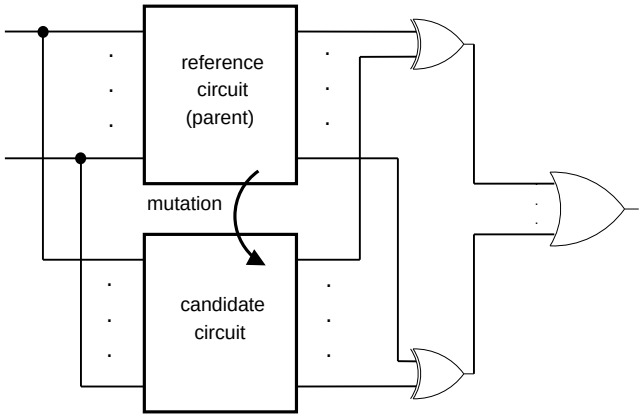

Figure 3. Functional equivalence checking of reference circuit and candidate circuit.

4.1 Logic synthesis

Logic synthesis is currently mainly based on deterministic algorithms. Any synthesis algorithm operates over a circuit representation, such as And-Inverter Graph or binary decision diagram. A key feature of common synthesis algorithms is that any intermediate transformation produces a circuit which is functionally equivalent to the original circuit. Some possibly efficient circuit transformations are not allowed in this process simply because they produce functionally incorrect solutions. However, iterative applications of such partially correct transforms may lead to a fully functional and area-efficient solution after some time. Since it was shown that conventional synthesis algorithms are not capable of efficient synthesis for some circuit classes,[35,36] it is reasonable to explore whether evolutionary approach could improve the result of conventional synthesis.

In case of logic synthesis, it must be ensured that the resulting circuit works correctly for all possible assignments to the inputs. In order to satisfy this requirement it is necessary to introduce a new fitness function; equation 1 is not applicable because of the scalability problems.

A solution to this problem was proposed by Vasicek and Sekanina.[20] It starts with a fully functional circuit (also called a reference circuit) which can be obtained using conventional synthesis. CGP is then applied to minimize the number of gates. In order to quickly check whether a candidate circuit and reference circuit are functionally equivalent, an equivalence checking algorithm based on the satisfiability problem (SAT) solver is employed. Figure 3 shows the approach adopted for functional checking. Both circuits have to be converted to a conjunctive normal form (CNF) and submitted to the SAT solver. Additional clauses representing the XOR-OR comparator network are also submitted to the SAT solver. MiniSAT 2 (version 070721) has been used as an equivalence decision tool.[37] If the candidate circuit and reference circuit are functionally equivalent the fitness score equals the number of gates in the candidate circuit.

The CGP parameters are as follows: $\lambda = 2$, $\Gamma = \{\text{BUFFER}, \text{NOT}, \text{AND}, \text{OR}, \text{XOR}, \text{NAND}, \text{NOR}, \text{XNOR}\}$, $l = N_g$, 1 mutation/chromosome, $n_c = N_g$ and $n_r = 1$ where N_g is the number of gates of the reference (seed) circuit, i.e. in the circuit created using conventional synthesis. The circuits were mapped to the 2-input gates using the conventional SIS tool.

The experimental evaluation was performed using the LGSynth93 benchmarks circuits, some of them possessing over 100 inputs and 1000 gates, i.e. sizes intractable for standard CGP. Results of 12-hour optimization for each benchmark circuit were compared with conventional academia synthesis tools (ABC and SIS) and three commercial synthesis tools. The average reduction provided by CGP is 25% gates.[38] Figure 4 shows the progress of evolution for the apex1 circuit.

4.2 Image filter evolution

We have mentioned in Section 2.3 that a solution providing perfect responses for all possible stimuli is not achievable in some applications, for example in domain of filters or classifiers. An evolutionary algorithm is then used to find a solution showing a minimum error for training data. Evolutionary design of image filters is one

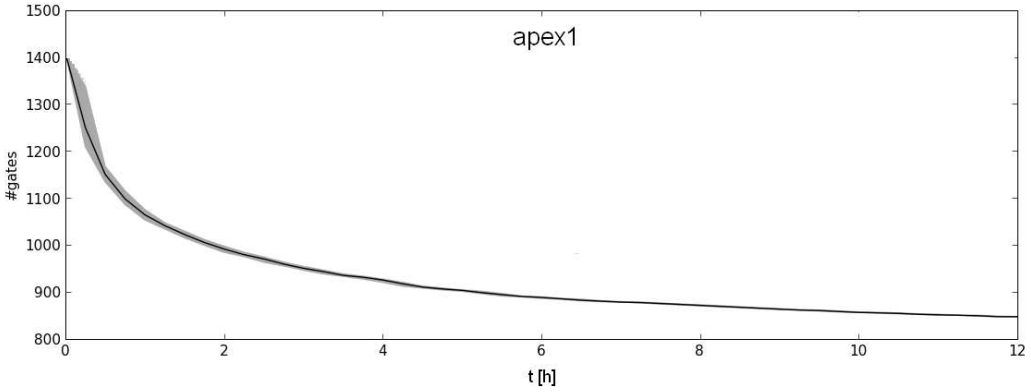

Figure 4. Minimum, maximum and mean number of gates obtained for the apex1 circuit out of 50 independent runs of CGP seeded using the result of SIS.

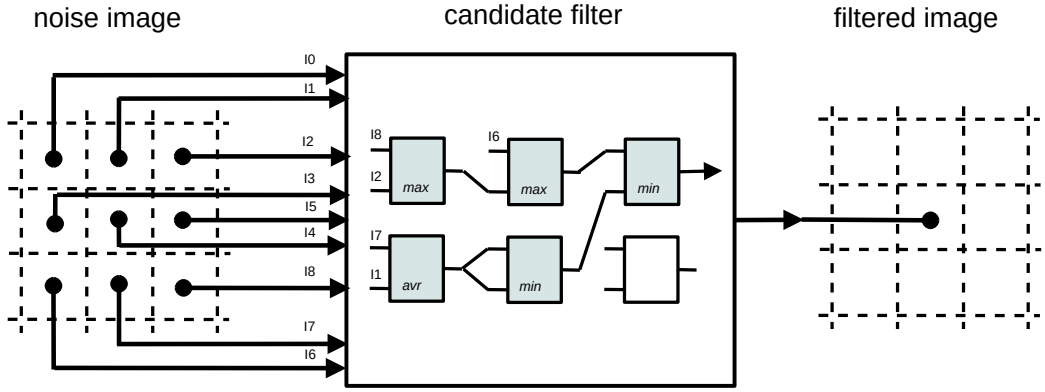

Figure 5. The utilization of CGP for image filter evolution.

of applications where the evolutionary approach can automatically produce solutions that are competitive with conventional solutions in terms of filtering quality and, moreover, reduce the implementation cost on a chip.[7,39]

CGP was employed to evolve an image filter suppressing a particular kind of noise (e.g. salt and pepper noise or burst noise). Edge detectors have also been evolved. In this scenario, every image filter operating with the 3×3-pixel kernel is considered as a function (a digital circuit in the case of hardware implementation) of nine 8-bit inputs and a single 8-bit output, which processes grayscale (8-bits/pixel) images. As Fig. 5 shows, every pixel value of the filtered image is calculated using a corresponding pixel and its eight neighbours in the processed image.

Any candidate filter is represented using $n_c \times n_r$ nodes, where a typical grid size is $n_c = 8$, $n_r = 4$. The setting of other CGP parameters is: $n_i = 9$, $n_o = 1$, $l = 1$, $n_a = 2$ and $\lambda = 8$. Each node represents a two-input function which receives two 8-bit values and produces an 8-bit output. Table 1 lists the functions that were confirmed as useful for this task.[7] It should be noted that these functions are also suitable for hardware implementation (i.e. there are no complex functions, such as multiplication or division).

CGP uses a single genetic operator – mutation – which modifies 5% of the chromosome. The initial population is randomly generated. The evolution is usually terminated when a predefined number of generations is exhausted.

Table 1. Elementary functions for image filter evolution

255	constant	$x \gg 1$	right shift by 1	x	identity	$x \gg 2$	right shift by 2
$255 - x$	inversion	$swap(x, y)$	swap nibbles	$x \vee y$	bitwise OR	$x + y$	+ (addition)
$\bar{x} \vee y$	bitwise $\bar{x}$ OR y	$x +^S y$	+ with saturation	$x \wedge y$	bitwise AND	$(x + y) \gg 1$	average
$\overline{x \wedge y}$	bitwise NAND	$max(x, y)$	maximum	$x \oplus y$	bitwise XOR	$min(x, y)$	minimum

In order to evolve an image filter capable of removing a given type of noise, the original uncorrupted (training) image is needed to determine the fitness value. The goal of CGP is to minimize the difference between the original image and the filtered image. Generality of evolved filters (i.e., whether they can operate sufficiently also for other images containing the same type of noise) is tested by means of a test (validation) set. The quality of filtering has to be numerically expressed. For this purpose, the *mean difference per pixel* is calculated

$$mdpp = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} |v(i,j) - w(i,j)| \tag{3}$$

where $N \times M$ is the size of image (typically $M = N = 128$), v denotes the filtered image and w denotes the original image. The goal of evolution is to minimize the *mdpp* value.

Extensive experimental evaluation of this method has been performed in literature.[7,39] A typical filter evolved to suppress the salt and pepper noise exhibits a 5 dB improvement in filtering quality (the mean value over 25 test images) while its implementation cost in FPGA is reduced by 25% with respect to standard median filter.[39]

4.3 Evolution of benchmark circuits with predefined testability

Over the years, there have been many attempts to create and use benchmark circuits for evaluation and comparison of algorithms performing the testability analysis of digital circuits. An EA-based automated approach for generating synthetic benchmark circuits with predefined complexity and testability properties has been proposed by Pecenka et al.[29]

In order to generate a benchmark circuit the user is required to provide: circuit interface, a list of circuit components, required average controllability, required average observability and EA parameters (population size, mutation rate, generation count etc.). The goal of evolution is to find an interconnection of given components leading to a circuit with the controllability and observability as close as possible to required controllability and observability. It should be noted that circuit functionality is not evolved; only testability properties are measured in the fitness function. Although candidate circuits may be very large (million of gates after synthesis) their testability must be obtained in a short time. Fortunately, that is possible for this application. The testability analysis is performed in the fitness function by an algorithm with quadratic time complexity.[29] The testability of resulting circuits was verified by a commercial ATPG tool.

Since benchmark circuits are designed by means of the EA, they can contain constructions that do not usually appear in circuits designed by classical design techniques. Thus, the use of evolved benchmark circuits in the process of evaluating new testability analysis tools can reveal problems of the tools that remain hidden when conventional benchmark circuits are used. The developed method was also utilized to create a set of 35 sequential synthetic benchmark circuits that are available at both RT- and gate levels. Evolved benchmark circuits currently represent the most complex benchmark circuits with a known level of testability.

Similarly to image filter evolution, EA attempts to minimize the error exhibited by a candidate solution when compared with required values. For complex real-world problems it is impossible to obtain a perfect solution, i.e. error = 0. However, a search-based EA is very competitive in minimizing the error in problems where uncertainty is implicitly present. Conventional top-down methods have provided only limited solutions.

5. FPGA ACCELERATORS

Various FPGA accelerators have been proposed for evolutionary circuit design. Initially, the FPGA served to accelerate fitness calculation only.[8] Recent implementations have included the fitness evaluation as well as a search engine on a single FPGA. The EA is implemented either as a custom logic controller or as a program running in the on-chip hard-core processor (such as PowerPC in Xilinx FPGAs) or a soft-core processor (such as MicroBlaze of Xilinx).[12] In order to instantiate and evaluate candidate circuits, a support for dynamic partial reconfiguration is needed. Some implementations have utilized the internal reconfiguration available via the internal configuration access port (ICAP) interface in Xilinx FPGAs. Other implementations avoid using the ICAP-based solution by creating a new reconfigurable layer on the top of the FPGA, so-called *virtual reconfigurable circuit* (VRC).[40]

5.1 Native reconfiguration for evolvable hardware

A typical evolvable hardware system, which utilizes the ICAP interface, operates with a set of pre-synthesized bit streams for several well-defined areas of the FPGA. The EA which is typically executed on the MicroBlaze or PowerPC processor may perform a mutation in such a way that a randomly selected area is configured using a randomly selected bit stream. The reconfiguration is performed dynamically and partially though the ICAP. This kind of systems was used to evolve combinational circuits[41,42] and image filters.[43] The advantage of this approach is that there is no area overhead in comparison with multiplexer-based VRCs. However, the reconfiguration process may be slow because it is necessary to follow the internal organization of a particular FPGA and upload many frames (elementary reconfiguration units of the Xilinx FPGA) even when only a small change in a configuration is required. For example, reconfiguration of a single processing element in the task of image filter evolution requires 12 μs when ICAP is executed at 250 MHz in the Virtex-5 LX110T FPGA.[43]

5.2 Virtual reconfigurable circuits

The basic idea of VRC-based accelerators is to implement the CGP circuit model as a reconfigurable circuit on the FPGA. Its configuration is defined using a bit stream which is stored in a configuration register implemented also in the FPGA.[40] Figure 6 shows an example of VRC – the 32 two-input Configurable Logic Elements (CLEs) organized in an array of 4 rows and 8 columns. A CLE can be programmed to perform one of the following functions: AND, OR, XOR and NAND. The configuration bits control a set of multiplexers which are responsible for interconnecting the CLEs and selecting their logic functions. In total, 8 bits define the configuration of a single CLE. The primary outputs $Z0 \ldots Z3$ are connected directly to CLEs of the last column. Since every CLE contains a register the whole structure works as a pipeline. The reconfiguration process is also pipelined. The main advantage of VRC is that the processing array is optimized for a particular task and its reconfiguration is very fast (effectively 1 clock cycle is needed). Since VRC does not utilize any features specific to a particular FPGA chip, it can be downloaded to any FPGA of sufficient capacity. A support for partial reconfiguration is not required. A huge area overhead is the main disadvantage of VRCs.

It has been demonstrated that a single-chip FPGA-based accelerator of image filter evolution running at 100 MHz can provide approx. 44 times higher performance in comparison with a common PC running at 2.4 GHz.[39] Speedup of 170 has been achieved when four candidate solutions were evaluated in parallel in the same FPGA XC2VP50 chip.[44]

6. CONCLUSIONS

In this paper, we have presented the concept of evolvable hardware and evolutionary circuit design. We have shown using three case studies that evolutionary design is capable of creating innovative solutions in the area of digital circuits. Finally, two approaches to FPGA implementation of accelerators for evolutionary circuit design have been sketched. We can observe that evolutionary computing is competitive with conventional techniques in several areas of digital design. However, the improvement is obtained for a cost of runtime which is, on the other hand, the main disadvantage of evolutionary circuit design nowadays.

ACKNOWLEDGMENTS

This work was supported by the grant Natural Computing on Unconventional Platforms GP103/10/1517, the FIT grant FIT-11-S-1 and the research plan Security-Oriented Research in Information Technology, MSM0021630528.

REFERENCES

[1] Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., and Furuya, T., "Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine," in [*Proc. of the 2nd International Conference on Simulated Adaptive Behaviour*], 417–424, MIT Press (1993).

[2] de Garis, H., "Evolvable Hardware – Genetic Programming of a Darwin Machine," in [*International Conference on Artificial Neural Networks and Genetic Algorithms ICANNGA'93*], (1993).

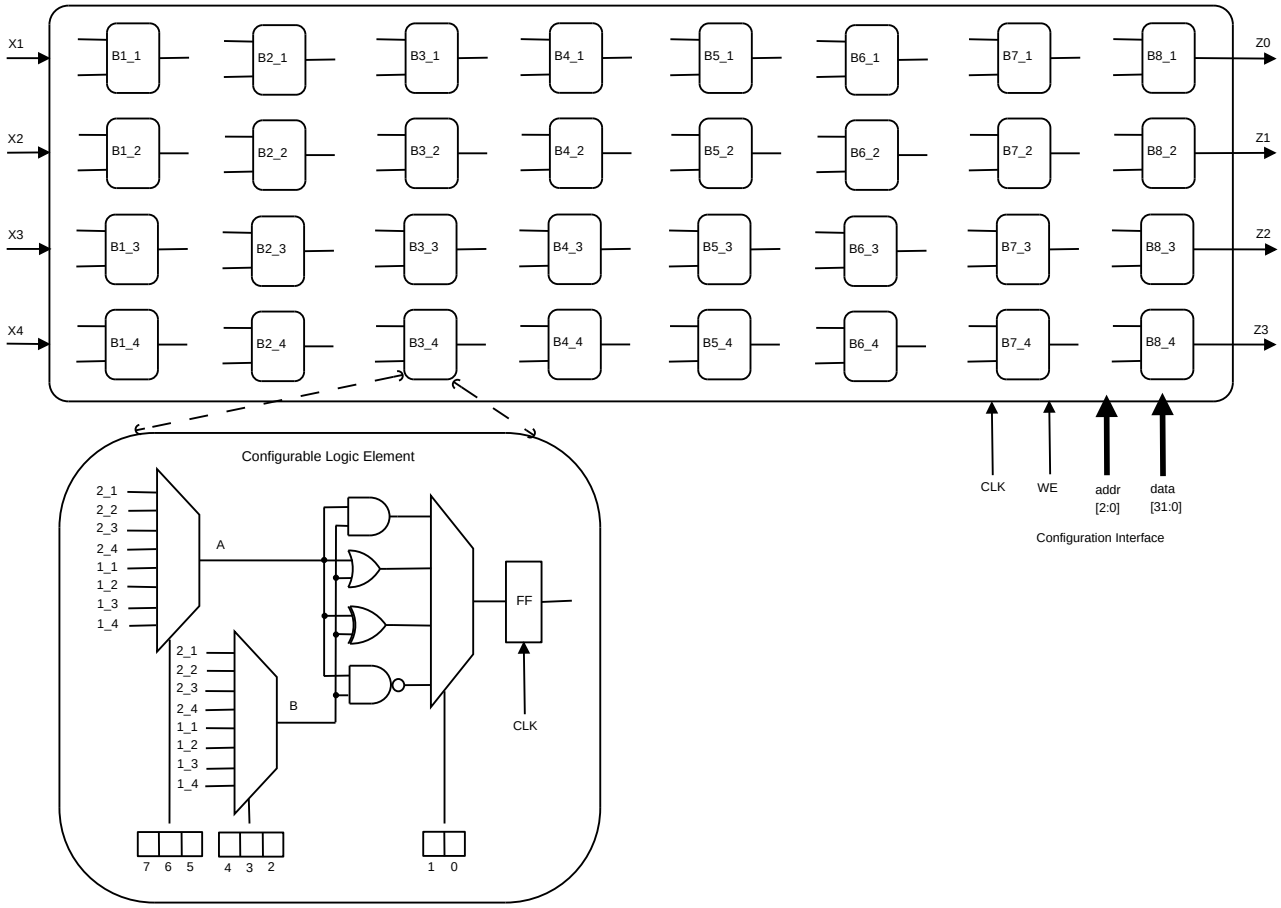

Figure 6. Example of virtual reconfigurable circuit and its implementation.

[3] Zebulum, R., Pacheco, M., and Vellasco, M., [*Evolutionary Electronics – Automatic Design of Electronic Circuits and Systems by Genetic Algorithms*], The CRC Press International Series on Computational Intelligence (2002).

[4] Thompson, A., [*Hardware Evolution: Automatic design of electronic circuits in reconfigurable hardware by artificial evolution*], Springer, London (1999).

[5] Greenwood, G. and Tyrrell, A. M., [*Introduction to Evolvable Hardware*], IEEE Press (2007).

[6] Higuchi, T., Liu, Y., and Yao, X., [*Evolvable Hardware*], Springer (2006).

[7] Sekanina, L., [*Evolvable Components: From Theory to Hardware Implementations*], Natural Computing Series, Springer Verlag (2004).

[8] Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A., [*Genetic Programming III: Darwinian Invention and Problem Solving*], Morgan Kaufmann Publishers, San Francisco, CA (1999).

[9] Koza, J. R., Keane, M. A., Streeter, M. J., Mydlowec, W., Yu, J., and Lanza, G., [*Genetic Programming IV: Routine Human-Competitive Machine Intelligence*], Kluwer Academic Publishers (2003).

[10] Gordon, T. G. H. and Bentley, P. J., "Evolving hardware," in [*Handbook of Nature-Inspired and Innovative Computing*], Zomaya, A. Y., ed., 387–432, Springer (2006).

[11] Lohn, J. D. and Hornby, G. S., "Evolvable hardware: Using evolutionary computation to design and optimize hardware systems," *IEEE Computational Intelligence Magazine* **1**(1), 19–27 (2006).

[12] Sekanina, L., "Evolvable hardware," in [*Handbook of Natural Computing – in print*], Rozenberg, G., Baeck, T., and Kok, J. N., eds., 1–49, Springer Verlag (2010).

[13] Koza, J. R., "Human-competitive results produced by genetic programming," *Genetic Programming and Evolvable Machines* **11**(3–4), 251–284 (2010).

[14] McConaghy, T., Eeckelaert, T., and Gielen, G. G. E., "Caffeine: Template-free symbolic model generation of analog circuits via canonical form functions and genetic programming," in [*Proc. of the Design, Automation and Test in Europe, DATE*], 1082–1087, IEEE Computer Society (2005).

[15] McConaghy, T., Palmers, P., Gielen, G. G. E., and Steyaert, M., "Simultaneous multi-topology multi-objective sizing across thousands of analog circuit topologies," in [*Proc. of Design Automation Conference – DAC 2007*], 944–947, IEEE Computer Society (2007).

[16] Das, A. and Vemuri, R., "A graph grammar based approach to automated multi-objective analog circuit design," in [*Proc. of the Design, Automation and Test in Europe, DATE*], 700–705, IEEE Computer Society (2009).

[17] Vassilev, V., Job, D., and Miller, J. F., "Towards the Automatic Design of More Efficient Digital Circuits," in [*Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*], Lohn, J., Stoica, A., Keymeulen, D., and Colombano, S., eds., 151–160, IEEE Computer Society, Los Alamitos, CA, USA (2000).

[18] Stomeo, E., Kalganova, T., and Lambert, C., "Generalized disjunction decomposition for evolvable hardware," *IEEE Transaction Systems, Man and Cybernetics, Part B* **36**(5), 1024–1043 (2006).

[19] Shanthi, A. P. and Parthasarathi, R., "Practical and scalable evolution of digital circuits," *Applied Soft Computing* **9**(2), 618–624 (2009).

[20] Vasicek, Z. and Sekanina, L., "Formal verification of candidate solutions for post-synthesis evolutionary optimization in evolvable hardware," *Genetic Programming and Evolvable Machines – in print* **12**(2), 1–23 (2011).

[21] Higuchi, T., Iwata, M., Keymeulen, D., Sakanashi, H., Murakawa, M., Kajitani, I., Takahashi, E., Toda, K., Salami, M., Kajihara, N., and Otsu., N., "Real-World Applications of Analog and Digital Evolvable Hardware," *IEEE Transactions on Evolutionary Computation* **3**(3), 220–235 (1999).

[22] Sekanina, L., "Image Filter Design with Evolvable Hardware," in [*Applications of Evolutionary Computing – Proc. of the 4th Workshop on Evolutionary Computation in Image Analysis and Signal Processing EvoIASP'02*], *LNCS* **2279**, 255–266, Springer Verlag, Kinsale, Ireland (2002).

[23] Glette, K., Torresen, J., and Yasunaga, M., "An online EHW pattern recognition system applied to face image recognition," in [*Applications of Evolutinary Computing, EvoWorkshops 2007*], *LNCS* **4448**, 271–280, Springer (2007).

[24] Thompson, A., Layzell, P., and Zebulum, S., "Explorations in Design Space: Unconventional Electronics Design Through Artificial Evolution," *IEEE Transactions on Evolutionary Computation* **3**(3), 167–196 (1999).

[25] Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., and Higuchi, T., "Evolvable Hardware at Function Level," in [*Parallel Problem Solving from Nature PPSN IV*], *LNCS* **1141**, 62–71, Springer (1996).

[26] Torresen, J., "A Divide-and-Conquer Approach to Evolvable Hardware," in [*Proc. of the 2nd International Conference on Evolvable Systems: From Biology to Hardware ICES'98*], Sipper, M., Mange, D., and Perez-Uribe, A., eds., *LNCS* **1478**, 57–65, Springer, Lausanne, Switzerland (1998).

[27] Gordon, T. G. W. and Bentley, P. J., "Towards development in evolvable hardware," in [*Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*], 241–250, IEEE Computer Society Press, Washington D.C., US (2002).

[28] Walker, J. A. and Miller, J. F., "The Automatic Acquisition, Evolution and Re-use of Modules in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation* **12**(4), 397–417 (2008).

[29] Pecenka, T., Sekanina, L., and Kotasek, Z., "Evolution of Synthetic RTL Benchmark Circuits with Predefined Testability," *ACM Transactions on Design Automation of Electronic Systems* **13**(3), 1–21 (2008).

[30] Miller, J. F., Job, D., and Vassilev, V. K., "Principles in the Evolutionary Design of Digital Circuits – Part I," *Genetic Programming and Evolvable Machines* **1**(1), 8–35 (2000).

[31] Vasicek, Z., Zadnik, M., Sekanina, L., and Tobola, J., "On evolutionary synthesis of linear transforms in FPGA," in [*Proc. of the 8th Conference on Evolvable Systems: From Biology to Hardware*], *LNCS* **5216**, 141–152, Springer Verlag (2008).

[32] Miller, J. F. and Thomson, P., "Cartesian Genetic Programming," in [*Proc. of the 3rd European Conference on Genetic Programming EuroGP2000*], *LNCS* **1802**, 121–132, Springer (2000).

[33] Miller, J. F. and Smith, S. L., "Redundancy and Computational Efficiency in Cartesian Genetic Programming," *IEEE Transactions on Evolutionary Computation* **10**(2), 167–174 (2006).

[34] Kalganova, T. and Miller, J. F., "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness," in [*The First NASA/DoD Workshop on Evolvable Hardware*], 54–63, IEEE Computer Society, Pasadena, California (1999).

[35] Cong, J. and Minkovich, K., "Optimality Study of Logic Synthesis for LUT-Based FPGAs," *IEEE Transactions on Computer-aided Design of Integrated Circuits and Systems* **26**(2), 230–239 (2007).

[36] Fiser, P. and Schmidt, J., "Small but nasty logic synthesis examples," in [*Proc. 8th Int. Workshop on Boolean Problems*], 183–190 (2008).

[37] Een, N. and Sorensson, N., "MiniSAT web pages," (2010).

[38] Vasicek, Z. and Sekanina, L., "A global postsynthesis optimization method for combinational circuits," in [*Proc. of the Design, Automation and Test in Europe, DATE*], 1525–1528, IEEE Computer Society (2011).

[39] Vasicek, Z. and Sekanina, L., "An area-efficient alternative to adaptive median filtering in fpgas," in [*Proc. of 2007 International Conference on Field Programmable Logic and Applications*], 216–221, IEEE Computer Society (2007).

[40] Sekanina, L., "Virtual reconfigurable circuits for real-world applications of evolvable hardware," in [*Evolvable Systems: From Biology to Hardware, Fifth International Conference, ICES 2003*], *LNCS* **2606**, 186–197, Springer (2003).

[41] Upegui, A. and Sanchez, E., "Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs," in [*The 1st NASA/ESA Conference on Adaptive Hardware and Systems*], 153–160, IEEE Computer Society, Los Alamitos, CA, USA (2006).

[42] Cancare, F., Santambrogio, M., and Sciuto, D., "A direct bitstream manipulation approach for virtex4-based evolvable systems," in [*Proceedings of 2010 IEEE International Symposium on of Circuits and Systems (ISCAS)*], 853–856, IEEE Computer Society (2010).

[43] Salvador, R., Otero, A., Mora, J., de la Torre, E., Riesgo, T., and Sekanina, L., "Evolvable 2d computing matrix model for intrinsic evolution in commercial fpgas with native reconfiguration support – submitted," in [*The NASA/ESA Conference on Adaptive Hardware and Systems*], 1–8, IEEE Computer Society (2011).

[44] Vasicek, Z. and Sekanina, L., "Hardware accelerator of cartesian genetic programming with multiple fitness units," *Computing and Informatics* **29**(6), 1359–1371 (2010).