CHAPTER 33

EVOLVABLE FPGAs

Andres Upegui, Eduardo Sanchez
School of Computer and Communication Sciences
Ecole Polytechnique Fédérale de Lausanne
Reconfigurable and Embedded Digital Systems Institute
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

One of the main advantages of living beings over engineered computing systems is their capacity to adapt. While computers are tied to a fixed architecture predefined at design time, the human brain exhibits an impressive structural plasticity whereby interconnections are constantly being reinforced or destroyed according to environmental interactions. This and other comparisons between computers and living beings have given rise to what we know today as bioinspired hardware design.

Evolvable hardware is a bioinspired technique that has enjoyed impressive growth during the last decade. In 1993 Higuchi et al. and de Garis proposed an analogy between living beings and programmable hardware devices [1,2]: In both cases specification of the system is by means of a finite string of symbols. In the case of living beings, DNA determines how the organism develops into its final phenotypic representation; in programmable hardware devices, a configuration bitstream drives behavior. This parallel suggests the utilization of so-called *evolutionary algorithms* in the design of hardware systems.

33.1 THE POE MODEL OF BIOINSPIRED DESIGN METHODOLOGIES

Living organisms, from microscopic bacteria to giant sequoias, including animals such as butterflies and humans, have successfully survived on Earth for millions of years. If we had to propose but one key to explain this success, it certainly would be *adaptation*. In contrast with nature, adaptation has been very elusive to human technology. The model examples of adaptive systems are not among human's creations but among nature's—natural organisms show a striking capacity to adapt to changing circumstances, thus ensuring their continued functionality.

During the last few years, computer scientists, inspired by certain biological processes, have given birth to domains such as artificial neural networks and evolutionary computation.

Living organisms are complex systems exhibiting a range of desirable characteristics, such as evolution, adaptation, and fault tolerance, which have proved difficult to realize using traditional engineering methodologies. Such systems are characterized by a genetic program—the genome—that guides their development, their functioning, and their death. If one considers life on Earth from its very beginning, the following three levels of organization can be distinguished [3].

Phylogeny: The first level is the temporal evolution of the genetic program, the hallmark of which is the evolution of species, or *phylogeny*. The multiplication of living organisms is based on the reproduction of the program, subject to an extremely low error rate at the individual level to ensure that the species of the offspring remains unchanged. Mutation (asexual reproduction) or mutation with recombination (sexual reproduction) gives rise to new organisms. The phylogenetic mechanisms are fundamentally nondeterministic, with the mutation and recombination rate providing a major source of diversity. This diversity is indispensable for the survival of living species, for their continuous adaptation to a changing environment, and for the appearance of new species.

Ontogeny: This level constitutes the developmental process of multicellular organisms. The successive divisions of the mother cell, the zygote, into newly formed cells, each possessing a copy of the original genome, is followed by a specialization of the daughter cells in accordance with their surroundings (i.e., their position within the ensemble). This latter phase is known as cellular differentiation. The ontogenetic process is essentially deterministic: An error in a single base within the genome can provoke an ontogenetic sequence that results in notable, possibly lethal, malformations.

Epigenesis: The ontogenetic program is limited in the amount of information it can store, rendering the complete specification of the organism impossible. A well-known example is the human brain, whose some 10^{10} neurons and 10^{14} connections are far too many to be completely specified in the 4-character genome with a length of approximately 3×10^9. Therefore, when a certain level of complexity is reached, there must emerge a different process that permits the individual to integrate its vast quantity of interactions with the outside world. This is known as *epigenesis*, which primarily includes the nervous, immune, and endocrine systems. These systems are characterized by a basic structure that is entirely defined by the genome (the innate part), which is then subjected to modification through the individual's lifetime interactions with the environment (the acquired part). The epigenetic processes can be grouped under the heading of *learning* systems.

Analogous to nature, the space of bio-inspired hardware systems can be partitioned along the phylogenic, ontogenic, and epigenetic axes; we refer to this as the POE model [3, 4]. The distinction between the axes cannot be easily drawn

where nature is concerned. We therefore define each axis within the model's framework as follows:

- The phylogenetic axis involves *evolution*.
- The ontogenetic axis involves the *development* of a single individual from its own genetic material, essentially without environmental interactions.
- The epigenetic axis involves *learning* through environmental interactions that take place after the individual is formed.

As an example, consider the following three paradigms, whose hardware implementations can be positioned along the POE axes:

- *P*—evolutionary algorithms are the simplified artificial counterpart of phylogeny.
- *O*—self-replicating and self-repairing cellular automata are based on the concept of ontogeny, where a single mother cell gives rise through multiple divisions to a multicellular organism.
- *E*—artificial neural networks embody the epigenetic process, where the system's synaptic weights and perhaps topological structure change through interactions with the environment.

The domains collectively referred to as soft computing [5] often involve the solution of ill-defined problems coupled with the need for continual adaptation or evolution. The paradigms listed yield impressive results, frequently rivaling those of traditional methods.

We will talk about the phylogenetic axis of hardware bio-inspired systems, most known as evolvable hardware (EHW). The scope of EHW covers diverse areas ranging from analog circuits to antenna design, but this chapter focuses on evolution of digital circuits using reconfigurable computing devices, more precisely, field-programmable gate arrays (FPGAs).

33.2 ARTIFICIAL EVOLUTION

The idea of applying the biological principle of natural evolution to artificial systems, introduced more than three decades ago, has seen impressive growth in the past few years. Usually grouped under the term *evolutionary algorithms* (EAs) or *evolutionary computation*, we find the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming [6–9].

33.2.1 Genetic Algorithms

As a generic example of artificial evolution, we consider genetic algorithms (GAs) [10]. As illustrated in Figure 33.1, a GA is an iterative procedure applied to a constant-size population of individuals. Each individual represents a possible

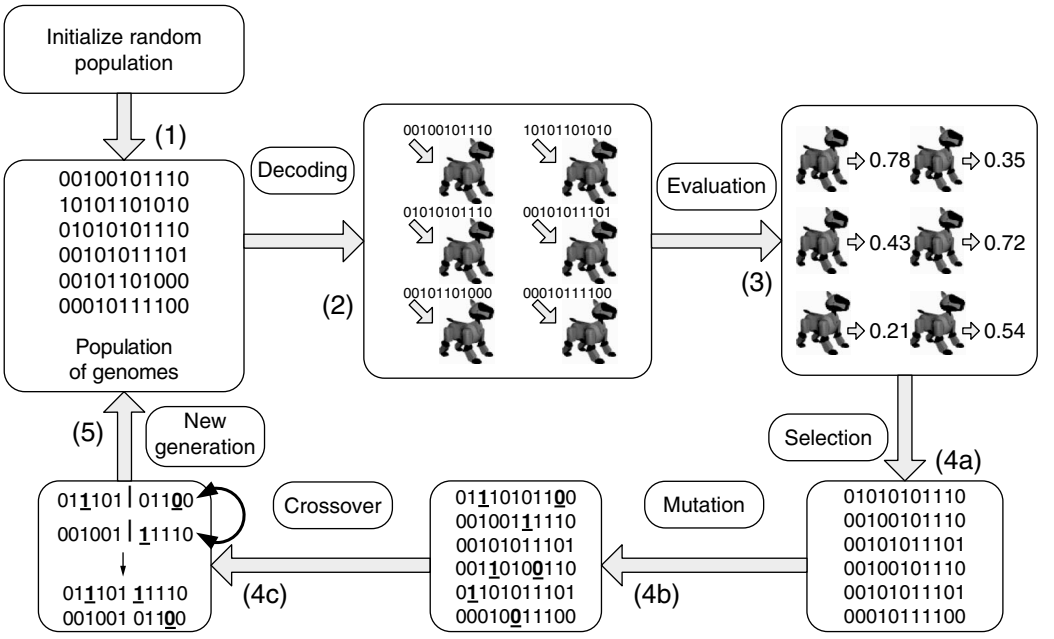

FIGURE 33.1 ■ A genetic algorithm.

solution to the given problem, and eventually one is chosen as the searched solution.

Each individual is coded by a finite string of symbols from a given alphabet, known as the *genome*. Each genome gives rise to the individual's *phenotype*, which constitutes the actual solution (a program or a circuit) to the problem at hand (e.g., a robot controller for the example in Figure 33.1). The individual receives a score (better known as fitness) depending on the performance exhibited during its evaluation. The process from the genome to a fitness value can be seen as an n-dimensional function (where n is the genome size), and the set of all possible solutions can be seen as an n-dimensional *search space*.

A GA can be summarized in the following steps:

1. *Initialization:* Create an initial population of individuals by defining a set of genomes in a random or heuristic manner.
2. *Decoding:* Generate the phenotypes for the individuals in the current population by decoding (mapping) the genotypes.
3. *Fitness evaluation:* Evaluate individuals according to some predefined quality criterion, referred to as *fitness* or *fitness function*.
4. *Genetic operators:* Apply genetically inspired operators to the current population:
   (a) *Selection:* Individuals are selected into a mating pool for reproduction according to their fitness. With stochastic or deterministic

selection mechanisms, the fittest individuals have more chances to transmit their genetic material to the next generation.
   (b) *Mutation:* The genome is randomly changed; and
   (c) *Crossover:* Two genomes are selected to be split and swapped at a random position.
5. If a predefined convergence condition has not been met, go back to step 2 to evaluate a new generation. Otherwise, deliver the best individual evaluated.

The basic components of GAs are always the same: a population of individuals, a decoding mechanism from a genotype to a phenotype, a fitness evaluation, genetic operators, and an iterative process. However, GAs allow variants: There exist several methods for defining each of the steps just listed. By running a large enough number of generations, the GA should eventually find an acceptable solution (i.e., one with high fitness).

EAs can be considered as a family of stochastic global optimization algorithms, mainly differing from their deterministic counterparts [11] by the lower knowledge of the problem they require and by the absence of mathematical proofs of convergence due to their stochastic nature. For highly nonlinear search spaces, EAs have exhibited faster convergence than deterministic methods, given their population-based approach. In most cases, the applications solved by EAs can also be tackled with deterministic optimization methods.

EAs are very common, having been successfully applied to numerous problems from domains as diverse as optimization, circuit design, disease diagnosis assistance, precision agriculture, self-organizing systems, automatic programming, machine learning, economics, immune systems, ecology, population genetics, studies of evolution and learning, and social systems [9].

33.3 EVOLVABLE HARDWARE

In the case of humans, adaptation due to evolution comes about through modifications in our DNA (deoxyribonucleic acid), which constitutes the encoding of every living being on Earth. DNA is a double-stranded molecule composed of two sugar-phosphate chains linked together by pairs of the bases adenine, cytocine, guanine, and thymine, constituting a string of symbols from a quaternary alphabet (A, C, G, T). Similarly, reconfigurable logic devices are configured by a string of symbols (the configuration bitstream) from a binary alphabet (0, 1). This string determines the function implemented by each of the programmable components and the connectionism of each of the switch matrices.

With this description, a rough analogy arises naturally between DNA and a configuration bitstream and between a living being and a circuit (Figure 33.2). In both cases there is a mapping from a string representation to an entity that will perform one or more actions: growing, moving, reproducing, and so forth, for living beings; computing a function for circuits.

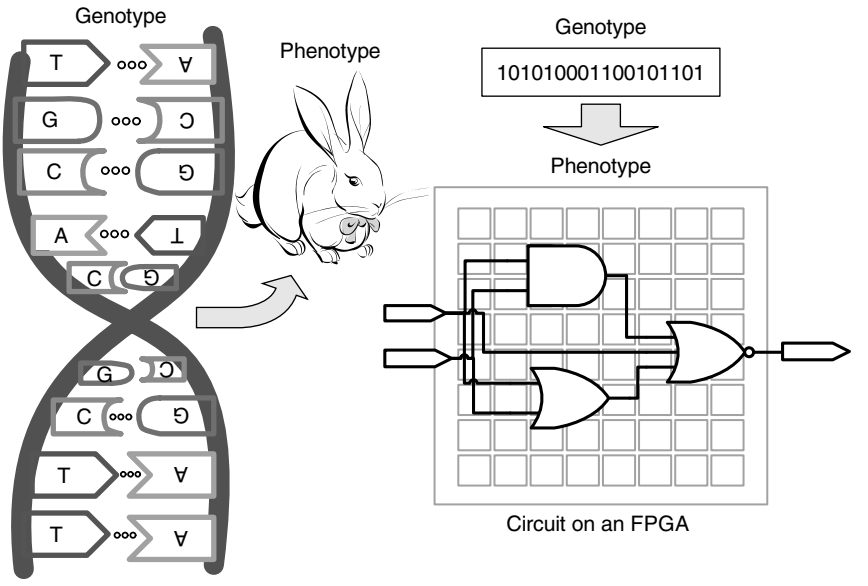

FIGURE 33.2 ■ The analogy between living beings and digital circuits.

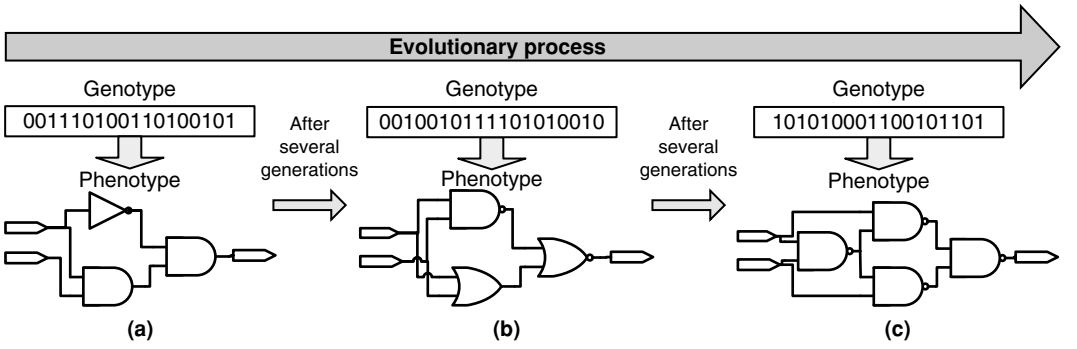

FIGURE 33.3 ■ The evolutionary design of digital circuits: (a) intial random circuit, (b) intermediate circuit, and (c) final circuit.

This analogy between living beings and digital circuits suggests the possibility of applying the principles of artificial evolution to circuit design (Figure 33.3). Designing analog and digital electrical circuits is, by tradition, a hard engineering task vulnerable to human error, and for large circuits the optimality of a solution cannot be guaranteed. Design automation has become a challenge for tool designers, and given the increasing complexity of circuits, higher abstraction levels are needed. Evolvable hardware arises as a promising solution to this

problem: From a given behavior specification of a circuit, an EA will search for a bitstream describing a circuit that satisfies it.

If we carefully examine the EHW work carried out to date, it becomes evident that it mostly involves the application of EAs to the synthesis of digital systems [12–23]. From this perspective, EHW is simply a subdomain of artificial evolution, where the final goal is the synthesis of an electronic circuit. The work of Koza [8], which includes the application of genetic programming to the evolution of a 3-variable multiplexer and a 2-bit adder, may be considered an early precursor along this line. It should be noted that, in Koza's time, the main goal was to demonstrate the capabilities of the genetic programming methodology rather than to design actual circuits. We argue that the term *evolutionary circuit design* would be more descriptive of such work than the term *evolvable hardware* [24]. For now, we will stay with the latter (popular) term; however, we will return to the issue of definitions in Section 33.4.

Taken as a design methodology, EHW offers a major advantage over classical methods. The designer's job is reduced to constructing the evolutionary setup, which involves specifying the circuit requirements, the basic elements, a decoding mechanism, and the testing scheme used to assign fitness (this last phase is often the most difficult). If the setup has been well designed, evolution may then (automatically) generate the desired circuit. Currently, most evolved digital designs are suboptimal with respect to traditional methodologies; however, improved results are regularly demonstrated.

There are two critical questions to ask when setting up a system to be evolved: how to map a phenotype from a genotype and how to compute the fitness of a circuit. The answers to these questions are critical and can make the difference between a successful and an unsuccessful evolution.

33.3.1 Genome Encoding

In examining the EHW work carried out to date, we can derive a classification of current EHW in accordance with genome encoding (i.e., the circuit description) and the calculation of a circuit's fitness.

High-level languages

Using a high-level functional language to encode the evolving population implies an additional step to obtain the final circuit implementation: The chosen individual must be synthesized. Koza's evolved solution [8] was a program that described the (desired) multiplexer or adder rather than an interconnection diagram of logic elements (the actual hardware representation). Mermoud et al. [25] used fuzzy rules as evolvable components, and Murakawa et al. [26] and Upegui et al. [27] proposed the evolution of artificial neural network topologies at the neuron and layer levels. Hemmi et al. [28] used a high-level HDL to represent the genomes. Koza et al. [29] used the rewriting operator, in addition to crossover and mutation, to form a hierarchical structure.

Low-level languages

The idea of directly incorporating the bit string representing the configuration of a programmable circuit within the genome was presented early on by Atmar [30] and more recently by Higuchi et al. [1] and de Garis [2]. As a first step, a set of basic logic gates must be chosen (e.g., AND, OR, and NOT) and suitably codified, along with the interconnections between gates, to produce the genome encoding. For example, Higuchi et al. [31] used a low-level bit-string representation of the system's logic diagram to describe small-scale programmable array logics (PALs), where the circuit is restricted to a logic sum of products. The limitations of PAL circuits have been overcome to a large extent by the introduction of FPGAs, as used initially by Thompson [32,33] and later by a number of research groups.

The use of a low-level circuit description that requires no further transformation is an important step forward because it potentially enabled the placing of the genome directly into the actual circuit and thus paved the way toward true EHW (we will elaborate on this in Section 33.4). However, FPGAs presented two major problems: (1) The genome's length was on the order of tens of thousands of bits, rendering evolution practically impossible using current technology, and (2) within the circuit space, consisting of all representable circuits, many circuits were invalid.

With the introduction of the Xilinx XC6200 [34] family of FPGAs, these problems were reduced. As with previous FPGA families, there was a direct correspondence between the bit string of a cell and the actual logic circuit; however, because the XC6200 was completely multiplexer based, the result was always a viable system with no short circuits. Moreover, as opposed to previous FPGAs where the entire system had to be configured, the XC6200 family permitted the separate configuration of each cell, which was markedly faster and more flexible. Thompson [32] employed this feature to reduce the genome's size, although he did not introduce real-time, partial system reconfigurations. Unfortunately, the XC6200 was discontinued after a few years; however, the results achieved by directly evolving its bitstream led to increased visibility for the EHW community and made possible the growth of this research field.

Fitness calculation

Note the following with regard to calculations for fitness with evolvable hardware.

- *Off-chip*. The use of a high-level language for genome representation means that we have to transform the encoded system to evaluate its fitness. This is usually carried out by simulation, and only the final solution found by evolution is actually implemented in hardware.
- *On-chip*. As noted previously, the low-level genome representation enables a direct configuration (and reconfiguration) of the circuit, which leads to the possibility of using real hardware during the evolutionary process. An example of *on-chip fitness calculation* is presented in the next section in the form of an intrinsic evolvable system.

33.4 EVOLVABLE HARDWARE: A TAXONOMY

In EHW, the phylogenetic axis admits four qualitative subdivisions of evolution (Figure 33.4) according to the level of bio-inspiration: extrinsic, intrinsic, complete, and open ended.

33.4.1 Extrinsic Evolution

At the bottom of this axis, we find what is in essence *evolutionary circuit design*, where all operations are carried out in software, and the resulting solution may be loaded onto a real circuit. Though a potentially useful design methodology, this falls completely within the realm of traditional evolutionary techniques. This category is also widely known as *extrinsic* EHW.

Extrinsic EHW has typically targeted the synthesis of circuits—that is, from a desired behavior specification, an EA finds a schematic of a circuit implementing a function that satisfies the specification [29]. This category supports different levels of abstraction, allowing to evolve logical gates, arithmetic operations, more complex functional blocks, or HDL code; however, it is not suited for evolving circuits at the bitstream level. Evolution has also been used in other extrinsic aspects of circuit design such as placement and routing [35, 36] and scheduling and allocation [37].

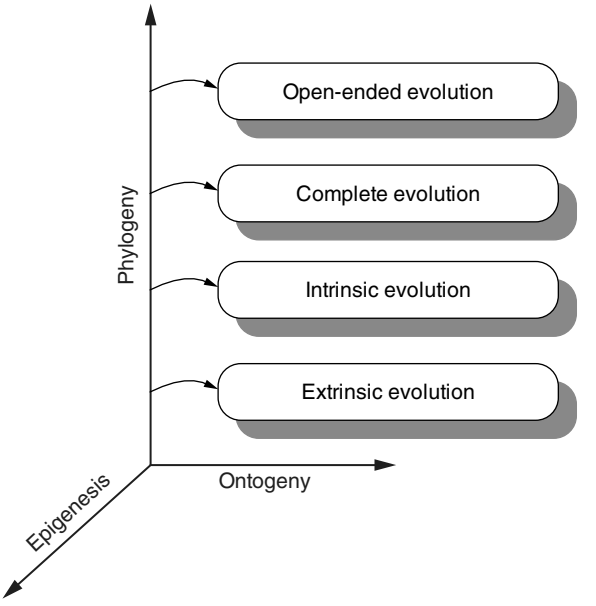

FIGURE 33.4 ■ The divisions of phylogenetic hardware.

33.4.2 Intrinsic Evolution

Moving upward along the axis, we find research in which a real circuit is used during the evolutionary process for fitness computation, although most operations are still carried out offline, in software, as depicted in Figure 33.5.

The very first intrinsic evolution was reported by Thompson [32]. He evolved a section of an XC6216 FPGA, consisting of 10×10 cells (the full array size was 64×64), to discriminate between square waves of 1 kHz and 10 kHz presented as inputs. His complete system setup is depicted in Figure 33.6 (see Thompson [33]). From a PC, he configured the FPGA with a configuration bitstream generated by a GA, which used a genome of 1800 bits (18 configuration bits per cell) to represent a possible circuit. Then the individual's fitness was automatically evaluated as follows:

1. The tone generator, driven by the PC, presented five bursts each of both waves (1 kHz and 10 kHz) to the circuit. The analog integrator was reset before the generation of each burst, and it then integrated the circuit's output during the presentation of the burst.
2. Back in the PC, the individual's fitness was computed by a function aiming to maximize the difference between the average output voltages when presenting both waves.
3. After running the experiment for 2 to 3 weeks, during which 5000 generations of 50 individuals were evaluated, the resulting circuit achieved successful discrimination of the waves. However, the perfect desired behavior was obtained around generation 4100.

In another interesting project, Thompson et al. [38] evolved a hardware controller for a two-wheeled autonomous mobile robot that was required to display simple wall avoidance behavior in an empty rectangular arena.

A very important aspect of Thompson's work is the unconstrained use of hardware. Conventional (human) design requires that constraints be applied to the circuit's spatial structure and dynamic behavior, but evolution can do away with

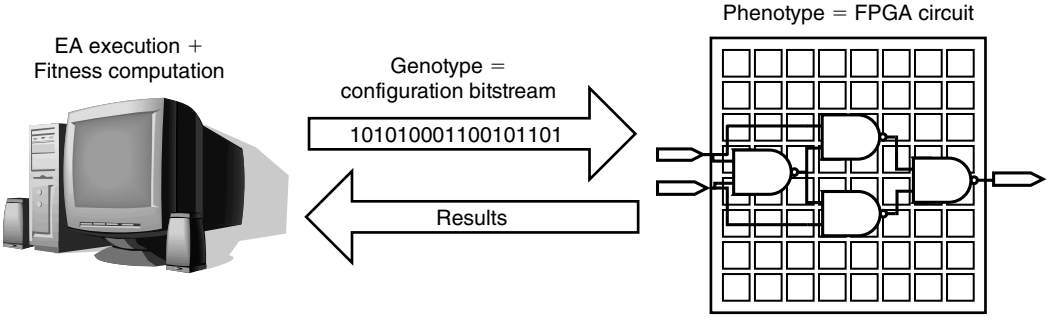

FIGURE 33.5 ■ Intrinsic evolution.

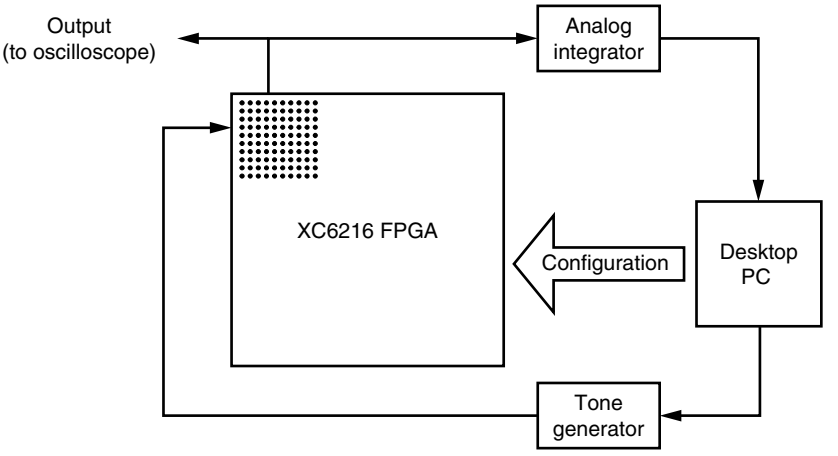

FIGURE 33.6 ■ Adrian Thompson's intrinsic evolvable system setup.

these. The circuits evolved by Thompson [33, 38] and Ly and Mowchenko [37] had no enforced spatial structure (e.g., limitations on recurrent connections), no impositions upon modularity, and no dynamic constraints such as a synchronizing clock or handshaking between modules. Unconstrained circuit design can better exploit the dynamics of the circuit supporting it; however, such circuits exhibit two main drawbacks. One is the impossibility of reproducing a solution: The same bitstream does not behave in the same manner in two different devices. The other is the circuit's high sensitivity to external conditions: Slight temperature changes can modify its behavior.

Two more examples from this subdivision of the phylogenetic axis are the works of Murakawa et al. [39] and Iwata et al. [40]. One of the major obstacles these researchers hoped to overcome was large genome size (defining the FPGA's full configuration). They suggested two solutions:

1. Variable-length chromosome GAs (VGA), where the genome does not directly represent the configuration bit string but rather codifies the possible logical operations and interconnections [40].
2. Evolution at the function level, where the basic units are not elementary logic gates (e.g., AND, OR, and NOT) but rather higher-level functions (e.g., sine-wave generator, multiplier) [39].

Because no such commercial FPGA currently exists, Murakawa and Iwata and their colleagues proposed a novel architecture, dubbed F^2PGA (function-based FPGA).

It is important to note that while experiments of the above type have been referred to by some as intrinsic evolution, they have a prominent extrinsic aspect because the population is stored in an external computer, which also controls the evolutionary process.

33.4.3 Complete Evolution

Still further along the phylogenetic axis, we find systems in which all operations (selection, crossover, mutation), as well as fitness evaluation, are carried out intrinsically, in hardware (Figure 33.7). This category, called *complete* evolution by Haddow and Tufte [41], has as its main motivation attaining adaptive systems that are able to accomplish difficult tasks, possibly involving real-time behavior in a complex, dynamic environment. The major aspect missing here, compared with biological evolution, is that the evolution is not open ended (i.e., there is a predefined goal and no dynamic environment to speak of).

Within the category of complete evolution, we find two subdivisions: *centralized* and *population oriented*.

Centralized evolution

The main characteristic of centralized evolution is the existence of a single evolvable circuit and a single evolvable algorithm computation (Figure 33.7(a)). With this approach an on-chip genetic machine, a hardwired EA, is implemented. The approach also comprises implementations where the EA is executed in an on-chip processor. Centralized evolution holds special interest because it greatly enhances the autonomy of the circuit, allowing the EHW to adapt to a changing environment during its lifetime. Implementations of EAs in general-purpose processors, in spite of their lower performance compared to their fully hardwired counterparts, exhibit several important advantages that permit them to benefit from a more general framework: They provide a more user-friendly interface for implementing chromosome manipulations, fitness evaluations, and memory access; they support easier algorithm upgrades; and they enhance the possibilities of immediately using the evolving circuit for useful computations.

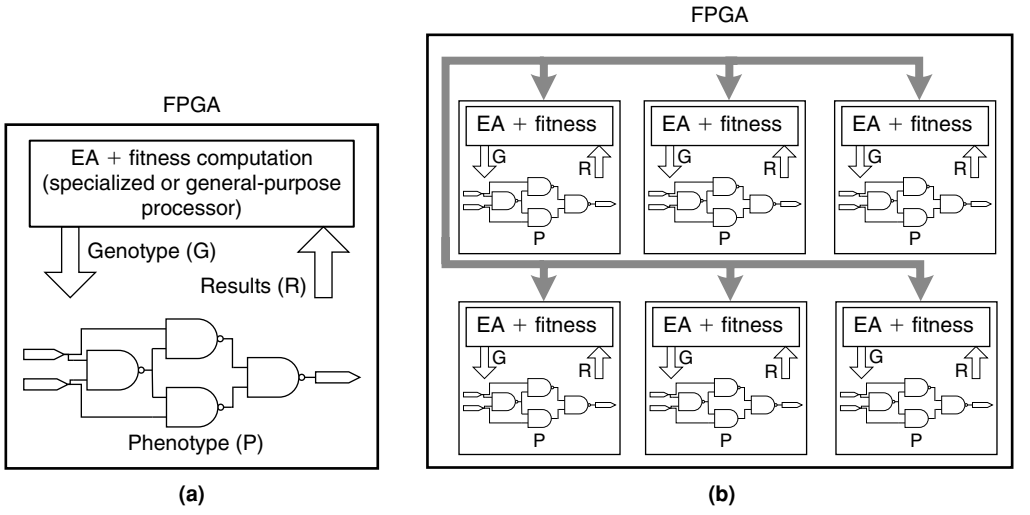

FIGURE 33.7 ■ Complete evolution: centralized (a) and population oriented (b).

One example of a self-reconfigurable platform that performs online and on-chip evolution is that of Upegui and Sanchez [42, 43]. Their standalone platform consists of a MicroBlaze processor with memory access control, ICAP (internal configuration access port) access, and a reconfigurable evolvable section, as depicted in Figure 33.8. The full system, implemented in a Virtex-II FPGA, runs an EA on the MicroBlaze processor, reads a section of the configuration bitstream through the ICAP, modifies the bitstream according to the genome currently evaluated in the MicroBlaze, sends back the bitstream though the ICAP for partially reconfiguring the FPGA, and evaluates the fitness of the current individual by interacting with the reconfigurable evolvable section through the standard OPB bus. Upegui and Sanchez [42] evolve nonuniform cellular rules and FPGA lookup table (LUT) configurations with fixed interconnectivity. In Upegui and Sanchez [43], Boolean networks are evolved as well, but in this case the interconnectivity is not fixed, so the system topology is also driven by the evolutionary algorithm.

Other interesting experiments were carried out by Haddow and Tufte [41] in which a hardware implementation of a GA, the "GA pipeline," evolves a robot controller. Glette and Torresen [44] report the implementation of a GA on an embedded PowerPC processor in a Virtex-II Pro FPGA that evolves a circuit in the same FPGA.

Population-oriented evolution

A hardware implementation of the *full population*, not only of one individual (as was the case in previous categories), is the distinctive feature of the population-oriented approach (Figure 33.7(b)). A significant example is the work of Goeke et al. [45], where an evolving cellular system was implemented in which evolution takes place completely on-chip. This system is based on the cellular automata model—a discrete dynamic system that performs computations in

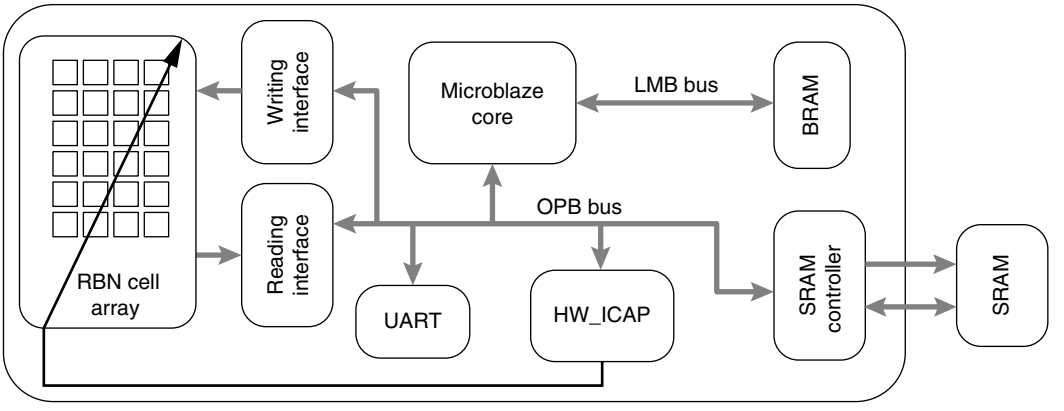

FIGURE 33.8 ■ The setup of a complete and centralized self-reconfigurable evolvable platform.

a distributed fashion on a spatially extended grid. A cellular automaton consists of an array of cells, each of which can be in one of a finite number of possible states, updated synchronously in discrete timesteps according to a *local*, *identical* interaction rule [46]. The *state* of a cell at the next timestep is determined by the current state of a surrounding neighborhood of cells. This transition is usually specified in the form of a *rule table*, which delineates the cell's next state for each possible neighborhood configuration. The cellular array (grid) is n-dimensional, where typically $n = 1, 2, 3$. Nonuniform cellular automata have also been considered in which the local update rule need not be identical for all grid cells [47].

Based on the *cellular programming* EA of Sipper [47], Goeke et al. [45] implemented an evolving, one-dimensional, nonuniform cellular automaton. The main feature of the cellular programming algorithm is the fact that genetic operators are computed in a distributed way: Each automaton modifies its own rule based on its own and its neighbors' fitness. Each of the system's 56 binary-state cells contains a genome that represents its rule table. These genomes are initialized at random and then are subjected to evolution.

The environment imposed on the system specifies the resolution of a global synchronization task: On presentation of a random initial configuration of cellular states, the system must reach, after a bounded number of timesteps, a configuration for which the states of the cells oscillate between all zeros and all ones on successive timesteps. This may be compared to a swarm of fireflies, thousands of which may flash on and off in unison, having started from totally uncoordinated flickerings. Each insect has its own rhythm, which changes only through local interactions with its neighbors'. Because of the local connectivity of the system, this global behavior, which involves the entire grid, makes for a difficult task. Nonetheless, applying the evolutionary process of Sipper [47], the system evolves (i.e., the genomes change) such that the task is completed.

The evolving cellular system described here exhibits complete on-chip evolution in that all operations are performed in hardware in a distributed population-based manner with no reference to an external computer.

33.4.4 Open-ended Evolution

The last subdivision, situated at the top of the phylogenetic axis, involves a population of hardware entities evolving in an open-ended environment. When the fitness criterion is imposed by the user in accordance with the task to be performed (currently the rule with artificial evolution techniques), we attain a form of guided, or directed, evolution. This is to be contrasted with the open-ended evolution that occurs in nature, which admits no externally imposed fitness criterion but rather an implicit, emergent, dynamic one (which can arguably be summed up as reproducibility). Open-ended undirected evolution is the only form of evolution known to produce such devices as eyes, wings, and nervous systems and to give rise to the formation of species. Undirectedness may have to be applied to artificial evolution if we want to observe the emergence of completely novel systems.

We argue that only open-ended evolution can be truly considered EHW, which is still an elusive goal at present. We point out that a more correct term would probably be *evolving hardware*. A natural application area for such systems is the field of autonomous robots—that is, machines capable of operating in unknown environments without human intervention [48]. Specifically, collective robotics exhibits a population of individuals interacting in a common environment, in which they can learn to cooperate or to compete for achieving their goals [49]. In their interactions the individuals exhibit a high level of emergence as a first step to open endedness. Modular robotics, a subtype of collective robotics, also offers a promising open-ended real environment.

A modular robotic platform well suited for evolving distributed hardware is YaMoR. This is a modular robot composed of mechanically homogeneous modules [50], each of which contains an FPGA-based system that allows wireless FPGA configuration and on-board self-reconfiguration. Another interesting example is what we call Hard-Tierra. This involves the hardware implementation (e.g., FPGA circuits) of the Tierra "world," which consists of an open-ended environment of evolving computer programs [51]. Hard-Tierra is important because it demonstrates that open-endedness does not necessarily imply a real, biological environment.

33.5 EVOLVABLE HARDWARE DIGITAL PLATFORMS

The hardware substrate that supports evolution is one of the most important initial decisions to make when evolving hardware. The hardware architecture is closely related to the type of solution being evolved. Hardware platforms usually have a cellular structure composed of uniform or nonuniform components. In some cases, we can evolve the components' functionality; in others, the connectivity; or sometimes both, with the most powerful ones. FPGAs fit well into this third category because they are composed of configurable logic elements interconnected by configurable switch matrices. FPGA configuration is contained in a configuration bitstream, which holds every function and switch position to be configured for implementing a given design. Current FPGAs allow the processing of partial bitstreams, reconfiguring just a sector of the FPGA while the remaining logic stays the same.

When evolving a circuit on an FPGA, we consider the logic cell as the basic element. The logic cells' configuration and their interconnectivity are defined by the evolution. However, this implies a huge search space to explore and can prevent the EA from finding a solution. A common technique to constrain the search space is to define a basic block as a set of logic cells. In this way each basic block can be an artificial neuron, a fuzzy rule, or a more complex cell in general. Another option is to constrain the connectionism, using layered architectures, to a certain neighborhood, or by just defining it as fixed.

The most basic requirement when evolving hardware is to have a set of high- or low-level evolvable components and a hardware substrate supporting them.

These evolvable components are the basic elements from which the evolved circuits will be built (transistors, logic gates, arithmetic functions, functional cells, etc.), and the evolvable substrate must be a flexible hardware platform that allows arbitrary configurations mapped from a genome. FPGAs constitute the perfect hardware substrate, given their connectivity and functional flexibility. The evolvable substrate can be implemented using one of two main techniques: (1) exploiting the flexibility provided by the FPGA's configuration logic and (2) building a virtual flexible substrate on top of the logic.

In the first approach the configuration bitstream of the FPGA is directly generated. In this way, we can make better use of FPGA resources—logic functions are directly mapped into the FPGAs LUTs, and connections are directly mapped to routing switch matrices and multiplexers—but the penalty is very low-level circuit descriptions [33, 38, 52]. In the second approach a virtual reconfigurable circuit is built on top of the actual circuit [53]. In this way the designer can also define the configuration bitstream and determine which features of the circuit to evolve. This approach has been widely used by several groups, as it produces enhanced flexibility and ease of implementation. The penalty here is the cost of an inefficient use of logic resources [25, 27, 42, 45, 53–60].

Different custom chips have been proposed for this purpose with very interesting results: The main interest in proposing an architecture is that commercial FPGAs are designed for general-purpose applications, so they do not necessarily fit the requirements for evolvable architectures. For example, commercial devices may have illegal configurations that cause short circuits; this is reasonable for standard FPGA users who rely on the CAD flow to create the design, but it can be disastrous for genetically evolved bitstreams. Custom evolvable chips generally provide dynamic and partial reconfiguration, contain multi-context configuration memories, and can be configured with arbitrary bitstreams. However, although the custom chips are better suited to EHW applications, the commodity devices benefit from economies of scale and access to more advanced fabrication processes.

Different chips and platforms have been developed to provide the flexibility necessary for evolving analog, digital, and mixed circuits; some of them have been designed specifically for EHW, while for others EHW is just another application field. Among them we find different levels of granularity, different types of reconfiguration including dynamic and static reconfigurations, and the possibility of loading partial configuration bitstreams, and the utilization of context memories.

33.5.1 Xilinx XC6200 Family

The obsolete Xilinx XC6200 family [61] deserves a special mention in a discussion of EHW platforms. For several years, the XC6200 family constituted the perfect platform for intrinsic EHW, because it made possible downloading any arbitrary bitstream without risking contention given its multiplexer-based connection architecture. It also allowed dynamic reconfiguration, making it more flexible for adaptive algorithms in a general sense. The results reported

by Thompson [32, 33, 38, 62], discussed previously, are a very good example of the XC6200's potential for evolving circuits.

The XC6200 represents an important initial stepping-stone in the EHW field. It has also been used for implementing several types of applications, among them cooperative robot controllers [63], sorting networks [64], and image-processing algorithms [65].

33.5.2 Evolution on Commercial FPGAs

After the XC6200 disappeared, many research groups turned to the Xilinx XC4000 family. However, these FPGAs had an important drawback for evolving hardware: They were not partially reconfigurable, and no arbitrary bitstreams were allowed. When the Virtex FPGAs appeared, they exhibited two well-appreciated features for the EHW community: partial and dynamic reconfiguration. However, not all the evolution-friendly features from the XC6200 were kept. Specifically, the connection mechanism does not support arbitrary bitstreams, making these FPGAs susceptible to damage by internal short circuits.

Recent work on evolvable circuits in commercial FPGAs has focused on the Virtex and Virtex-II architectures from Xilinx [66] and will extend its focus to Virtex-4 in the near future. Two main approaches have been used for evolving Virtex circuits: using virtual reconfigurable circuits [67] and partially reconfiguring the FPGA.

Virtual reconfiguration

Two solutions were used in order to replace the obsolete XC6200 family: implementing an ASIC evolvable circuit (only achievable by some privileged groups, summarized in Section 33.5.3) and building a reconfigurable circuit on top of another reconfigurable circuit (i.e., a virtual reconfigurable device [53]). The concept of a virtual reconfigurable circuit is depicted in Figure 33.9, where a reconfigurable neuron cell constitutes the device's basic logic cell.

In the beginning, the most intuitive method was to reconstruct the XC6200 architecture. At the University of York, a virtual XC6200 CLB was implemented in Virtex FPGAs [68, 69]. Slorach and Sharman [54] also used virtual XC6200 cells in the Xilinx XC4010 and Altera EPF6010A, evolving configuration bitstreams that configured not the FPGA itself but the virtual XC6200 CLBs. Afterward, other research groups developed different reconfigurable architectures with enhanced features, several of which had the goals of flexibility and easy reconfiguration [54–59, 70–72]. For example, Sekanina and Drabek [70] developed a virtual reconfigurable cell called a *functional block* (FB) and used an array of FBs for image compression. Durbeck and Macias [71] implemented an 8×8 cell matrix using a Xilinx Spartan-2 FPGA.

With this approach came the possibility of designing any desired reconfigurable fabric. In most cases the architecture consists of a fine-grained cellular array in which a general-purpose evolvable architecture is proposed. However,

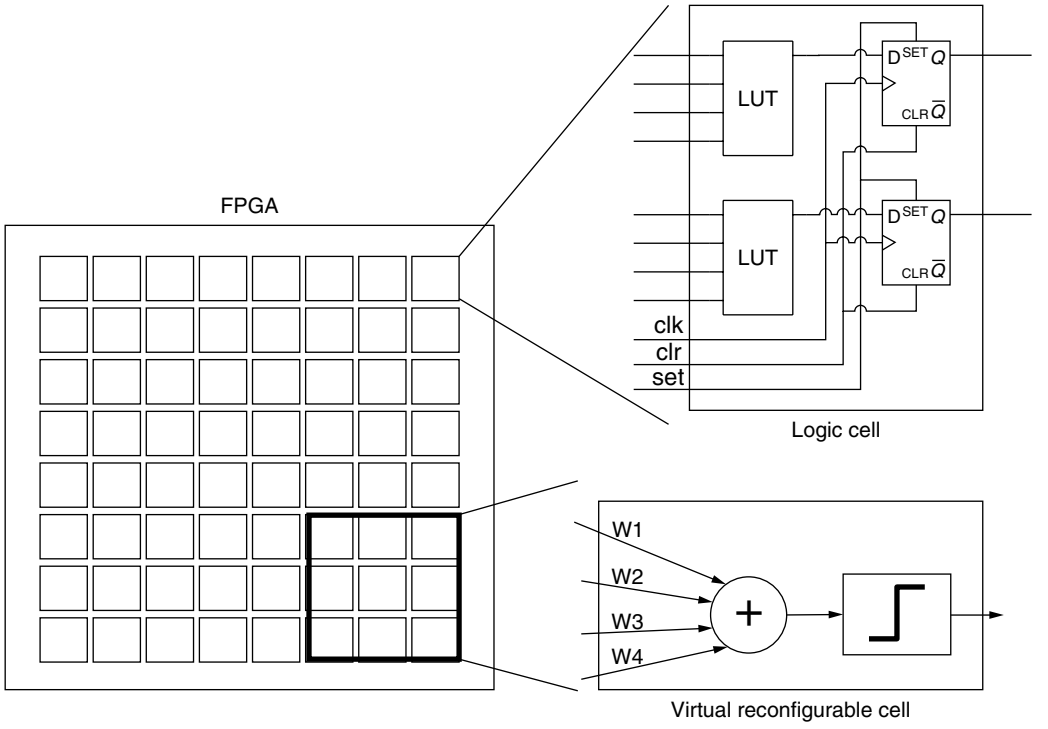

FIGURE 33.9 ■ A virtual reconfigurable circuit with a reconfigurable neuron.

problem-oriented reconfigurable fabrics can use coarser-grained architectures, where a reduced set of features is evolved.

Dynamic partial reconfiguration

In addition to the Xilinx XC6200, other commercial platforms have been partially reconfigured for evolving circuits, with the main focus on the Xilinx Virtex families. However, there are two main issues in evolving circuits by partially reconfiguring Virtex architectures. The first is the size of their configuration bitstreams, which implies a huge search space for the EA. The second is the generation of invalid bitstreams—that is, bitstreams that cause internal contentions. Different solutions to these problems have been suggested.

Haddow and Tufte proposed a two-dimensional array of Sblocks [72], each containing a flip-flop, a 5-input LUT, and some routing resources. Sblocks provide a reduced configurability compared to Virtex cells in order to reduce the search space size and to guarantee contention-free configurations. Even though the Sblock array is virtually reconfigurable, the functionality is reconfigured by partially reconfiguring a Virtex FPGA. Haddow and Tufte used a partial bitstream for reconfiguring only the LUT contents.

At the University of York, JBits [73] has been used for evolving circuits. JBits is a Java API for describing circuits and manipulating configuration bitstreams. It allows safe generation of partial bitstreams, permitting the modification of internal modules in the FPGA design. At York, LUT contents have been mapped from a genome for evolving simple combinatorial functions [74], fault tolerance circuits [69], and robot controllers for obstacle avoidance [75]. Also using JBits, Levi and Guccione from Xilinx developed a tool called GeneticFPGA [76], which translates a configuration bitstream from a chromosome, making it easy to generate legal bitstreams.

Even though JBits provides interesting features for EHW, it has several limitations, such as the impossibility of running on an embedded platform (for on-chip evolution), dependence on supported FPGA families and supported boards, incompatibility with other hardware description languages (HDLs), and limited support from Xilinx, mainly reflected in insufficient documentation.

Several ways to overcome these limitations have been proposed at the EPFL. Upegui and Sanchez [52] summarize three techniques for EHW by partially reconfiguring Virtex and Virtex-II families dynamically, without using JBits. The first is a coarse-grained high-level solution based on the modular partial reconfiguration flow proposed by Xilinx [77]. It defines large evolvable functions, implemented as modules, that are well suited for architecture exploration [27].

The second and third techniques are fine-grained low-level solutions. In both of the cases, hard-macros are used to define an evolvable component. Then by placing the hard-macros they modify, the bitstream partially reconfigures components of the hard macros. The second technique uses the difference-based partial reconfiguration flow proposed by Xilinx [77]. The third technique directly manipulates the bitstream in a manner similar to the XC6200, by adding some constraints (only LUT and multiplexer configuration modifications are allowed). These techniques are well suited for fine-tuning. With the difference-based approach, Mermoud et al. [25] report the intrinsic evolution of a fuzzy classifier; and with the bitstream manipulation, they report a complete evolution of cellular automata [42] and Boolean networks [43].

33.5.3 Custom Evolvable FPGAs

One of the more recent evolvable chips is the POEtic tissue [78,79], a computational substrate optimized for the implementation of digital systems inspired by the POE model presented in the introduction to this chapter. The POEtic tissue is a self-contained, flexible physical substrate designed (1) to interact with the environment through spatially distributed sensors and actuators; (2) to develop and adapt its functionality through a process of evolution, growth, and learning to a dynamic and partially unpredictable environment; and (3) to self-repair parts damaged by aging or environmental factors in order to remain viable and retain the same functionality.

The POEtic tissue is composed of a two-dimensional array of POEtic cells, each designed as a 3-layer structure following the three axes of bio-inspiration (Figure 33.10):

- The phylogenetic layer acts on a cell's genetic material. It can be used to find and select the genes of the cells for the genotype layer, which is conceptually the simplest of the three tissue layers as it is mainly a memory containing the genetic information of the organism.
- Ontogeny concerns the development of the individual and thus the mapping or configuration layer of the cell, which implements cellular differentiation and growth. In addition, it has an impact on the system as a whole for self-repair. The configuration layer selects which gene will be expressed depending on a user-defined differentiation algorithm.
- The epigenetic axis modifies the behavior of the organism during its operation and is therefore best applied to the phenotype, which is probably the most application-dependent layer. If the final application is a neural network, the phenotype layer will consist of an artificial neuron.

A key aspect of the applicability of the POEtic tissue, in addition to its architecture, is its reconfigurability. A molecule can be partially reconfigured by an on-chip microprocessor or by neighbor molecules. For EHW, this feature is

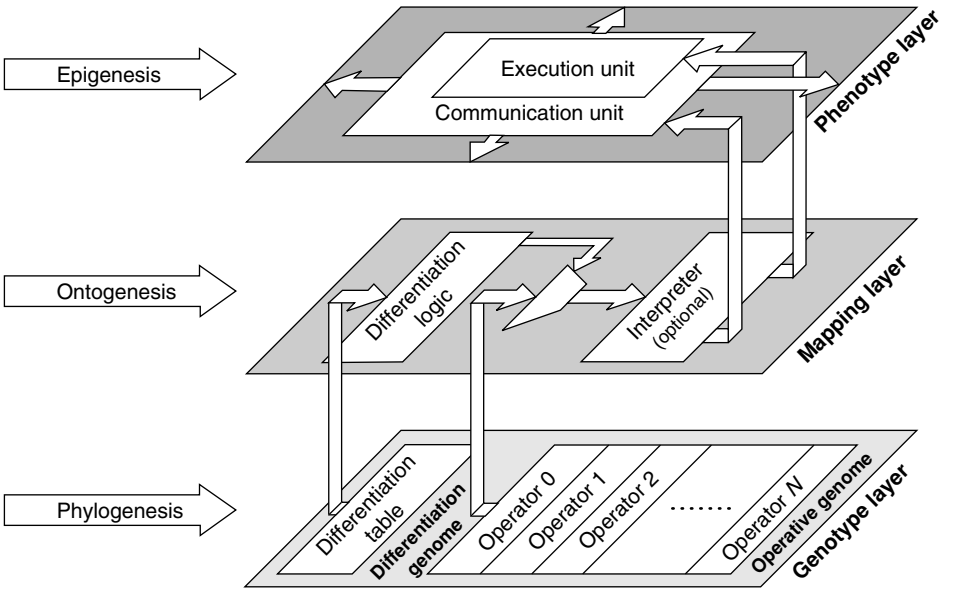

FIGURE 33.10 ■ The organizational layers of the POEtic cell.

very important in terms of execution time. Because only two clock cycles are needed for a write, and three words of 32 bits define a complete molecule, the configuration of the entire array (or a part of it) is very fast. In comparison with commercial FPGAs, such as the Virtex-II, in which at least a full configuration frame must be sent each time, reconfiguration takes place in parallel, allowing a huge speedup.

A distinctive feature of the POEtic tissue is its two-dimensional array of routing units that implement a dynamic routing algorithm [80]. It is used for intercellular communication, allowing the tissue to dynamically create paths between cells. The dynamic routing can be performed by a distributed algorithm [80] or by the on-chip processor.

Another very important circuit is the evolvable LSI chip developed by Higuchi's group [81]. It includes a GA unit and has the ability to process two chromosomes in parallel. Higuchi's group is famous for the large number of applications implemented in their chips [82, 83]. They have implemented an adaptive prosthetic hand controller [84, 85] that can adapt to the user's electromyographic signals in less than 10 minutes with a much more compact circuit than required with a neural network (before that, the user had to adapt to the hand instead of the hand to the user, requiring more than a month of training). They have also evolved data compressors for electrophotographic printing [86, 87], often attaining compression ratios twice those obtained with international standard compression algorithms such as Lempel-Ziv, JBIG, and JBIG2. It must be noted that Higuchi's applications often finish as part of a commercial product. Other interesting applications implemented by the same group include robot navigation controllers [88] and low-power integrated circuits [89].

This chapter focused primarily on evolution for digital devices; however, several platforms have been proposed for analog and mixed-signal circuit evolution. At the Jet Propulsion Laboratory of the California Institute of Technology, a field-programmable transistor array (FPTA) [90] has been developed that is the basis of the Standalone Board-level Evolvable System (SABLES) [91]. Layzell [92] proposed the evolvable motherboard: a diagonal matrix of analog switches connected to up to six plug-in daughter boards, which contain the desired basic elements for evolution.

33.6 CONCLUSIONS AND FUTURE DIRECTIONS

EHW has been shown to be effective at finding solutions [82, 83] for real-world applications. Additionally, some solutions have proven to perform better than their engineered counterparts [83, 89, 93]. On the other hand, EHW generally performs poorly, as a system-level solution: Microprocessor architectures, for example, are not among evolution results. As a matter of fact, evolution works better when the target is a complex cellular architecture: cellular automata, neural networks, or gate arrays.

If we look at the EHW work carried so far, we find many common characteristics spanning most current systems that often differ from biological evolution (this difference is not necessarily disparaging):

- Evolution pursues a predefined goal: The design of an electronic circuit is subject to precise specifications. On finding the desired circuit, the evolutionary process terminates.
- The population has no material existence. At best, in what has been called intrinsic and complete evolution, there is one circuit available onto which individuals from the population are loaded *one at a time* to evaluate their fitness.
- The absence of a real population in which individuals coexist simultaneously entails notable difficulties in the realization of interactions between "organisms." This usually results in a completely independent fitness calculation, contrary to nature, which exhibits a coevolutionary scenario.
- The different phases of evolution are carried out sequentially, controlled by a central unit.

These limitations suggest that the simple application of EAs to hardware design is not enough and that future research in EHW must not be limited to exploration of architectures and substrates; there is also much to do at the algorithmic level. Human-made adaptable systems are still far from exhibiting an adaptation comparable to living beings, and even though we have yet to attain circuits of equivalent complexity, limitations are not just a matter of magnitude. Only by modeling together the three axes of life (phylogeny, ontogeny, and epigenesis) will we be able to build systems featuring naturelike adaptation.

Future trends in nanotechnology are also guiding us toward "Avogadro computers"—that is, massively parallel devices with 10^{23} transistors. What to do with such huge number of transistors, and how to use, interconnect, and program them, goes beyond present engineering knowledge; however, EHW architectures and algorithms arise as a promising solution for dealing with the design complexity of these machines.

In this chapter we focused on evolving silicon circuits, which constitute the main developments achieved by the EHW community. However, other types of substrates have been evolved that extend the domain and represent new directions for evolvable hardware. For example, NASA researchers have been working on evolving antennas for space missions [94, 95]. Miller and Downing are currently working on evolving liquid crystals (LC) [96]—by applying electric fields mapped from a genome, they modify the LC molecular alignment to implement a desired function. Molecular circuit design is another promising evolvable substrate. Masiero et al. [97] report the use of a GA for tuning component parameters in a molecular circuit. Quantum circuit synthesis, too, is a potential field for EHW [98], given that designing circuits in such a substrate will require new design paradigms.

References

[1] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, T. Furuya. Evolving hardware with genetic learning: A first step towards building a Darwin Machine. From animals to animals 2. *Proceedings of the International Conference on Simulation of Adaptive Behavior*, 1993.

[2] H. de Garis. Evolvable hardware: Genetic programming of a Darwin Machine. *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms,* 1993.

[3] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, A. Stauffer. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. *Evolvable Systems: From Biology to Hardware, LNCS* 1259, 1997.

[4] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, A. Stauffer. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation* 1(1), 1997.

[5] S. Mitra, Y. Hayashi. Neuro-fuzzy rule generation: Survey in soft computing framework. *IEEE Transactions on Neural Networks* 11(3), 2000.

[6] T. Bäck. *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*, Oxford University Press, 1996.

[7] D. B. Fogel. *Evolutionary Computation: Toward a New Philosophy of Machine Intelligence*, 2nd ed., IEEE Press, 2000.

[8] J. R. Koza. *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press, 1992.

[9] M. Mitchell. *An Introduction to Genetic Algorithms*, MIT Press, 1996.

[10] M. D. Vose. *The Simple Genetic Algorithm: Foundations and Theory*, MIT Press, 1999.

[11] J. Pinter. *Global Optimization in Action (Continuous and Lipschitz Optimization: Algorithms, Implementations and Applications)*, Kluwer Academic Press, 1996.

[12] E. Sanchez, M. Tomassini. Towards evolvable hardware. *LNCS* 1062. Springer-Verlag, 1996.

[13] Y. Liu. Evolvable systems: from biology to hardware. *Proceedings of the Fourth International Conference, ICES,* October 2001.

[14] A. M. Tyrrell, P. C. Haddow, J. Torresen. Evolvable systems: From biology to hardware. *Proceedings of the 5th International Conference, LNCS,* March 2003.

[15] J. M. Moreno, J. Madrenas, J. Cosp. Evolvable systems: From biology to hardware. *Proceedings of the Sixth International Conference, ICES 2005,* September 2005.

[16] T. Higuchi, M. Iwata, W. Liu. Evolvable systems: From biology to hardware. *Proceedings of the First International Conference*, October 7–8, 1996. *LNCS* 1259, Heidelberg: Springer-Verlag, 1997.

[17] M. Sipper, D. Mange, A. Pérez-Uribe. Evolvable systems: From biology to hardware. *Proceedings of the Second International Conference*, September, *LNCS* 1478, Heidelberg: Springer, 1998.

[18] J. Miller. Evolvable systems: From biology to hardware. *Proceedings of the Third International Conference, ICES 2000,* April 17–19, 2000. *LNCS* 1801, Heidelberg: Springer, 2000.

[19] A. Stoica, D. Keymeulen, J. D. Lohn. *Proceedings of the First NASA/DOD Workshop on Evolvable Hardware,* July. IEEE Computer Society, 1999.

[20] A. Stoica, J. D. Lohn, R. Katz, D. Keymeulen, R. Zebulum. *Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware*, July. IEEE Computer Society, 2002.
[21] J. D. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, M. Ferguson. *Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware,* July. IEEE Computer Society, 2003.
[22] R. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. D. Lohn. A. Stoica. *Proceedings of the 2004 NASA/DOD Conference on Evolvable Hardware*, July 2004. IEEE Computer Society.
[23] J. D. Lohn, D. Gwaltney, G. Hornby, R. Zebulum, D. Keymeulen. A. Stoica. *Proceedings of the 2005 NASA/DOD Conference on Evolvable Hardware,* June 2005. IEEE Computer Society.
[24] X. Yao, T. Higuchi. Promises and challenges of evolvable hardware. *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews* 29(1), 1999.
[25] G. Mermoud, A. Upegui, C. A. Pena. E. Sanchez. A dynamically-reconfigurable FPGA platform for evolving fuzzy systems. *Computational Intelligence and Bioinspired Systems, LNCS* 3512, 2005.
[26] M. Murakawa, S. Yoshizawa, I. Kajitani, X. Yao, N. Kajihara, M. Iwata, T. Higuchi. The GRD chip: Genetic reconfiguration of DSPs for neural network processing. *IEEE Transactions on Computers* 48(6), 1999.
[27] A. Upegui, C. A. Peña-Reyes, E. Sanchez. An FPGA platform for on-line topology exploration of spiking neural networks. *Microprocessors and Microsystems* 29(5), 2005.
[28] H. Hemmi, J. Mizoguchi, K. Shimohara. Development and evolution of hardware behaviors. *Towards Evolvable Hardware, LNCS* 1062, 1996.
[29] J. R. Koza, F. H. Bennett, D. Andre, M. A. Keane. Synthesis of topology and sizing of analog electrical circuits by means of genetic programming. *Computer Methods in Applied Mechanics and Engineering* 186(2), 2000.
[30] J. W. Atmar. Speculation on the Evolution of Intelligence and Its Possible Realization in Machine Form, Ph.D. dissertation, New Mexico State University, Las Cruces, 1976.
[31] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, Y. Hirao, F. T. Furuya, B. Manderick. Evolvable hardware and its application to pattern recognition and fault-tolerant systems. *Towards Evolvable Hardware, LNCS* 1062, 1996.
[32] A. Thompson. Silicon evolution. *Proceedings of Genetic Programming*, J. R. Koza et al. (eds.), MIT Press, 1996.
[33] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. *Evolvable Systems: From Biology to Hardware, LNCS* 1259, 1997.
[34] Xilinx, Inc. *The Programmable Logic Data Book*, 1996.
[35] G. K. Venayagamoorthy, V. G. Gudise. Swarm intelligence for digital circuits implementation on field-programmable gate array platforms. *Proceedings of the 2004 NASA/DOD Conference on Evolvable Hardware*, July 2004.
[36] B. C. Kahne. *A Genetic Algorithm-Based Place-and-Route Compiler for a Run-time Reconfigurable Computing System*, Master's thesis, Virginia Polytechnic Institute and State University, Blacksburg, VA, 1997.
[37] T. A. Ly, J. T. Mowchenko. Applying simulated evolution to high-level synthesis. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 12(3), 1993.
[38] A. Thompson, I. Harvey, P. Husbands. Unconstrained evolution and hard consequences. *Towards Evolvable Hardware, LNCS*, 1996.

[39] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, T. Higuchi. Hardware evolution at function level. *Parallel Problem Solving from Nature (PPSN IV)*, *LNCS* 1141, 1996.
[40] M. Iwata, I. Kajitani, H. Yamada, H. Iba, T. Higuchi. A pattern recognition system using evolvable hardware. *Parallel Problem Solving from Nature (PPSN IV)*, *LNCS* 1141, 1996.
[41] P. Haddow, G. Tufte. Evolving a robot controller in hardware. *Proceedings of the Norwegian Computer Science Conference*, 1999.
[42] A. Upegui, E. Sanchez. On-chip and on-line self-reconfigurable adaptable platform: The non-uniform cellular automata case. *Proceedings of the 20th IEEE International Parallel and Distributed Processing Symposium*, 2006.
[43] A. Upegui, E. Sanchez. Evolving hardware with self-reconfigurable connectivity in Xilinx FPGAs. *Proceedings of the First NASA/ESA Conference on Adaptive Hardware and Systems*, 2006.
[44] K. Glette, J. Torresen. A flexible on-chip evolution system implemented on a Xilinx Virtex-II Pro device. *Evolvable Systems: From Biology to Hardware, LNCS* 3637, 2005.
[45] M. Goeke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, M. Tomassini. Online autonomous evolware. *Evolvable Systems: From Biology to Hardware, LNCS* 1259, 1997.
[46] T. Toffoli, N. Margolus. *Cellular Automata Machines: A New Environment for Modeling*. MIT Press Series in Scientific Computation, 1987.
[47] M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach*, Springer, 1997.
[48] R. A. Brooks. New approaches to robotics. *Science* 253, 1991.
[49] Y. U. Cao, A. S. Fukunaga, A. B. Kahng. Cooperative mobile robotics: Antecedents and directions. *Autonomous Robots* 4(1), 1997.
[50] R. Moeckel, C. Jaquier, K. Drapel, E. Dittrich, A. Upegui, A. Ijspeert. YaMoR and Bluemove: An autonomous modular robot with Bluetooth interface for exploring adaptive locomotion. *Proceedings of the 8th International Conference on Climbing and Walking Robots* (*CLAWAR)*, 2005.
[51] T. S. Ray. An approach to the synthesis of life. *Artificial Life II, SFI Studies in the Sciences of Complexity* 10, 1992.
[52] A. Upegui, E. Sanchez. Evolving hardware by dynamically reconfiguring Xilinx FPGAs. *Evolvable Systems: From Biology to Hardware, LNCS* 3637, 2005.
[53] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations*, Springer, 2004.
[54] C. Slorach, K. Sharman. The design and implementation of custom architectures for evolvable hardware using off-the-shelf programmable devices. *Evolvable Systems: From Biology to Hardware, LNCS*, 2000.
[55] Y. Zhang, S. Smith, A. Tyrrell. Digital circuit design using intrinsic evolvable hardware. *Proceedings of the 2004 NASA/DOD Conference on Evolvable Hardware*, July 2004.
[56] L. Sekanina, S. Friedl. On routine implementation of virtual evolvable devices using COMBO6. *Proceedings of the 2004 NASA/DOD Conference on Evolvable Hardware*, July 2004.
[57] K. Vinger, J. Torresen. Implementing evolution of FIR-filters efficiently in an FPGA. *Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware*, July 2003.
[58] L. Sekanina. Towards evolvable IP cores for FPGAs. *Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware*, July 2003.

[59] P. C. Haddow, G. Tufte. An evolvable hardware FPGA for adaptive hardware. *Proceedings of the 2000 Congress on Evolutionary Computation*, 2000.
[60] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, M. Tomassini. The firefly machine: Online evolware. *Proceedings of the IEEE International Conference on Evolutionary Computation*, 1997.
[61] Xilinx, Inc. *The XC6200 Data Sheet v.1.7*, 1996.
[62] A. Thompson, P. Layzell. Evolution of robustness in an electronics design. *Evolvable Systems: From Biology to Hardware, LNCS* 1801, 2000.
[63] D.-W. Lee, C.-B. Ban, K.-B. Sim, H.-S. Seok, L. Kwang-Ju, B.-T. Zhang. Behavior evolution of autonomous mobile robot using genetic programming based on evolvable hardware. *Proceeding of the 2000 IEEE International Conference on Systems, Man, Cybernetics*, 2000.
[64] J. R. Koza, F. H. Bennett, J. Hutchings, S. L. Bade, M. A. Keane, D. Andre. Evolving sorting networks using genetic programming and rapidly reconfigurable field-programmable gate arrays. *Workshop on Evolvable Systems. International Joint Conference on Artificial Intelligence*, 1997.
[65] J. Dumoulin, J. A. Foster, J. F. Frenzel, S. McGrew. Special purpose image convolution with evolvable hardware. *Real-World Applications of Evolutionary Computing, EvoWorkshops 2000, LNCS,* 2000.
[66] Xilinx, Inc. *Virtex-II Platform FPGA User Guide* (*www.xilinx.com*), March 2005.
[67] L. Sekanina. Virtual reconfigurable circuits for real-world applications of evolvable hardware. *Evolvable Systems: From Biology to Hardware, LNCS* 2606, 2003.
[68] G. Hollingworth, S. Smith, A. Tyrrell. Safe intrinsic evolution of Virtex devices. *Proceedings of the Second NASA/DoD Workshop on Evolvable Hardware*, 2000.
[69] R. O. Canham, A. Tyrrell. Evolved fault tolerance in evolvable hardware. *Proceedings of the Congress on Evolutionary Computation*, 2002.
[70] L. Sekanina, V. Drabek. The concept of pseudo evolvable hardware. *Proceedings of the IFAC Workshop on Programmable Devices and Systems*, 2000.
[71] L. Durbeck, N. J. Macias. Defect-tolerant, fine-grained parallel testing of a cell matrix. *Proceedings of SPIE ITCom* 4867, 2002.
[72] P. Haddow, G. Tufte. Bridging the genotype-phenotype mapping for digital FPGAs. *Proceedings of the Third NASA/DoD Workshop on Evolvable Hardware*, 2001.
[73] S. A. Guccione, D. Levi, P. Sundararajan. JBits: A Java-based interface for reconfigurable computing. *Proceedings of the Second Annual Military and Aerospace Applications of Programmable Devices and Technologies Conference*, 1999.
[74] G. Hollingworth, S. Smith, A. Tyrrell. The intrinsic evolution of Virtex devices through Internet reconfigurable logic. *Evolvable Systems: From Biology to Hardware, LNCS* 1801, 2000.
[75] A. M. Tyrrell, R. A. Krohling, Y. Zhou. Evolutionary algorithm for the promotion of evolvable hardware. *IEE Proceedings—Computers and Digital Techniques* 151(4), 2004.
[76] D. Levi, S. A. Guccione. Genetic FPGA: Evolving stable circuits on mainstream FPGA devices. *Proceedings of the First NASA/DOD Workshop on Evolvable Hardware*, 1999.
[77] Xilinx, Inc. *XAPP 290: Two Flows for Partial Reconfiguration: Module Based or Difference Based* (*www.xilinx.com*), September 2004.
[78] Y. Thoma, E. Sanchez. A reconfigurable chip for evolvable hardware. *Proceedings of the Genetic and Evolutionary Computation Conference*, 2004.

[79] Y. Thoma, G. Tempesti, E. Sanchez, J.M.M. Arostegui. POEtic: An electronic tissue for bio-inspired cellular applications. *Biosystems* 76(1–3), 2004.
[80] Y. Thoma, E. Sanchez, J.M.M. Arostegui, G. Tempesti. A dynamic routing algorithm for a bio-inspired reconfigurable circuit. *Proceedings of the International Conference on Field-Programmable Logic and Applications* 2778, 2003.
[81] M. Iwata, I. Kajitani, Y. Liu, N. Kajihara, T. Higuchi. Implementation of a gate-level evolvable hardware chip. *Evolvable Systems: From Biology to Hardware, LNCS* 2210, 2001.
[82] T. Higuchi, M. Iwata, H. Sakanashi, E. Takahashi, M. Murakawa, I. Kajitani. Dynamic adaptive devices and their applications. *Bulletin of the Electrotechnical Laboratory, Special Issue: RWC Research Toward Realization of Real World Intelligence* 64(4/5), 2000.
[83] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, M. Salami, N. Kajihara, N. Otsu. Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation* 3(3), 1999.
[84] I. Kajitani, M. Iwata, M. Harada, T. Higuchi. A myoelectric controlled prosthetic hand with an evolvable hardware LSI chip. *Technology and Disability, Special Issue: Advances in the Control of Prosthetic Arms* 15(2), 2003.
[85] I. Kajitani, T. Hoshino, N. Kajihara, M. Iwata, T. Higuchi. An evolvable hardware chip and its application as a multi-function prosthetic hand controller. *Proceedings of the 16th National Conference on Artificial Intelligence*, 1999.
[86] H. Sakanashi, M. Iwata, T. Higuchi. Evolvable hardware for lossless compression of very high resolution bi-level images. *IEE Proceedings—Computers and Digital Techniques* 151(4), 2004.
[87] H. Sakanashi, M. Iwata, D. Keymulen, M. Murakawa, I. Kajitani, M. Tanaka, T. Higuchi. Evolvable hardware chips and their applications. *Proceedings of the International Conference on Systems, Man, and Cybernetics*, 1999.
[88] D. Keymeulen, M. Iwata, Y. Kuniyoshi, T. Higuchi. Online evolution for a self-adapting robotic navigation system using evolvable hardware. *Artificial Life* 4, 1998.
[89] E. Takahashi, M. Murakawa, Y. Kasai, T. Higuchi. Power dissipation reductions with genetic algorithms. *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, 2003.
[90] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, A. Thakoor. Reconfigurable VLSI architectures for evolvable hardware: From experimental field-programmable transistor arrays to evolution-oriented chips. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems* 9(1), 2001.
[91] A. Stoica, R. Zebulum, M. Ferguson, D. Keymeulen, V. Duong. Evolving circuits in seconds: Experiments with a stand-alone board-level evolvable system. *Proceedings of the 2002 NASA/DOD Conference on Evolvable Hardware,* July 2002.
[92] P. Layzell. A new research tool for intrinsic hardware evolution. *Evolvable Systems: From Biology to Hardware*, *LNCS*, 1998.
[93] L. Sekanina, R. Ruzicka. Easily testable image operators: The class of circuits where evolution beats engineers. *Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware*, July 2003.
[94] J. Lohn, J. Crawford, A. Globus, G. Hornby, W. Kraus, G. Larchev, A. Pryor, D. Srivastava. Evolvable systems for space applications. *Proceedings of the International Conference on Space Mission Challenges for Information Technology*, 2003.

[95] J. Lohn, D. Linden, G. Hornby, W. Kraus, A. Rodriguez-Arroyo. Evolutionary design of an X-band antenna for NASA's space technology 5 mission. *Proceedings of the 2003 NASA/DoD Conference on Evolvable Hardware*, 2003.

[96] J. F. Miller, K. Downing. Evolution in materio: Looking beyond the silicon box. *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware*, 2002.

[97] L. P. Masiero, M. Pacheco, C. R. Hall, C. Santini. Molecular circuit design. *Proceedings of the 2005 NASA/DOD Conference on Evolvable Hardware*. June–July, 2005.

[98] L. Spector, H. Barnum, H. J. Bernstein, N. Swamy. Quantum computing applications of genetic programming. *Advances in Genetic Programming*, MIT Press, 1999.