From Darwin to Darware: Why computer engineers have started listening to Mother (Nature).

Quo Vadis Evolvable Hardware?

Moshe Sipper, Daniel Mange, and Eduardo Sanchez

Whether one looks outside the window or stands in front of a mirror, nature's work is evident in all its glory. Be it a pine tree, a flea, or a human being, nature has designed highly complex machines, the construction of which is still well beyond our current engineering capabilities. Nature is the grandest engineer known to man—though a blind one at that: her designs come into existence through the slow process, over millions of years, known as evolution by natural selection [2]. As put forward by Charles Darwin in his 1859 masterpiece, *On the Origin of Species*, evolution is based on "...one general law, leading to the advancement of all organic beings, namely, multiply, vary, let the strongest live and the weakest die." [1].

The evolutionary process is based on four basic principles:

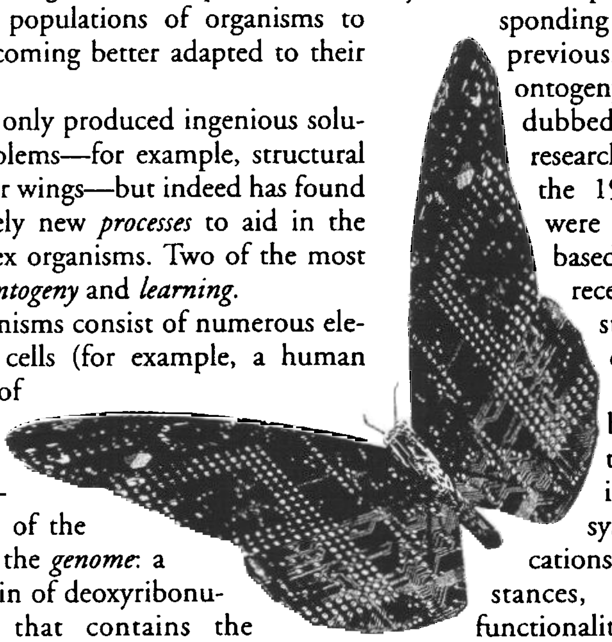
- Individual organisms vary in viability in the environments that they occupy.
- This variation is heritable.
- Individuals tend to produce more offspring than can survive on the limited resources available in the environment.
- In the ensuing struggle for survival, the individuals best adapted to the environment are the ones that will survive to reproduce.

The continual workings of this process over the millennia cause populations of organisms to change, generally becoming better adapted to their environments.

Evolution has not only produced ingenious solutions to specific problems—for example, structural designs such as eyes or wings—but indeed has found (and founded) entirely new *processes* to aid in the emergence of complex organisms. Two of the most important ones are *ontogeny* and *learning.*

Most natural organisms consist of numerous elemental units called cells (for example, a human being is composed of approximately sixty trillion cells). Each and every cell contains the entire plan of the organism, known as the *genome*: a one-dimensional chain of deoxyribonucleic acid (DNA) that contains the instructions necessary for the making of the individual. Known also as multicellular organisms, they develop through the process of ontogeny, which involves the successive divisions of a fertilized mother cell, ultimately resulting in a complete being. The genome—or *genotype*—thus gives rise to the so-called *phenotype*: the mature organism that emerges from the ontogenetic process. The genotype-phenotype distinction is fundamental in nature: while it is the phenotype that is subjected to the survival battle, it is the genotype that retains the evolutionary benefits.

Ontogeny can be viewed as one of nature's tricks for combating the information explosion inherent in the definition of a complex design: rather than define a one-to-one plan (a homunculus), nature uses a compressed definition that is more akin to a recipe or a construction program. Further compression can be attained by introducing the process of learning. The organism that emerges from the ontogenetic process is not fully equipped to handle every aspect of its daily existence, but rather proceeds to improve its behavior by learning how to cope with new situations as they are encountered (this is most notable among mammals). The introduction of learning reduces the amount of information that needs to be encoded in the genome.



Over the past few years a growing number of computing scientists and engineers have been turning to nature, seeking inspiration to augment the capabilities of their artificial systems. In a recent paper [10], Sipper et al. noted that such *bio-inspired* systems can be partitioned along three axes, corresponding to the three processes mentioned previously: phylogeny (evolution), ontogeny, and epigenesis (learning); they dubbed this the *POE* model. While research in this area can be traced back to the 1950s, when the first computers were used to carry out simulations based on these ideas, much of the recent work is centered on the construction of actual hardware devices.

The ultimate goal of bio-inspired system engineers is to create more adaptive systems, in which "adaptive" refers to a system's ability to undergo modifications according to changing circumstances, thus ensuring its continued functionality. One often speaks of an environment and of a system's adjustment to changing environmental conditions. The fact that hardware evolves is perhaps not surprising in and of itself, after all, the Intel 80*x*86 "species" evolved from a mere 29,000 transistors in 1978 (the 8086) to 7,500,000 transistors in 1997 (the Pentium II), not to mention the many functionalities that have been added over the years.[1] Thus, the processor has adapted to its environment, which is, ultimately, the users. This evolutionary process involves what is sometimes referred to as the "hand of God": the engineer who designs the chip. With bio-inspired hardware the aim is to reduce the amount of human design necessary. The ultimate goal is to build a system that will evolve, develop (in an ontogenetic sense), and learn—in short, adapt—on its own, that is, with no human intervention. As an example,

[1]Intel chip history: www.visionbs.com/intel.htm

imagine some future chip family that evolves on its own from the "8086" epoch to the "Pentium" epoch. Obviously, it will need to be equipped with the ability to accrue additional material resources (as with natural evolution); what is far from obvious at this point is how to induce this kind of ability. In summary, we want to replace the human engineer with a blind yet potentially more powerful one.

There is still a long road to travel before we can create an autonomously evolving system. Our intent in this article is to describe a number of current milestones, and to trace some of the possible developments of the near future.

From Carbon to Silicon: Configurable Circuits

Within the field of bio-inspired hardware, a major enabling technology is that of configurable circuits, and especially field-programmable gate arrays, or FPGAs. FPGAs, which have been coming of age over the past few years, are large, fast integrated circuits that can be modified or configured at almost any point by the end user [12].

The primary distinction that this technology brings about is that between *programmable* circuits and *configurable* ones [7]. A programmable circuit ceaselessly iterates through a three-phase loop, where an instruction is first *fetched* from memory, after which it is *decoded*, then to be passed on to the final *execute* phase. The execute phase may require several clock cycles; the process is then repeated for the next instruction, and so on. A configurable circuit, on the other hand, can be regarded as having a single, non-iterative fetch phase: the so-called *configuration string*, fetched from memory, requires no further interpretation, and is directly used to configure the hardware. No further phases or iterations are needed, as the circuit is now configured for the task at hand. The ability to control the hardware in such a direct manner is a double-edged sword: the user is able to der range with the price to be paid being a more arduous design task.

Within the domain of configurable computing one can distinguish between two types of configuration strings: static and dynamic [7]. A static configuration string, aimed at configuring the circuit to perform a given function, is loaded once at the outset, after which it does not change during the execution of the task at hand. Static applications are mainly aimed at attaining the classic goal in computing: that of improving performance, either in terms of speed, resource utilization, or area usage. Dynamic configurability involves a configuration string that can change during execution of the task at hand. Dynamic systems are able to undergo modifications according to changing circumstances, thus continuing to function within their dynamic environments. As such, they represent an excellent substrate for implementing the adaptive systems discussed in this article.

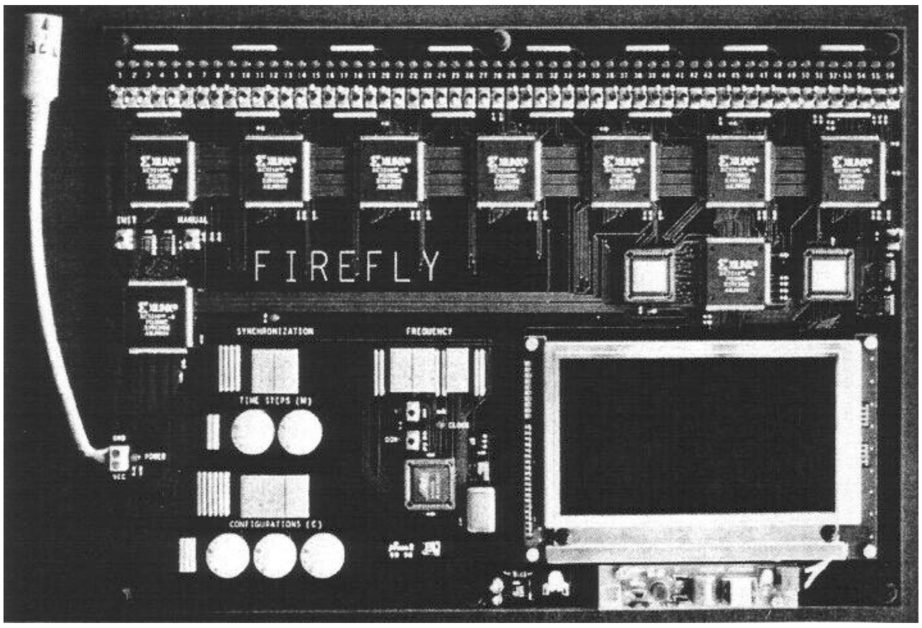


Figure 1. The Firefly evolware board. The system is an evolving, one-dimensional, non-uniform cellular automaton. Each of the 56 cells contains a genome that represents its rule table; these genomes are initialized randomly, to be subjected to evolution. The board contains the following components: (1) LED indicators of cell states (top), (2) switches for manually setting the initial states of cells (top, below LEDs), (3) Xilinx FPGA chips (below switches), (4) display and knobs for controlling two parameters ('time steps' and 'configurations') of the cellular programming algorithm (bottom left), (5) a synchronization indicator (middle left), (6) a clock pulse generator with a manually adjustable frequency from 0.1Hz to 1MHz (bottom middle), (7) an LCD display of evolved rule tables and fitness values obtained during evolution (bottom right), and (8) a power-supply cable (extreme left). (Note that the latter is the system's sole external connection.)

***The ultimate goal** of bio-inspired system engineers is to create more adaptive systems, in which "adaptive" refers to a system's ability to undergo modifications according to changing circumstances, thus ensuring its continued functionality.*

Two Examples of Current-Day Bio-Inspired Hardware

In the following section we will describe two dynamic systems inspired by two of the processes outlined previously: evolution and ontogeny. Learning hardware, as well as more in-depth expositions of the systems discussed herein can be found in the recent book by Mange and Tomassini [4].

Evolving hardware: The Firefly machine. The idea of applying the biological principle of natural evolution to artificial systems, introduced more than four decades ago, has experienced impressive growth in the past few years. Usually grouped under the term *evolutionary algorithms* or *evolutionary computation,* are the domains of genetic algorithms, evolution strategies, evolutionary programming, and genetic programming [6]. As a generic example of artificial evolution we consider genetic algorithms.

A genetic algorithm is an iterative procedure that involves a constant-size population of individuals, each one represented by a finite string of symbols—the genome—encoding a possible solution in a given problem space. This space, referred to as the search space, comprises all possible solutions to the problem at hand. The algorithm sets out with an initial population of individuals that is generated at random or heuristically. In every evolutionary step, known as a generation, the individuals in the current population are decoded and evaluated according to some predefined quality criterion, referred to as the fitness function. To form a new population (the next generation), individuals are selected according to their fitness and transformed via genetically inspired operators, of which the most well known are *crossover* ("mixing" two or more genomes to form novel offspring) and *mutation* (randomly flipping bits in the genomes). Iterating this evaluation-selection-crossover-mutation procedure, the genetic algorithm may eventually find an acceptable solution, that is, one with high fitness.

Evolutionary algorithms are common nowadays, having been successfully applied to numerous problems from different domains, including optimization, automatic programming, circuit design, machine learning, economics, immune systems, ecology, and population genetics, to mention a few.

One of the recent uses of evolutionary algorithms is in the burgeoning field of evolvable hardware, which involves, among others, the use of FPGAs as a platform on which evolution takes place [8, 10]. The Firefly machine is one such example; our goal in constructing it was to demonstrate a system in which *all* evolutionary operations (fitness evaluation, selection, crossover, and mutation) are carried out *online,* that is, in hardware [9, 10].

Firefly is based on the cellular automata model, a discrete dynamical system that performs computations in a distributed fashion on a spatially extended grid [9]. 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 time steps according to a *local, identical* interaction rule. The state of a cell at the next time step is determined by the current states of a surrounding neighborhood of cells. This transition is usually specified in the form of a rule table, delineating the cell's next state for each possible neighborhood configuration. The cellular array (grid) is n-dimensional, where n=1, 2, 3 is used in practice. Here, we consider a one-dimensional grid, in which each cell can be in one of two states (0 or 1), and has three neighbors (itself, and the cells to its immediate left and right); the rule table thus comprises eight bits since there are eight possible neighborhood configurations. Non-uniform cellular automata have also been considered; for these the local update rule need not be identical for all grid cells [9].

Based on the cellular programming evolutionary algorithm [9], we implemented an evolving, one-dimensional, non-uniform cellular automaton. Each

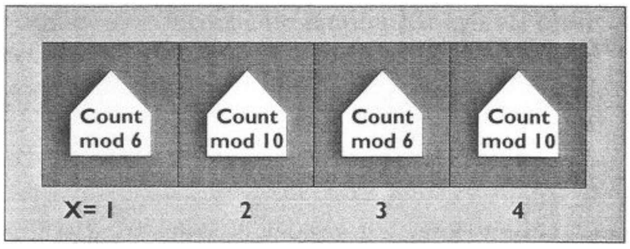


Figure 2a. Multicellular organization of the BioWatch.

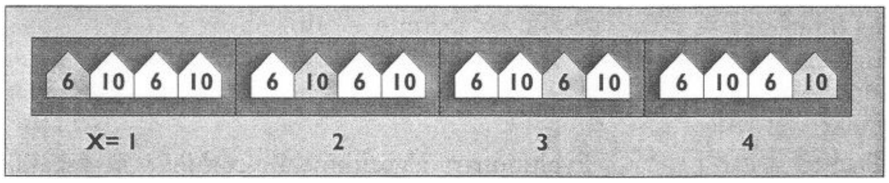


Figure 2b. Cellular differentiation of the BioWatch.

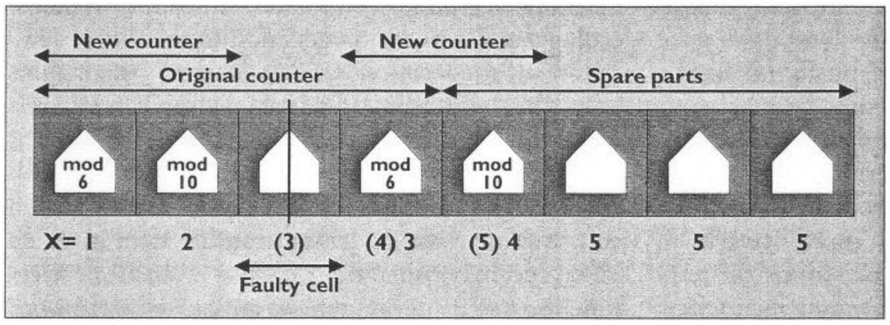


Figure 3. Self-repair of the BioWatch. Old coordinates are shown in parentheses.

of the system's 56 binary-state cells contains a genome that represents its rule table. These genomes are initialized at random, to be subjected to evolution. The system must evolve to resolve a global synchronization task: upon presentation of a random initial configuration of cellular states, the cellular automaton must reach, after a bounded number of time steps, a configuration whereupon the states of the cells oscillate between all 0s and all 1s on successive time steps (this may be compared to a swarm of fireflies, which evolve over time to flash on and off in unison). Due to the local connectivity of the system, this global behavior—which involves the entire grid—represents a difficult task. Nonetheless, by applying cellular programming, the system evolves (that is, the genomes change) such that the task is solved. The machine is depicted in Figure 1.

The Firefly machine exhibits complete online evolution; all its operations are carried out in hardware with no reference to an external computer. This demonstrates that evolving ware, or *evolware*, can be constructed [9]. Such evolware systems permit enormous gains in execution speed (for example, Firefly runs 1000 times faster than a simulation on a high-performance workstation). While the synchronization task is not a real-world application and was selected to act as a benchmark problem for our evolware demonstration, Firefly does open up interesting avenues for future research. Evolware machines that operate in an autonomous manner can be used to construct autonomous mobile robots, as well as for the construction of controllers for noisy, changing environments [10].

Ontogenetic hardware: The BioWatch. The BioWatch is one of the applications designed as part of the Embryonics (embryonic electronics) project, whose final objective is the development of very large-scale integrated circuits, which are capable of self-repair and self-replication [3, 4]. These two bio-inspired properties, characteristic of the living world, are achieved by transposing certain features of cellular organization in nature onto the two-dimensional world of integrated circuits in silicon.

The BioWatch is an artificial "organism" designed to count minutes (from 00 to 59) and seconds (from 00 to 59); it is thus a modulo-3600 counter. This organism is one-dimensional and is comprised of four cells with identical physical connections and an identical set of resources. The organization is multicellular (as with living beings), with each cell realizing a unique function, described by its gene (see Figure 2a).

The genome is the set of all the genes of the BioWatch, and each gene is a sub-program, characterized by a set of instructions and by its horizontal coordinate *X*. Storing the whole genome in each cell renders the cell universal, that is, capable of realizing any gene of the genome. This is another bio-inspired property: each of our (human) cells also contains the entire genome, though only part of it is used (for example, liver cells do not use the same genes as muscle cells). Depending on its position in the organism, each cell interprets the genome, extracting and executing the gene that configures it. The BioWatch thus performs what is known in biology as cellular differentiation (see Figure 2b).

Self-repair of an artificial organism allows partial reconstruction of the original device in case of a

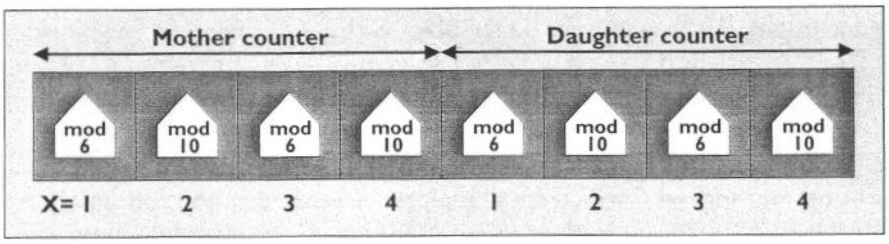


Figure 4. Self-replication of the BioWatch.

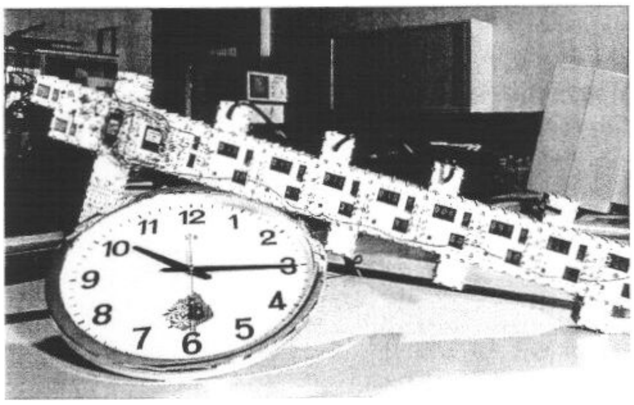


Figure 5. An eight-cell BioWatch. (Note: while our long-term objective is the design of very large-scale integrated circuits, each BioWatch cell is currently implemented in an Actel 1020 FPGA circuit and embedded within a small plastic box that serves as a demonstration module.)

minor fault. In the BioWatch, each cell performs one of two specific tasks: a modulo-6 or a modulo-10 count (see Figure 2a). Self-repair can be attained by dynamically reconfiguring the task executed by some of the cells. In order to implement this process, the BioWatch must have as many spare cells to the right of the array as there are faulty cells to repair (there are four spare cells in the example of Figure 3). Self-repair is achieved by bypassing the faulty cell and shifting to the right all or part of the original cellular array. The new coordinates, thus defined, lead to the dynamic reconfiguration of the task performed by the cell (modulo-6 or modulo-10 count).

Self-replication of an artificial organism completely reconstructs the original device in case of a major fault. In the BioWatch, the self-replication process rests on two assumptions: (1) there exists a sufficient number of spare cells to the right of the array (four in our example), and (2) the calculation of the coordinates produces a cycle (X=1 -> 2 -> 3 -> 4 -> 1 in Figure 4). As the same pattern of coordinates produces the same pattern of genes, self-replication can be easily accomplished if the microprogram of the genome, associated with the homogeneous network of cells, produces several instances of the basic pattern of coordinates.

With a larger number of cells it becomes possible to add the extensions needed for a practical use of the BioWatch: preserving the current time while self-repair is being effected, and setting and resetting the time (see Figure 5). It is also quite easy to introduce additional functions such as computing the date, keeping track of the day of the week, and handling leap years.

The Future of Bio-Inspired Hardware

According to current evolutionary theory, life on earth originated about 4 billion years ago, but evolved slowly for about 3.5 billion years; this is known as the pre-Cambrian period. Then, in a relatively short span of a few million years, beginning about 550 million years ago—the Cambrian period—a vast array of multicellular life abruptly emerged. This period is also known as the Cambrian explosion. The first advanced cell, which served as a basis for all higher life forms (eukaryotes), came into existence when some kind of host acquired as symbiotic partner a number of smaller components; these latter—known as organelles—originated as free-living bacteria. The host and the organelles entered into an endosymbiotic relationship, that is, symbiosis in which a symbiont dwells within the body of its symbiotic partner. (The endosymbiotic theory was put forward by Lynn Margulis in the early 1970s [5]; though controversial at the time, it has gained acceptance during the intervening years.)

The evolution of microprocessors, as effected by engineers (the "hand of God" discussed earlier in this article), seems to have followed a similar path. Units that were originally "free-living," situated outside the processor, were moved into it: the memory management unit (MMU), the floating point unit (FPU), and the cache memory, all of which were at first separate units, are found today within the processor itself. The latest development involves the addition of reconfigurable on-chip surfaces; for example, Triscend has recently announced a microprocessor chip with a 65% reconfigurable surface, with only 35% of the surface serving as a classical controller to manage the chip's operation [11].

Does this new "organelle" portend the upcoming "Cambrian explosion" of adaptive, bio-inspired hardware? The trend toward a merger of the classical-processor industry with the configurable-computing one will probably not only continue, but intensify in the future. While FPGAs currently represent only a small niche within the computer-hardware industry, this new development might elevate them to ubiquity. These new processors will find their way to every desktop, providing a base for adaptive machines. This might lead to an explosion of adaptive-hardware applications—computers that can change not only their inner workings but indeed their inner structure in response to changes in the environment.

The environment in question includes everything that the configurable processor contacts: other internal units within the computer, the user (or users), and the network. Future computers might thus be shipped only partially designed, to be then subjected to evolution in the field—that is, on the desktop. Furthermore, they might repair themselves upon suffering damage, as the BioWatch does. Ultimately, computers might evolve to improve their own performance—that is, they might go from the 8086 epoch to the 80286 epoch (as noted previously) without the intervention of a human engineer.

The road ahead is, however, still fraught with many obstacles. While the electronic support for increased adaptability might exist, there are still major unresolved issues, some of which we discuss here. For one thing, we have still not mastered this technology: configuring a configurable processor is at least as hard (if not harder) than programming a classical processor, and the available tools for configuration are far behind those for programming.

The application of bio-inspired methodologies might serve to offset this problem by obviating the need for human design altogether. One must remember, however, that natural evolution has required a huge amount of resources (in terms of time and space), and most of the avenues explored have been dead ends (there are many more extinct species than surviving ones). The ideal path might thus be somewhere between complete design and complete autonomy: initial human design followed by further autonomous adaptation.

Another major issue is that of hierarchy. Nature has evolved elemental building blocks, out of which she constructs more complex systems: molecules combine to form cells, cells combine to form tissues, tissues combine to form organisms, and organisms combine to form societies. Such a layered approach has not escaped computing practitioners in their battle against increasing complexity (for example, high-level languages have been introduced, hiding much of the lower-level complexity). We have addressed this issue within the Embryonics project framework by adding a molecular level to the extant cellular level: each cell is composed of yet finer elements—referred to as molecules—allowing self-repair to take place within the cell as well as outside it; the resulting system is more efficient in its use of chip surface (only extracellular self-repair was delineated earlier in this article; for a full account the reader is referred to [4]). Note that such hierarchies, which are of major concern within the configurable-computing community, are currently engineered into our artificial systems whereas in nature they have emerged through evolution.

Despite these obstacles, the future seems pregnant with possible applications for adaptive hardware. In the end it will probably be the enabling technology that furnishes the impetus for progress in this area. The evolution of chip technology might well be the ultimate harbinger of evolving chips. ■

References

1. Darwin, C. *On the Origin of Species by Means of Natural Selection, or the Preservation of Favoured Races in the Struggle for Life.* John Murray, London, 1859.
2. Dawkins, R. *The Blind Watchmaker.* W.W. Norton and Company, New York, 1986.
3. Mange, D., Madon, D., Stauffer, A., and Tempesti, G. Von Neumann revisited: A Turing machine with self-repair and self-reproduction properties. *Robotics and Autonomous Systems 22,* 1 (1997), 35–58.
4. Mange, D. and Tomassini, M Eds. *Bio-Inspired Computing Machines: Toward Novel Computational Architectures.* Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.
5. Margulis, L. Symbiosis and evolution. *Scientific American 225,* 1 (July 1971), 48–57.
6. Michalewicz, Z. *Genetic Algorithms + Data Structures = Evolution Programs, 3d ed.* Springer-Verlag, Heidelberg, 1996.
7. Sanchez, E., Sipper, M., Haenni, J-O., Beuchat, J.L., Stauffer, A. and Pérez-Uribe, A. Static and dynamic configurable systems. *IEEE Transactions on Computers.* To appear.
8. Sanchez, E. and Tomassini, M., Eds. Towards evolvable hardware. *Lecture Notes in Computer Science, vol. 1062,* Springer-Verlag, Heidelberg, 1996.
9. Sipper, M. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Springer-Verlag, Heidelberg, 1997.
10. Sipper, M., Sanchez, E., Mange, D., Tomassini, M., Pérez-Uribe, A., and Stauffer, A. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Transactions on Evolutionary Computation 1,* 1 (Apr. 1997), 83–97.
11. Turley, J. Triscend ES reconfigures microcontrollers. *Microprocessor Report 12,* 15 (1998), 12–13.
12. Villasenor, J. and Mangione-Smith, W.H. Configurable computing. *Scientific American 276,* 6 (June 1997), 54–59.

Moshe Sipper (Moshe.Sipper@epfl.ch) is a senior researcher at the Swiss Federal Institute of Technology in Lausanne, Switzerland; lslwww.epfl.ch

Daniel Mange (Daniel.Mange@epfl.ch) is a professor at the Swiss Federal Institute of Technology in Lausanne, Switzerland; lslwww.epfl.ch

Eduardo Sanchez (Eduardo.Sanchez@epfl.ch) is a professor at the Swiss Federal Institute of Technology in Lausanne, Switzerland; lslwww.epfl.ch