Evolvable Hardware: From Applications to Implications for the Theory of Computation

Lukáš Sekanina

Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz

Abstract. The paper surveys the fundamental principles of evolvable hardware, introduces main problems of the field and briefly describes the most successful applications. Although evolvable hardware is typically interpreted from the point of view of electrical engineering, the paper discusses the implications of evolvable hardware for the theory of computation. In particular, it is shown that it is not always possible to understand the evolved system as a computing mechanism if the evolution is conducted with real hardware in a loop. Moreover, it is impossible to describe a continuously evolving system using the computational scenario of a standard Turing machine.

1 Introduction

Evolutionary algorithms (EAs) are population-based search algorithms that have been successfully applied to solve hard optimization problems in many application domains [1]. In the recent years, EAs have been also utilized in the area of engineering design [2].

In the field of *evolvable hardware*, EA is used to generate configurations for reconfigurable chips that can dynamically alter the functionality and physical connections of their circuits [3, 4]. Research in the field of evolvable hardware can be split into the two related areas of *evolutionary hardware design* and *adaptive hardware*. 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 characteristics in order to allow them to operate successfully in a changing environment or under presence of faults.

The recent years of development of this field can be characterized as a continuous search for promising problems/applications from the point of view of evolutionary design. In this paper, we will survey the fundamental principles of evolvable hardware, introduce main problems of the field and briefly describe the most successful applications.

The second part of the paper (Sections 3 and 4) is more theoretically oriented. Although evolvable hardware is typically interpreted from the point of view of electrical engineering, we will discuss the implications of evolvable hardware for the theory of computation. There are two significant features from the perspective of computer science: (1) Resulting systems are not designed conventionally; they are created in the process of evolution which is based on the 'generate-and-test' approach. We will show that it is not always possible to understand the evolved system as a computing mechanism (in the sense of Turing machines) if the evolution is conducted in a physical hardware. (2) It is impossible to describe a continuously evolving system in terms of Turing machine if EA is used to dynamically adapt the system in a changing environment. Then, evolvable machines can be characterized as super-computing systems.

2 Evolvable Hardware

This section presents the concept of evolvable hardware, introduces relevant terminology, discusses open problems and briefly surveys some important applications.

2.1 The Method

Figure 1 shows the basic principle of the evolvable hardware method: electronic circuits that are encoded as bit strings (chromosomes, in the parlance of EAs) are constructed and optimized by the evolutionary algorithm in order to obtain the circuit implementation satisfying the specification given by designer. In order to evaluate the candidate circuit, the new configuration of a reconfigurable device is created on the basis of the chromosome content. This configuration is uploaded into the reconfigurable device and evaluated for a chosen set of input stimuli. The fitness function, which reflects the problem specification, can include behavioral as well as non-behavioral requirements. For example, the correct functionality is a typical behavioral requirement. As a non-behavioral requirement, we can mention the requirement for minimum power consumption or minimum area occupied on the chip. Once the evaluation of the population of candidate circuits is complete, a new population can be produced. That is typically performed by applying the genetic operators (such as mutation and crossover) on existing circuit configurations. High-scored candidate circuits have got a higher probability that their genetic material (parts of configuration bitstreams) will be selected for next generations. The process of evolution is terminated when a perfect solution is obtained or when a certain number of generations is evaluated.

As the EA is a stochastic algorithm, the quality of resultant circuits is not guaranteed at the end of evolution. However, the method has two important advantages: (1) Artificial evolution can in principle produce intrinsic designs for electronic circuits which lie outside the scope of circuits achievable by conventional design methods. (2) The challenge of conventional design is replaced by that of designing an evolutionary algorithm that automatically performs the design in a target place (e.g., in space). This may be harder than doing the design directly, but makes autonomy possible.

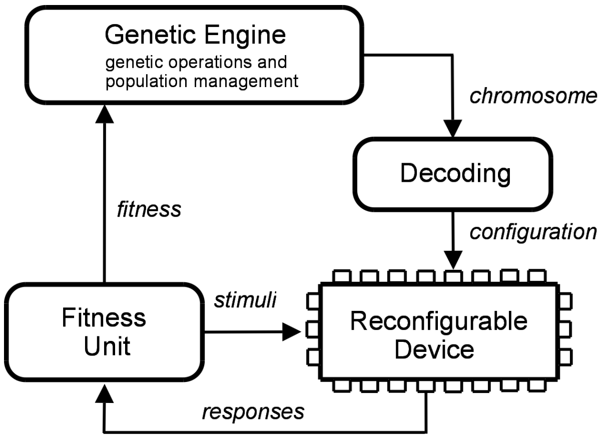

Fig. 1. High-level description of the evolvable hardware approach

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 [5].

2.2 Extrinsic and Intrinsic Evolution

Most reconfigurable circuits consist of configurable blocks whose functions and interconnections are controlled by the configuration bitstream. On the position of the reconfigurable circuit we can find various reconfigurable device, including Field Programmable Gate Arrays (FPGAs), Field Programmable Analog Arrays (FPAAs), Field Programmable Transistor Arrays (FPTAs) and special application-specific circuits (RISA [6], PAMA [7], Evolvable Motherboard [8], REPOMO32 [9] etc.). More exotic devices include reconfigurable mirrors [10], reconfigurable nanosystems [11], reconfigurable antennas [12] and reconfigurable liquid crystals [13].

If all candidate circuits are evaluated in a physical reconfigurable device, the approach is called *intrinsic evolution*. If the evolution is performed using a circuit simulator and only the resulting circuit is uploaded to a reconfigurable device the approach is called *extrinsic evolution*. Why is it important to distinguish between these approaches?

In 1996 and the following years, Adrian Thompson performed a series of experiments which clearly demonstrated that there can be a significant difference in the resulting behavior if candidate circuits are evaluated not in a circuit simulator but directly in a physical reconfigurable device [14]. Thompson used FPGA XC6216 chip to evolve a tone discriminator – a circuit discriminating between square waves of 1 kHz and 10 kHz. With 10 x 10 configurable blocks of an XC6216 device, the circuit should output 5V for one of the frequencies and 0V for the other. The problem is that evolved circuit has to discriminate between input periods five orders of magnitude longer than the propagation time of each configurable block. Thompson evolved a surprising solution: Only 21 out of 10x10 blocks contributed to the actual circuit behavior. However, some unconnected blocks also influenced the circuit behavior. According to Thompson, these blocks interacted with the circuit in some non-standard ways. Evolved circuit was carefully analyzed using simulators, tested under different conditions and on other XC6216 chips [15]. Surprisingly, Thompson was not able to create a reliable simulation model of the evolved circuit. In addition, he observed that the circuit configuration works only with the chip used during evolution. Although the circuit has a digital interface its internal behavior is typical for analog circuits. It can be stated that the evolution was able to explore a full repertoire of behaviors available from the silicon resources provided to create the required behavior. This is an outstanding result, practically unreachable by means of conventional design methods.

Similar results were observed by those doing evolutionary design in FPTA [16], liquid crystals [13] and some other unconventional platforms.

2.3 Scalability Problems

The scalability problem has been identified as the most important problem from the point of view of practical applicability of evolvable hardware. The *scalability of representation* means 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, the four main techniques have been developed:

– *Functional-level evolution* [17]: Instead of gates and single-wire connections, the system is composed of complex application-specific functional blocks (such as adders, multipliers and comparators) connected using multi-bit connections. The selection of suitable functional blocks represents a domain knowledge that has to be included into the system.
– *Incremental evolution* [18, 19]: Target circuit is decomposed onto modules which are evolved separately. The decomposition strategy is a kind of domain knowledge which has to be supplied by designer.
– *Development* [20, 21]: The above mentioned approaches employ a direct encoding of target circuit (phenotype) in the chromosome (genotype). Hence the size of the chromosome is proportional to the size of the circuit. Developmental approaches utilize indirect (generative) encodings which specify how to construct the target circuit. The phenotype is, in fact, constructed by a program which is encoded in the genotype. Designing these developmental encodings is not trivial and represents a domain knowledge which has to be supplied by designer.

- *Modularization* [22]: Some EAs enable to dynamically create and destroy reusable modules (subcircuits). The reuse of modules make the evolution easier even for large circuits.

Another problem is related to the fitness calculation time. 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). Hence, the evaluation time becomes the main bottleneck of the evolutionary approach when complex circuits with many inputs are evolved. This problem is known as the problem of *scalability of evaluation.* In order to reduce the time of evaluation, various techniques have been adopted:

- Only a subset of all possible input vectors is utilized. That is typical for evolution of filters, classifiers or robot controllers [23]. Evolved circuits have to be validated at the end of evolution using a test set — a representative set of input vectors which differs from the training set.
- In some cases it is sufficient to evaluate only some structural properties (not the function) of candidate circuits which can be done with a reasonable time overhead. For example, because the testability of a candidate circuit can be calculated with the $O(n^2)$ complexity, very large benchmark circuits (more than 1 million gates) have been evolved with predefined testability properties [24].
- In case that the target system is linear, it is possible to perfectly evaluate a candidate circuit using a single input vector independently of the circuit complexity. Multiple-constant multipliers composed of adders, subtractors and shifters were evolved for tens of multibit outputs [25].

An obvious conclusion is that the perfect evaluation procedures are applicable only for small circuits or in very specific cases of large circuits. On the other hand, when more complex circuits have to be evolved, only an imperfect fitness calculation method may be employed due to time constraints.

2.4 Applications

The applications of evolvable hardware fall into two categories: evolutionary design and adaptive hardware. We will briefly survey them.

Evolutionary Hardware Design Innovative designs were presented in the areas of small combinational circuits (multipliers [26]), digital filters [27], image operators [28], classifiers [29], diagnostics benchmark circuits [24] and many others.

Innovative implementations of analog circuits, antennas and optical systems were mostly obtained using indirect encodings, for example, *developmental genetic programming* introduced by John Koza [21, 30]. Koza's team has clearly illustrated that genetic programming can automatically produce a circuit that is competitive with human performance. In particular, genetic programming has created patentable new inventions and created circuits that either infringe or duplicate the functionality of previously patented inventions.

Adaptive Hardware Examples of adaptive hardware systems can be presented according to the types of changes which usually lead to the requirement for system adaptation. These changes can be classified either as changes

- in the hardware platform itself (i.e., faults),
- in the input data characteristics, or
- in the specification.

A promising application for evolution of analog circuits is the adaptation in extreme environments. JPL's evolvable hardware group has demonstrated for simple circuits created in the FPTA-2 that the evolutionary approach can recover the functionality lost (1) after a fault artificially injected into FPTA-2, (2) in extreme high and low temperatures (from –196°C to 320°C) and (3) in high radiation environments (up to 250 krad of total ionizing dose) [31, 32, 16, 33]. For example, the Self-Reconfigurable Analog Array (SRAA) developed at JPL provides the capability of *continual temperature compensation* using evolutionary-oriented reconfiguration [34].

Adaptive image compression methods allow for modifying the compression algorithm according to the particular data which has to be compressed. In consequence, a higher compression ratio can be achieved. A chip for adaptive lossless data compression was presented in [35]. Another evolvable hardware chip was designed to control a myoelectric prosthetic hand which processes the signals generated with muscular movements (electromyography, EMG signals). It takes a long time, usually almost one month, before a disabled person is able to control a multifunction prosthetic hand freely using a fixed controller. The evolvable chip allows the myoelectric hand to adapt itself to the disabled person and thus significantly reduce the training period to several minutes [36].

A post-fabrication calibration of ASICs can be also considered as a kind of hardware adaptation. Fabrication variances in the manufacture of chips can cause problems for some high-performance applications, especially when a cutting-edge manufacturing technology is employed. In other words, only some of the fabricated chips really meet the specification. Hence an EA is used to tune selected circuit parameters with the aim of obtaining required circuit function even for the chips which do not work because of variance of the fabrication process. Although some area of the chip has to be spent for implementing the reconfiguration logic and controller, the overall benefits are significant: It is possible to fabricate simpler, smaller and low-power designs which do not contain large circuits needed to compensate fabrication variances. A typical chip in which the post-fabrication tuning was successfully applied is an intermediate filter (IF), commonly found in mobile telephones [37].

Off line as well as on line evolution if often applied in the area of evolutionary robotics to design a reactive robot controller [38]. In more sophisticated scenario, robot controllers are evolved together with robot physical bodies [39].

3 Are Evolved Systems Computing Mechanisms?

A common feature of the conventional design process and evolutionary design is that target system's input/output behavior (and interpretation) has to be defined in advance. In the case of evolutionary design, this interpretation is used to define the fitness function. However, the main difference between the two design approaches is that while the conventional design is based on a systematic transformation of the abstract problem model into the implementation in a given platform, the evolutionary approach is based on the 'generate-and-test' method. When not constrained, the evolution can utilize all the resources available, including normally unused characteristics of the reconfigurable platform and environment to build the target system. Although the evolution is often able to find an implementation perfectly satisfying the specification, we can have problems to understand how and why the solution works. Therefore, nothing is known about the mapping between an abstract computational model and its physical implementation [40]. The next paragraph will demonstrate this phenomenon.

Assume that a simple non-trivial circuit performing required transformation has been evolved in an FPTA, liquid crystals or another 'physically-rich' platform. Moreover, assume that the environment is stable (i.e., temperature, radiation and all the phenomena that could influence its behavior are stable). In principle, we are able to interpret the signals at the inputs and outputs because we had to declare our interpretation in the fitness evaluation process (i.e., in advance, before the circuit was evolved). However, we don't exactly understand at the end of evolution how the platform performs the required computation. We have specified only the required input/output relation in the fitness function, i.e. we have not introduced any abstract model indicating the way of computation.

Does the evolved system really perform a computation? Since we can interpret the input/output behavior of the circuit as computation we can agree that the system really computes. However, there are problems with the interpretation of internal behavior of the evolved system. Copeland, Johnson and others would like to see that the symbols within the system have a consistent interpretation throughout the computation and that the interpretation is specified beforehand [41, 42]. It makes no sense to establish the mapping before the evolution is executed because the evolution can use various physical properties of the physical device to implement the required behavior. On the other hand, establishing the mapping at the end of evolution could solve the problem because (under some assumptions, such as that the environment does not influence the physical device at all) it could be possible to identify the physical states that correspond to the abstract computational states. Unfortunately, the discovery of the computational states/functions within the physical system is not quite sure. Although we could apply all the physical theories we currently know to analyze and explain the evolved system, we could obtain no satisfactory answer. As Bartles et al. have shown [43], the evolution is able to utilize those physical behaviors that are physically impossible from the point of view of current physical theories.

If we agree that (1) it does not matter how the system realizes a function as long as the system is digital from the outside (a digital system can be treated as a "black box" with digital inputs and outputs), (2) we are able to interpret its input/output behavior as computation and (3) this interpretation is defined in advance then *the evolved system is computational system* (computing mechanism). On the other hand, if the correspondence between the abstract states and physical states is crucial to qualify the system as computational one then *the evolved system is not a computing mechanism.*

Table 1 compares ordinary computers, evolved computational systems and the brain (the brain represents here all biological systems that are often studied as computational devices). We can observe that the evolved computational devices represent a distinctive class of devices that exhibits a specific combination of properties, not visible in the scope of all computational devices up till now.

Table 1. Comparison of different types of computational devices

Property	Processor	The brain	Evolved device
I/O behavior can be interpreted as computing	Yes	Yes	Yes
Device is a computing mechanism	Yes	Unsure	Unsure
An abstract model exists before implementation	Yes	No	No
The required behavior is specified beforehand	Yes	No	Yes
Engineers can design&build	Yes	No	Yes

4 Evolvable Computational Machines as Super-Turing Machines

In this paragraph, we will investigate computational properties of evolvable systems that can be adapted in a dynamically changing environment. In order to provide a simple formal framework for machine evolution in a dynamic environment, a mathematical model of the evolvable computational machine will be defined (according to [28]) as a quadruple $M = (E, \mathcal{M}, g, f)$, where

- E denotes a search (evolutionary) algorithm. Note that classical evolutionary algorithms utilize the fitness function of the form $\Phi : \mathcal{C} \rightarrow \mathbb{R}$ where $\mathcal{C}$ denotes a set of chromosomes.
- $\mathcal{M}$ denotes a set of machines which can be constructed from chromosomes for a given problem domain using
- *genotype-phenotype mapping* $g : \mathcal{C} \rightarrow \mathcal{M}$.
- f is a 'machine' fitness function $f : \mathcal{M} \rightarrow \mathbb{R}$ which is used to evaluate candidate machines. Φ is expressed as composition $\Phi = f \circ g$.

Note that f is a problem specific function, g can cover any type of constructional process and $\mathcal{M}$ is defined implicitly or explicitly before the evolution is executed.

In order to model evolvable machine M in a dynamic environment, the following specification of *machine context* (environment) is proposed according to

[28]. The machine context is considered as a set of (machine) fitness functions (we can call them *context functions*) together with a mechanism of transition among them. Context functions are changed in discrete time points modeled as natural numbers $\mathbb{N} = \{1, 2, \ldots\}$. A set of all mappings from $\mathcal{M}$ into $\mathbb{R}$ will be denoted $\mathbb{R}^{\mathcal{M}}$. Formally, machine context is defined in terms:

– $\mathit{\Gamma} \subseteq \mathbb{R}^{\mathcal{M}}$ – a set of context functions ($\varphi_i \in \mathit{\Gamma}$ specifies the fitness function in environment i).

– $\varphi_0 \in \mathit{\Gamma}$ – an initial context function.

– $\varepsilon : \mathit{\Gamma} \times \mathbb{N} \rightarrow \mathit{\Gamma}$ – a relation that determines a successive context function.

The following example illustrates the proposed formal definitions: Consider a real-time adaptive and reactive robot controller implemented as evolvable machine (as evolvable hardware in practice). The controller controls the motors of a flying robot which is deployed to monitor Mars surface. The goal is to maximize the measurement accuracy and minimize robot's power consumption. The quality of measurement is proportional to the number of active sensors, i.e. to the power consumption. It is assumed that the robot's operational time is endless. The system exhibits three important properties:

– The computation is reactive because the robot is controlled according to the data coming from sensors. The robot also influences its environment (e.g., other robots).
– The control algorithm is dynamically changed. The changes are unpredictable because the stimuli from environment (e.g., the occurrence of a meteorite) or hardware failures are uncomputable.
– It is not assumed that the computation will be terminated. In reality, the computation will terminate because of unavoidable hardware failures or insufficient power supply.

The evolvable machine, in fact, produces a (potentially endless) sequence of controllers (i.e., machines from $\mathcal{M}$), one for the most recent environment/specification. The type of environment is reflected in the context function φ_i. The change of environment (i.e., the change of context function described by ε) is not computable.

It is evident that proposed robot does not share the computational scenario of a standard Turing machine and hence it can not be simulated on Turing machine. Nevertheless, van Leeuwen and Wiedermann have shown that such computations may be realized by an *interactive Turing machines with advice* [44]. The interactive Turing machine with advice is a classical Turing machine endowed with three important features: advice function (a weaker type of oracle), interaction and infinity of operation. The same authors have proposed that *any (non-uniform interactive) computation can be described in terms of interactive Turing machines with advice.*

We can observe that evolvable computational machines (theoretical models!) operating in a dynamic environment show simultaneous non-uniformity of computation, interaction with an environment, and infinity of operations. Furthermore, relation ε is in general *uncomputable*. It was proven that computational power of an evolvable computational machine operating in a dynamic environment is equivalent with the computational power of an interactive Turing machine with advice, however, only in the case that evolutionary algorithm is efficient [28].

Also physical implementations of evolvable machines, Internet and other similar devices are very interesting from a computational viewpoint. At each time point they have a *finite* description. However, when one observes their computation in time, they represent *infinite* sequences of reactive devices computing non-uniformly. The "evolution" of machine's behavior is supposed to be endless. In fact it means that they offer an example of real devices (physical implementations!) that can perform computations that no single Turing machine (without oracle) can.

5 Conclusions

In this paper, we have surveyed the fundamental principles of evolvable hardware, introduced main problems of the field and mentioned the most successful applications. Proposed theoretical analysis of evolvable hardware which was conducted from the perspective of computer science should help in understanding of what we can obtain from a design method that utilizes the evolutionary design instead of conventional design techniques. Finally, we have shown that adaptive hardware does not reflect the computational scenario of a standard Turing machine.

Acknowledgements

This research was partially supported by the Czech Science Foundation under No. 102/07/0850 *Design and hardware implementation of a patent-invention machine* and the Research Plan No. MSM 0021630528 *Security-Oriented Research in Information Technology.*

References

[1] Eiben, A.E., Smith, J.E.: Introduction to Evolutionary Computing. Springer, Berlin (2003)
[2] Bentley, P.J.: Evolutionary Design by Computers. Morgan Kaufmann, San Francisco CA (1999)
[3] 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: Proc. of the 2nd International Conference on Simulated Adaptive Behaviour, MIT Press (1993) 417–424

[4] de Garis, H.: Evolvable hardware – genetic programming of a darwin. In: International Conference on Artificial Neural Networks and Genetic Algorithms, Innsbruck, Austria, Springer Verlag (1993)

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

[6] Greensted, A., Tyrrell, A.: RISA: A hardware platform for evolutionary design. In: Proceedings of 2007 IEEE Workshop on Evolvable and Adaptive Hardware, IEEE (2007) 1–7

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

[8] Layzell, P.J.: A new research tool for intrinsic hardware evolution. In: Proc. of the Evolvable Systems: From Biology to Hardware Conference. Volume 1478 of LNCS., Springer (1998) 47–56

[9] Sekanina, L., Ruzicka, R., Vasicek, Z., Prokop, R., Fujcik, L.: Repomo32 – new reconfigurable polymorphic integrated circuit for adaptive hardware. In: Proceedings of 2009 IEEE Workshop on Evolvable and Adaptive Hardware, IEEE CIS (2009) 39–46

[10] Loktev, M., Soloviev, O., Vdovin, G.: Adaptive Optics – Product Guide. OKO Technologies, Delft (2003)

[11] Tour, J.M.: Molecular Electronics. World Scientific (2003)

[12] Linden, D.S.: A system for evolving antennas in-situ. In: EH '01: Proceedings of the The 3rd NASA/DoD Workshop on Evolvable Hardware, Washington, DC, USA, IEEE Computer Society (2001) 249–255

[13] Harding, S.L., Miller, J.F., Rietman, E.A.: Evolution in materio: Exploiting the physics of materials for computation. Journal of Unconventional Computing **4**(2) (2008) 155–194

[14] Thompson, A.: Silicon Evolution. In: Proc. of Genetic Programming GP'96, MIT Press (1996) 444–452

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

[16] Stoica, A., Zebulum, R.S., Keymeulen, D., Ferguson, M.I., Duong, V., Guo, X.: Evolvable hardware techniques for on-chip automated reconfiguration of programmable devices. Soft Computing **8**(5) (2004) 354–365

[17] Murakawa, M., Yoshizawa, S., Kajitani, I., Furuya, T., Iwata, M., Higuchi, T.: Evolvable Hardware at Function Level. In: Parallel Problem Solving from Nature PPSN IV. Volume 1141 of LNCS., Springer (1996) 62–71

[18] 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. Volume 1478 of LNCS., Lausanne, Switzerland, Springer (1998) 57–65

[19] Torresen, J.: A scalable approach to evolvable hardware. Genetic Programming and Evolvable Machines **3**(3) (2002) 259–282

[20] Kitano, H.: Morphogenesis for evolvable systems. In: Towards Evolvable Hardware: The Evolutionary Engineering Approach. Volume 1062 of LNCS., Berlin, Springer (1996) 99–117

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

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

[23] Higuchi, T., Liu, Y., Yao, X.: Evolvable hardware. Springer, Berlin (2006)

[24] Pecenka, T., Sekanina, L., Kotasek, Z.: Evolution of synthetic rtl benchmark circuits with predefined testability. ACM Transactions on Design Automation of Electronic Systems **13**(3) (2008) 1–21

[25] Vasicek, Z., Zadnik, M., Sekanina, L., Tobola, J.: On evolutionary synthesis of linear transforms in fpga. In: Proc. of the 8th Int. Conference on Evolvable Systems: From Biology to Hardware. Volume 5216 of LNCS., Berlin, Springer Verlag (2008) 141–152

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

[27] Hounsell, B.I., Arslan, T., Thomson, R.: Evolutionary design and adaptation of high performance digital filters within an embedded reconfigurable fault tolerant hardware platform. Soft Computing **8**(5) (2004) 307–317

[28] Sekanina, L.: Evolvable components: From Theory to Hardware Implementations. Natural Computing. Springer Verlag Berlin (2004)

[29] Glette, K., Torresen, J., Gruber, T., Sick, B., Kaufmann, P., Platzner, M.: Comparing evolvable hardware to conventional classifiers for electromyographic prosthetic hand control. In: Proc. of the 2008 NASA/ESA Conference on Adaptive Hardware and Systems, IEEE Computer Society (2008) 32–39

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

[31] Keymeulen, D., Zebulum, R., Jin, Y., Stoica, A.: Fault-tolerant evolvable hardware using field-programmable transistor arrays. IEEE Transactions on Reliability **49**(3) (2000) 305–316

[32] Stoica, A., Keymeulen, D., Zebulum, R.S., Guo, X.: Reconfigurable electronics for extreme environments. In Higuchi, T., Liu, Y., Yao, X., eds.: Evolvable Hardware, Springer (2006) 145–160

[33] Stoica, A., Wang, X., Keymeulen, D., Zebulum, R.S., Ferguson, M., Guo, X.: Characterization and Recovery of Deep Sub Micron (DSM) Technologies Behavior Under Radiation. In: 2005 IEEE Aerospace Conference, IEEE (2005) 1–9

[34] Stoica, A., Keymeulen, D., Zebulum, R.S., Katkoori, S., Fernando, P., Sankaran, H., Mojarradi, M., Daud, T.: Self-reconfigurable mixed-signal integrated circuits architecture comprising a field programmable analog array and a general purpose genetic algorithm ip core. In: Evolvable Systems: From Biology to Hardware, 8th International Conference, ICES 2008. Volume 5216 of LNCS., Springer (2008) 225–236

[35] Sakanashi, H., Iwata, M., Higuchi, T.: A lossless compression method for halftone images using evolvable hardware. In: Evolvable Systems: From Biology to Hardware, 4th International Conference, ICES 2001 Tokyo, Japan. Volume 2210 of LNCS., Springer (2001) 314–326

[36] Kajitani, I., Iwata, M., Higuchi, T.: A ga hardware engine and its applications. In Higuchi, T., Liu, Y., Yao, X., eds.: Evolvable Hardware, Springer (2006) 41–63

[37] Murakawa, M., Yoshizawa, S., Adachi, T., Suzuki, S., Takasuka, K., Iwata, M., Higuchi, T.: Analogue ehw chip for intermediate frequency filters. In: Evolvable Systems: From Biology to Hardware, Second International Conference, ICES 98, Lausanne, Switzerland. Volume 1478 of LNCS., Springer (1998) 134–143

[38] Nofli, S., Floreano, D.: Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines. MIT Press/Bradford Books, Cambridge, MA (2000)

[39] Lipson, H., Pollack, J.B.: Automatic design and manufacture of robotic lifeforms. Nature **406** (2000) 974–978

[40] Sekanina, L.: Evolved computing devices and the implementation problem. Minds and Machines **17**(3) (2007) 311–329

[41] Copeland, B.J.: What is computation? Synthese **108** (1996) 335–359

[42] Johnson, C.G.: What kinds of natural processes can be regarded as computations? In Paton, R., ed.: Computation in Cells and Tissues: Perspectives and Tools of Thought. Springer (2004) 327–336

[43] Bartels, R.A., Murnane, M.M., Kapteyn, H.C., Christov, I., Rabitz, H.: Learning from Learning Algorithms: Applications to attosecond dynamics of high-harmonic generation. Physical Review A **70**(4) (2004) 1–5

[44] van Leeuwen, J., Wiedermann, J.: The Turing Machine Paradigm in Contemporary Computing. In: Mathematics Unlimited - 2001 and Beyond, Springer (2001) 1139–1155