# Theory and Applications of Evolvable Embedded Systems

Lukáš Sekanina and Vladimír Drábek
Faculty of Information Technology, Brno University of Technology
Božetěchova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz drabek@fit.vutbr.cz

## Abstract

*This paper represents the first attempt to formulate a concept of the evolvable embedded system as a specialized kind of evolvable (hardware) systems. The paper defines the class of evolvable embedded systems, describes a general framework for their modeling, introduces theoretical models, and reviews possible implementations and applications. As a typical example, evolutionary functional recovery of damaged median circuits is considered and simulated.*

## 1. Introduction

Embedded systems—in addition to their expected higher performance, lower cost and better dependability—will have to exhibit various new features in future, including adaptability at the first place. We can imagine that forthcoming embedded systems will be able to *autonomously* modify the function (at the level of hardware) that they perform, repair themselves in case of a faulty event or reduce energy consumption if needed. As genetic programming and evolvable hardware have shown in the recent years, the evolutionary approach is probably the most competitive method to perform this task [13].

For the sake of clarity we have to mention two conceptual points. First, this paper primarily deals with evolvability and adaptation conducted directly at the hardware level. Second, evolvability is not considered as the ability to make system upgrades easily and consistently (as it is usual in the software domain in order to achieve code or service reuse [16]). Here the evolvability is considered as the ability of a system to produce totally new solutions to a changing environment (specification) autonomously.

This paper represents the first attempt to formulate a concept of the *evolvable embedded system* as a specialized kind of evolvable (hardware) systems. We will define the class of evolvable embedded systems, introduce a general framework for their modeling, introduce theoretical models, and review possible implementations and applications.

Evolutionary algorithms enabled us designing *adaptive computational machines* [22]. In our case the evolutionary algorithm will be an inherent part of a target (i.e. embedded) system and will autonomously produce computational machines according to requirements represented via a dynamic fitness function, which reflects a changing environment. We will also be interested in applications in which the problem specification remains *formally* unchanged but properties of a physical platform can be changed over time, for example, because of the changes in temperature or radiation. We will classify all these situations as a dynamic environment.

The primary focus of the paper will be on the field of evolvable hardware (EHW) [10]. This emerging field exists at the intersection of electronic engineering, computer science and biology. The benefits of evolvable hardware are particularly suited to a number of applications, including the design of low cost hardware, creation of adaptive systems, fault tolerant systems and innovations.

As an example, we will propose a case study in which close-to-perfect median circuits will be evolved on a platform that is partially damaged by an environment.

The paper is organized as follows. Section 2 determines the class of evolvable embedded systems. In Section 3 genetic programming and evolvable hardware are introduced and considered as techniques to ensure adaptation. Section 4 describes the general architecture of evolvable embedded systems. Some comments on theory will be given in Section 5. While Section 6 deals with the integration of evolvable subsystems into target applications, Section 7 defines four basic classes of these applications. As a typical example, functional recovery of damaged median circuits is described in Section 8. Discussion and conclusions are given in Sections 9 and 10, respectively.

## 2. Towards Evolvable Embedded Systems

Current typical embedded systems are based on microcontrollers [34] or combine processors (DSP) with programmable logic (such as FPGAs—field programmable gate arrays).

Proceedings of the 11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems (ECBS'04)
0-7695-2125-8/04 $ 20.00 © 2004 IEEE

The most important areas of application are communications and networking, computationally intensive algorithms, signal processing, control, multimedia and entertainment. These systems can be characterized as: reactive, real-time and autonomous.

Specialized Systems on a Chip (SoC) have been developed to integrate the most important components on a single chip (see, e.g. [20]). They usually represent either cheaper alternatives to universal personal computers in many industrial applications or high-performance systems unrealizable by means of personal computers.

The next improvent was done by using *reconfigurable computing* in which various hardware modules can be stored as configurations in a library and dynamically uploaded/removed to/from an FPGA. Optimal combinations of these modules are sought for any moment [14]. This approach leads to the effective usage of hardware resources (because more computation can be performed in hardware than physical hardware resources enable) and so to a higher performance. The usage of FPGAs has allowed designers to reconfigure physical hardware of embedded systems during the operational time.

In addition to the reconfiguration strategy, emerging hardware/software co-design problem has to be solved by designers. Reconfigurable computing in fact provides a limited sort of *adaptability*—it is possible to change the system function by means of reconfiguration if a requirement emerges. "All chips will be reconfigurable" (Mike Butts' Keynote Speech at the Field Programmable Logic 2003 conference) is becoming a generally accepted idea, as a way to tackle the complexity of system design.

Having the basic knowledge of embedded systems we can formulate our definition of evolvable embedded systems: An *evolvable embedded system* is such a reconfigurable embedded system in which an evolutionary algorithm is utilized to dynamically modify some of system (software and/or hardware) components in order to adapt the behavior of the system to a changing environment.

## 3. Adaptation Through Genetic Learning

The majority of the research in the field of evolutionary algorithms is devoted to problems with a single (static) fitness function. However, real-world applications (like robotics, evolvable hardware, scheduling, etc.) operate in a *dynamic* environment (i.e. with a time-varying specification of the fitness function) [3, 18]. The two basic methods are used to deal with the changing environment:

- The designer tries to speed up a single evolution as much as possible (e.g. through the hardware implementation). The evolution is restarted completely when a change is detected.
- The evolution is not usually restarted in a new environment. Rather, information gathered during the past generations is employed under a new environment, since it is supposed that the change is not substantial. Then the system could exhibit a prompt adaptation to the time-varying environment.

Instead of evolutionary optimization, we are interested in creative evolutionary design in this paper [1]. Among others, evolvable hardware and genetic programming fall into this category.

Genetic programming was developed to allow automatic programming and program induction [13]. It may be viewed as a specialized form of genetic algorithm, which manipulates with variable length chromosomes (i.e. with a specialized representation) using modified genetic operators.

Evolvable hardware is a computer-based system that autonomously changes its physical electronic circuits (and so its function) according to requirements of the environment [10]. Nowadays, it is not difficult to modify electronic circuits, even of a running computer, since these circuits are often implemented using reconfigurable devices. In case of evolvable hardware, configurations are designed automatically by an evolutionary algorithm. Up to now, evolvable hardware was successfully applied to design a number of unique digital as well as analog circuits. The evolved circuits exhibit the properties that we have never reached by means of traditional engineering methods [17, 22, 31]. Furthermore, online evolution has allowed us to implement high-performance and adaptive systems for the applications in which the problem specification is unknown beforehand and can vary in time [9, 18].

Because of inherent redundancy in reconfigurable devices, the evolvable hardware is inherently fault tolerant. It means that in case of a failure of a circuit element, the evolutionary algorithm is usually able to recover the functionality using the remaining elements. If a critical number of elements are damaged, the functionality cannot be recovered and the chip "dies". Hence evolvable hardware is a method for automatic designing of adaptive as well as fault-tolerant (e.g. self-repairing) systems [31].

The following list gives examples of reconfigurable platforms for evolvable hardware: PAL (Programmable Analog Array), FPGA (Field Programmable Gate Array), FPAA (Field Programmable Analog Array), massively parallel processor array (e.g. picoChip PC102 [19]), FPTA (Field Programmable Transistor Array), specialized ASIC (Application Specific Integrated Circuits), nanodevices (e.g. NanoCell [33]), reconfigurable antennas, MEMS (microelectromechanical systems), reconfigurable optics and some other.

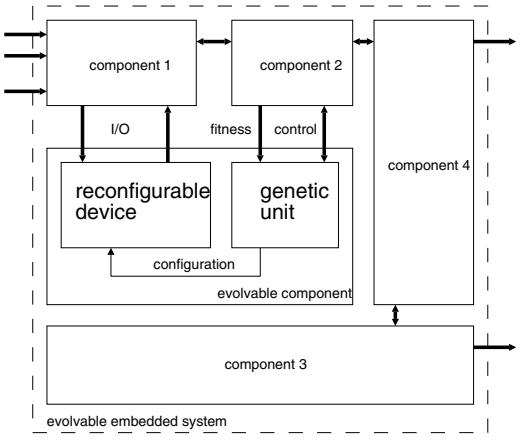


**Figure 1. Evolvable component placed in evolvable embedded system. Components 1–4 represent the environment for the evolvable component in this example.**

## 4. General Architecture of the Evolvable Embedded System

Embedded systems belong to modern computational systems. The computational scenario of these systems differs from traditional computational models such as Turing machines: instead of an algorithm which processes preprepared data, performs computations, and reports the result, modern computation is based on permanent and potentially endless *interactions* of a number of *components* [8, 35]. Furthermore, these components can change their functions as a consequence of permanent reconfigurations.

Stoica et al. mentioned four years ago that "the path leads to the IP (Intellectual Property) level and evolvable hardware solutions will become an integrated component in a variety of systems that will thus have an evolvable feature" [28]. Evolvable components were introduced in [22] as the components that can autonomously change their functionality according to requirements of an environment.

According to [22], the evolvable component consists of a reconfigurable device and a genetic unit. The genetic unit implements evolutionary algorithm; however, fitness calculation is performed outside the component, by environment, because only the environment "knows" the specification and thus only the environment can assign the fitness value to any circuit/program realized in the programmable device. Hence, with respect to definition of the evolvable embedded system proposed in Section 2, we can easily define the evolvable embedded system as an embedded system that contains at least one evolvable component.

Figure 1 shows a general architecture of an evolvable embedded system. Communication between the environment (that is represented by components 1–4 in our example) and the evolvable component is as follows: First the evolvable component is initialized (initial population is generated). Then the following sequence of operations is repeated endlessly. The environment requires a new circuit to be uploaded into the reconfigurable circuit. The component is to generate a new configuration and to configure the reconfigurable circuit. If there is no configuration (chromosome) in the chromosome memory available then a new population has to be generated autonomously. When the evaluation of the circuit behavior is finished, then a fitness value is sent to the evolvable component. The component is to store the fitness value and to wait for another request from the environment. The environment can also require uploading of the best circuit that has been evolved so far. Hence the component has to continually store the best configuration that has been evolved so far and to provide it when requested. In all cases the evolvable component encapsulates the reconfiguration process that is *invisible* from the external environment.

## 5. Theoretical Issues

As soon as we have introduced the component approach to evolvable embedded systems, we should start to think of a suitable formal description and a mathematical theory that could make understanding and designing these systems easier. Some standard formal models for (non-adaptive) embedded systems were summarized in [23].

In case of evolvable embedded systems it is possible to combine a theory of evolutionary algorithms (e.g. [25]), a theory of reconfigurable systems (e.g. [2]) and a theory of evolvable systems (e.g. [22]). However, all these theories have one point in common: they are weak for real-world applications. No results are available for these issues for practical problems.

### 5.1. Reconfigurable Computing

No general theoretical approach exists for modeling reconfigurable computing. However, various area-specific models have been proposed [4]. These models are utilized for developing the actual mapping and scheduling of algorithms onto the reconfigurable platform. The representation of the complete application is a task graph with functions as nodes and edges representing the precedence constraints. This sequence of tasks is mapped onto a sequence of configurations. The optimal sequence of configurations is sought.

The configurations required for executing a specific function have to be generated either at compile time or on-the-fly at runtime. Compile time configurations can be generated by using schematic techniques or CAD mapping

tools. Runtime configurations can be generated by using run-time parameterized circuits or by dynamic modification of the configuration information before loading the configuration.

For instance, Bondalapati and Prasanna have developed HySAM model which is a general model consisting of a von-Neumann style processor with additional reconfigurable device [2]. The model is able to calculate cost of changing the configuration. New configurations are generated using generators. These generators are defined formally and allow designers to derive next configurations and their delays. There are defined three basic generators: parallel, serial and pipelined. Recursive applications of these generators can be utilized to generate a large class of configurations from the basic configuration.

### 5.2. Evolvable Systems

Section 4 shows that in the case of evolvable computational machines, a machine which fulfils the objectives (formulated via the fitness function) is sought in a given set of machines by means of an evolutionary algorithm in which representation, genotype–phenotype mapping and genetic operators remain unchanged during the run (see Fig. 1). Similarly to Eberbach's results [7], it was proven that evolvable computational machines operating in a dynamic environment exhibit a super-Turing computational power [22].

We can observe that evolvable computational machines operating in a dynamic environment show simultaneous non-uniformity of computation, interaction with an environment, and infinity of operations. Furthermore, the point at time in which the fitness function (specification) is changed is in general *uncomputable*. For example, see the case study in Section 8.

This viewpoint corresponds with the recent results in an emerging field—hypercomputation (or super-Turing computation) [5, 8, 36, 35]—that some theoretical models and modern computational systems do not share the computational scenario of a standard Turing machine and hence they can not be simulated on Turing machines. For instance, see the *Driving Home from Work* problem presented in [8] that is uncomputable on a standard Turing machine, but computable in reality.

At each time point these evolvable devices 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 computation that no single Turing machine (without oracle) can.

We can conclude that the unpredictable behavior and inherent complexity get designers into troubles and hence the design process is more intuitive than rigorous.

## 6. System Integration

A problem is how to integrate an "evolvable" feature into contemporary and forthcoming embedded systems. For example: Mobile phones belong to classical examples of embedded systems. It is typically impossible to change their software/hardware during the operational time. However, it is assumed that new generations of mobile phones will integrate this feature [37].

In case of adaptive evolvable embedded systems, the evolutionary algorithm has to be integrated in the system during the operational time. We have to specify the *object* of evolution—to choose a subsystem (a circuit or a subprogram) that will continually be evolved in order to ensure high-performance data processing, adaptation to unpredictable events, fault-tolerance, etc. Finally, we have to specify the platform where genetic operations and fitness calculation will be carried out. Note that fitness calculation is usually the most time consuming part of the evolutionary design process.

We have introduced the following classes to characterize (adaptive) embedded systems.

- Class 0 (fixed software and hardware): Software as well as hardware are defined at the design time. Neither reconfiguration nor adaptation is performed. This class also contains the systems with reconfigurable FPGAs that are only configured during reset. A coffee machine could be a good example.
- Class 1 (reconfigurable SW/HW): Software or hardware (a configuration of an FPGA) is changed during the run in order to improve performance and the utilization of resources (e.g. in reconfigurable computing). Evolutionary algorithm can be used to schedule the sequence of configurations at the compile time, but not at the operational time.
- Class 2 (evolutionary optimization): Evolutionary algorithm is a part of the system. Only some coefficients in SW (some constants) or HW (e.g. register values) are evolved, i.e. limited adaptability is available. Fitness calculation and genetic operations are performed in software. Example: an adaptive filter—changing coefficients for a fixed structure of an FIR filter.
- Class 3a (evolution of programs): Entire programs are constructed using genetic programming in order to ensure adaptation or high-performance computation. Everything is performed in software [15].
- Class 3b (evolution of hardware modules): Entire hardware modules are evolved in order to ensure adaptation, high-performance computation, fault-tolerance

or low-energy consumption. Fitness calculation and genetic operations are carried out in software or using a specialized hardware. Reconfigurable hardware is configured using evolved configurations. The system typically consists of a DSP and a reconfigurable device. Example: NASA JPL SABLES [29].

- Class 4 (evolvable SoC): All components of class 3b are implemented on a single chip. It means that the SoC contains a reconfigurable device. Some of such devices have been commercialized up to now, for example, a data compression chip [30].

- Class 5 (evolvable IP cores): All components of class 3b are implemented as IP cores, i.e. at the level of HDL source code (Hardware Description Language). It requires describing the reconfigurable device at the HDL level as well. An approach—called the virtual reconfigurable circuit—has been introduced to deal with this problem [21]. Then the entire evolvable subsystem can be realized in a single FPGA.

- Class 6 (co-evolving components): The embedded system contains two or more co-evolving hardware or software devices. These co-evolving components could be implemented as multiprocessors on a SoC or as evolvable IP cores on an FPGA. No examples representing this class are available nowadays.

While classes 0 and 1 do not represent adaptive embedded systems, class 2 shows a certain kind of adaptation. Class 3 includes fully adaptive software (3a) and hardware (3b) systems realized using a common PC and/or standalone boards. SoC solutions (class 4) are typical for real-world industrial applications. Class 5 offers the reusability concept for evolvable embedded systems because the evolvable IP core is derived from the idea of reusable evolvable component. Class 6 represents rather directions for the future research than a contemporary paradigm.

## 7. Application Classes

In this section, potential applications of evolvable embedded systems will be reviewed. These applications include adaptive solutions for mobile systems, space systems, signal processing, energy management, sensors, controllers, scheduling and many others.

Real-world applications of evolvable hardware were classified in a number of papers, e.g. [32]. In this paper, we will classify the embedded systems evolving in a dynamic environment according to the four following sections ([22]).

### 7.1. Embedded Evolutionary Design

Although the fitness function (i.e. problem specification) is changed rarely, it is not (economically, physically, etc.) possible to remove the evolutionary algorithm from the system, perform the evolutionary design in a design laboratory and then upload the resulting configuration into the reconfigurable circuit. As an example, one can imagine a controller inside a prosthetic hand. An approach in which the controller is uniquely evolved (and so adapted) for any disabled person was developed to make the rehabilitation time shorter [11]. The usage of a traditional fixed controller leads to almost a one month rehabilitation time, while the evolvable controller is able to solve the problem in 10 minutes [9].

The aircraft recovery problem is a typical application from the software domain that belongs into this category. The problem involves decisions concerning aircraft to flight assignments in situations where unforeseen events have disrupted the existing flight schedule, e.g. bad weather causing flight delays [15].

Common features of the systems which belong to this category are as follows: (1) the evolution is terminated when a sufficient solution is found, (2) the evolution is triggered manually or remotely (for instance, through the Internet), (3) a system function can usually be interrupted when training is needed.

### 7.2. Self-adaptive Systems

In contrast to the previous category, the problem specification is often changed here. It means that there is stimulus providing information about a change, for instance, a system performance measure. Because the system has to operate continuously, a strategy with two physical reconfigurable circuits RC1 and RC2 can be utilized. The RC1 contains the best design evolved so far. The evolution of the RC2 configuration is executed forever, because it has to reflect the changes of fitness function (problem specification). When something interesting is evolved in the RC2, the RC2 sends its configuration to RC1. As examples we can mention a real-time adaptive image filtration [22] or a dynamic hashing function, which could be applied in a cache memory [6].

In another application, a microphone acquires a voice signal coming from radio in real-time (it is the reference signal) which is mixed with a noise signal and conditioned for the FPTA. FPTA is evolved to separate the two signals and hence it operates as an adaptive noise filter [26].

### 7.3. Self-triggered Evolution

In this category a data stream (which is processed using evolvable system in order to increase performance) typ-

ically represents the environment. Every new fitness function (specification) can be taken into account only if the processing of the previous one is finished. For instance, the image compression domain offers a challenging task. Evolution has to find such a predictor that is optimal for a given block of an image. An image compression system based on this idea was implemented and commercialized [30].

### 7.4. Online Evolution

Evolution of a robot controller could be understood as online evolution, since the robot finds its way by using interactions with the environment only (i.e. in a kind of unsupervised learning). Thus different controllers ("robots") in a population may experience different environmental conditions. It means that the data needed for the fitness calculation are collected from a real environment during the fitness calculation [18].

## 8. Case Study: Functional Recovery

This section presents an example of evolvable embedded system that is able to recover its function after a faulty event. Assume that the system consists of many components and one of them—an evolvable reconfigurable circuit—is responsible for calculation the median values from nine 8bit inputs, e.g. it operates as a filter. As Fig. 2 shows, the circuit is implemented using 24 MinMax elements (compare&swap units); the configuration bits determine their interconnection. The objective is to design as good median circuit as possible using resources available. It is assumed that the system operates in the environment that damages the elements. The reconfigurable circuit is configured using an evolutionary algorithm, which tries to put elements together to fulfill the objectives.

Thanks to the *zero-one principle* discovered for sorting networks [12], every candidate median circuit can be evaluated in $2^9 = 512$ steps representing the application of training vectors 000000000 – 111111111. We can imagine that such the test is not performed in fitness calculation only, but it is regularly performed to diagnose hardware. If the current circuit is not competent, the evolution is restarted to find a better solution. We simulated faulty events and tried to recover functionality of the circuit.

Candidate solutions are represented as sequences of pairs $(a, b)$ indicating that $a$ is compared/swapped with $b$. The evolved sequences are transformed onto configuration bitstreams of the reconfigurable circuit. The setting of the evolutionary algorithm is as follows. Initial population of 200 individuals is seeded randomly using alleles $0 - 8$. New individuals are generated using mutation (1 integer per chromosome). Four best individuals are considered as parents and every newly formed population consists of their clones.

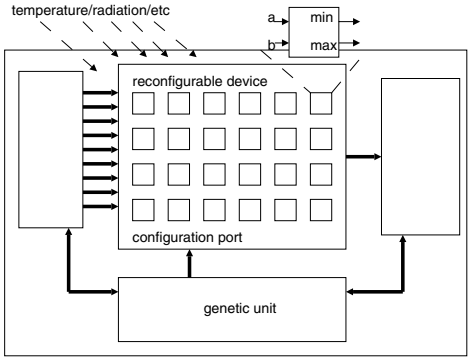


**Figure 2. Reconfigurable device consisting of MinMax elements placed in an embedded system.**

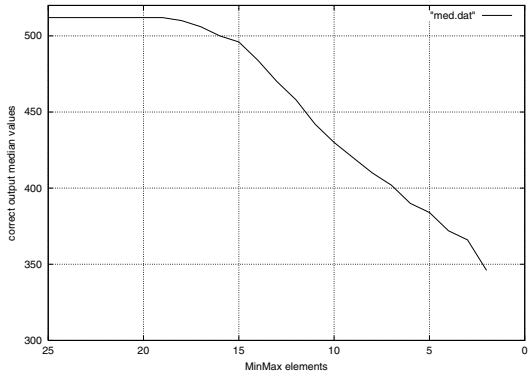


**Figure 3. Reducing the number of uncorrupted MinMax elements and its impact on calculation of correct output values for 9-median circuits.**

The evolutionary algorithm is left running until a fully correct individual is found or 3000 generations are exhausted. We also increase mutation rate if no improvement is observable during the last 30 generations.

Figure 3 shows how many output (median) values remain correct if the number of uncorrupted elements decreases. We can see that 19 elements still ensure the correct behavior. However, with decreasing resources the quality of the circuits gets down. It is interesting that mere two elements yield 346 correct output values, i.e. more than 50%!

For instance, the following sequence utilizing 19 elements was evolved as the perfect 9-median circuit: (7, 2), (5, 8), (4, 0), (6, 0), (6, 7), (7, 3), (5, 4), (5, 6), (7, 8), (1, 4), (7, 6), (2, 1), (2, 6), (8, 1), (4, 3), (0, 8), (6, 0), (4, 0), (6, 4). Here is an example of a 2-element circuit: (8, 6), (8,4).

The well-known conventional hardware implementation

of 9-median circuit has confirmed that we need at least 19 elements to perform the 9-median task [24]. As far as we know, no technique has been proposed in literature how to realize close-to-perfect median circuits if sufficient resources are not available. Our experiment has confirmed again that evolutionary techniques are able to produce novel designs in the situations in which conventional approaches fail. The evolution requires a few seconds on a common PC and hence it is suitable for real-time applications.

From the system integration viewpoint, the proposed application belongs to the class 3b at least. It could also be characterized as "Embedded evolutionary design" according to Section 7.1.

## 9. Discussion

It is hard to achieve online (real-time) adaptation. The difficulty is not caused by the evolvable system, but by the online requirement [38]. Most current approaches can only be considered as offline adaptations, where the adaptation happens during the learning phase. For instance, Stoica et al. have claimed that current lack of validation for online evolutionary systems means that critical spacecraft *control* systems cannot realistically be evolved online [27]. However, *sensors* and *sensory* information control systems are not critical. Hence systems utilized to capture, process and transmit such data are suitable for evolutionary design.

We argue that a *sufficient mean-long-time-performance* (quality) of the evolved solutions is required in adaptive embedded systems. These systems **must** outperform conventional (nonadaptive) approaches. If not, there is not any reason to apply them. This viewpoint determines the *class of applications* in which evolvable embedded systems can be applied.

Stoica has noted that the challenge of conventional design is replaced with that of designing an evolutionary process that automatically performs the design in our place. This may be harder than doing the design directly, but makes autonomy possible [26]. Nowadays it is difficult to evolve complex systems mainly because it is difficult to reduce the evaluation time of candidate circuits. Only behaviors required during conditions of fitness calculation are guaranteed. It is hard to predict behavior of a solution outside the domain in which it was evolved.

Although some evolvable SoCs have been presented [9, 11, 30], a typical evolvable system is implemented using a stand-alone board equipped with a reconfigurable circuit and a powerful personal computer where an evolutionary algorithm is carried out. If evolvable IP cores were dominated in future, the design process of evolvable hardware will be transformed to the software design process at the level of HDLs.

## 10. Conclusions

In this paper we surveyed a promising area of evolvable embedded systems. We focused our attention on theory, applications and hardware implementations of these systems using evolvable hardware. A classification for system integration has been developed that consists of six classes. In another classification, evolvable embedded systems have been divided into four groups according to the environment where they operate. An example of autonomous functional recovery has been implemented.

Because a general concept for modeling of evolvable embedded systems has been formulated by means evolvable components, our future research will be devoted to creating tools for the semi-automatic design of these systems on the basis of evolvable components.

**Acknowledgments:** The research was performed with the Grant Agency of the Czech Republic under No. 102/03/P004 *Evolvable hardware based application design methods* and the Research intention No. CEZ MSM 262200012 – *Research in information and control systems*.

## References

[1] P. Bentley. *Evolutionary Design by Computers*. Morgan Kaufmann Publishers, San Francisco, CA, 1999.

[2] K. Bondalapati and V. K. Prasanna. Reconfigurable Computing Systems. *Proc. of the IEEE*, 90(7):1201–1217, 2002.

[3] J. Branke. Evolutionary Approaches to Dynamic Optimization Problems – Update Survey. In *Proc. of the GECCO Workshop on Evolutionary Algorithms for Dynamic Optimization Problems*, pages 27–30, San Francisco, CA, 2001. Morgan Kaufmann.

[4] K. Compton and S. Hauck. Reconfigurable Computing: A Survey of Systems and Software. *ACM Computing Surveys*, 34(2):171–210, 2002.

[5] B. J. Copeland and R. Sylvan. Beyond the Universal Turing machine. *Australasian J. of Philosophy*, 77(1):46–66, 1999.

[6] E. Damiani, V. Liberali, and A. Tettamanzi. Dynamic Optimisation of Non-linear Feed-Forward Circuits. In *Proc. of the 3rd Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'00*, volume 1801 of *LNCS*, pages 41–50, Edinburgh, Scotland, UK, 2000. Springer-Verlag.

[7] E. Eberbach. On Expressiveness of Evolutionary Computation: Is EC Algorithmic? In *Proc. of Congress on Evolutionary Computation 2002*, pages 564–569. IEEE Press, 2002.

[8] E. Eberbach, D. Goldin, and P. Wegner. Turing's Ideas and Models of Computation. In C. Teuscher, editor, *Alang Turing: Life and Legacy of a Great Thinker*, pages 166–173. Springer-Verlag, 2004.

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

[10] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya. Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In *Proc. of the 2nd Int. Conf. on Simulated Adaptive Behaviour*, pages 417–424. MIT Press, 1993.

[11] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iawta, D. Keymeulen, and T. Higuchi. A Gate Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI. In *Proc. of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'98*, volume 1478 of *LNCS*, pages 1–12, Lausanne, Switzerland, 1998. Springer-Verlag.

[12] D. E. Knuth. *The Art of Computer Programming: Sorting and Searching (2nd ed.)*. Addison Wesley, 1998.

[13] J. R. Koza, F. H. Bennett, D. Andre, and M. A. Keane. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, 1999.

[14] F. Kurdahi, N. Bagherzadeh, P. Athanas, and J. Munoz. Configurable Computing. *IEEE Design and Test*, January-March 2000.

[15] M. Love, K. R. Sorensen, J. Larsen, and J. Clausen. Disruption Management for an Airline – Rescheduling of Aircraft. In *Applications of Evolutionary Computing, EvoWorkshops 2002*, volume 2279 of *LNCS*, pages 315–324. Springer-Verlag, 2002.

[16] C. Luer, D. S. Rosenblum, and A. van der Hoek. The Evolution of Software Evolvability. In *Int. Conf. on Software Engineering, Proc. of the 4th Int. Workshop on Principles of Software Evolution*, pages 134–137. ACM Press, 2001.

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

[18] S. Nolfi and D. Floreano. *Evolutionary Robotics: The Biology, Intelligence, and Technology of Self-Organizing Machines*. MIT Press, Cambridge MA, 2000.

[19] picoChip home page, 2003. http://www.picochip.com.

[20] J. M. Rabaey. Silicon Platforms for the Next Generation Wireless Systems – What Role Does Reconfigurable Hardware Play? In *Proc. of the 10th Int. Conf. on Field-Programmable Logic and Applications FPL2000*, volume 1896 of *LNCS*, pages 277–285, Villach, Austria, 2000. Springer-Verlag.

[21] L. Sekanina. Towards Evolvable IP Cores for FPGAs. In *Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 145–154, Chicago, IL, 2003. IEEE Computer Society.

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

[23] M. Sgroi, L. Lavango, and A. Sangiovanni-Vincentelli. Formal Models for Embedded System Design. *IEEE Design and Test*, 17(2):14–27, 2000.

[24] J. Smith. Implementing Median Filters in XC4000E FPGAs. *XCell Journal, Xilinx, Inc.*, (23), 1996. http://www.xilinx.com/xcell/xl23/xl23_16.pdf.

[25] C. R. Stephens and A. Zamora. EC Theory: A Unified Viewpoint. In *Proc. of GECCO 2003*, volume 2724 of *LNCS*, pages 1394–1405. Springer-Verlag, 2003.

[26] A. Stoica. Evolvable Hardware for Autonomous Systems. In *Tutorials given at the Congress on Evolutionary Computation CEC'03*, page 132, 2003.

[27] A. Stoica and et al. Evolvable Hardware for Space Applications. In *Proc. of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware*, volume 1478 of *LNCS*, pages 166–173, Lausanne, Switzerland, 2002. Springer-Verlag.

[28] A. Stoica, D. Keymeulen, A. Thakoor, T. Daud, G. Klimech, Y. Jin, R. Tawel, and V. Duong. Evolution of Analog Circuits on Field Programmable Transistor Arrays. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware*, pages 99–108, Palo Alta, CA, 2002. IEEE Computer Society.

[29] A. Stoica, R. S. Zebulum, D. Keymeulen, M. I. Ferguson, and X. Guo. Evolving Circuits in Seconds: Experiments with a Stand-Alone Board Level Evolvable System. In *Proc. of the 2002 NASA/DoD Conference on Evolvable Hardware*, pages 67–74, Alexandria, Virginia, 2002. IEEE Computer Society.

[30] M. Tanaka, H. Sakanashi, M. Salami, M. Iwata, T. Kurita, and T. Higuchi. Data Compression for Digital Color Electrophotographic Printer with Evolvable Hardware. In *Proc. of the 2nd Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'98*, volume 1478 of *LNCS*, pages 106–114, Lausanne, Switzerland, 1998. Springer-Verlag.

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

[32] J. Torresen. Possibilities and Limitations of Applying Evolvable Hardware to Real-World Applications. In *Proc. of the 10th Int. Conf. on Field-Programmable Logic and Applications FPL2000*, volume 1896 of *LNCS*, pages 203–239, Villach, Austria, 2000. Springer-Verlag.

[33] J. M. Tour. *Molecular Electronics*. World Scientific, 2003.

[34] J. W. Valvano. *Embedded Microcomputer Systems*. Brooks/Cole, 2000.

[35] J. van Leeuwen and J. Wiedermann. A Computational Model of Interaction in Embedded Systems. Technical Report UU-CS-2001-02, Utrecht University, 2001.

[36] J. van Leeuwen and J. Wiedermann. The Turing Machine Paradigm in Contemporary Computing. In *Mathematics Unlimited – 2001 and Beyond*, pages 1139–1155. Springer-Verlag, 2001.

[37] M. Vorbach and J. Becker. Reconfigurable Processor Architectures for Mobile Phones. In *Proc. of the Int. Parallel and Distributed Processing Symposium (IPDPS'03)*, pages 181–190. IEEE Press, 2003.

[38] X. Yao and T. Higuchi. Promises and Challenges of Evolvable Hardware. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(1):87–97, 1999.