ORDINARY HARDWARE DOES THE SAME OLD JOB UNTIL IT WEARS OUT, WHEREAS EVOLVABLE HARDWARE ADAPTS ITSELF TO A CHANGING TASK

A new species of hardware

MOSHE SIPPER
Swiss Federal Institute of Technology
&
EDMUND M. A. RONALD
Ecole Polytechnique, Center for Applied Mathematics

AS YOU READ THESE LINES, YOU ARE USING A POWERFUL set of devices—including eyes, hands, and a brain—all of which share a fundamental characteristic: they are the products of the random mutations and genetic mixing of Darwinian evolution. As anthropologist Melvin Konner wrote not long ago: "Neuroanatomy in any species—but especially in a brain-ridden one like ours—is the product of a sloppy, opportunistic, half-billion year [process of evolution] that has pasted together, and only partly integrated, disparate organs that evolved in different animals, in different eras, and for very different purposes" [see To Probe Further, p. 64].

This single sentence, which captures the basic workings of natural evolution, also contains all the qualifiers (sloppy, opportunistic, pasted together, only partly integrated, and so forth) needed to describe "bad" engineering. And yet, between Man and Nature, Nature is the more ingenious engineer of the two. The proof is right before your eyes. Part of the proof is, in fact, your eyes, as well as the other organs that make up your body.

If natural evolution is so successful a designer, why not simulate its workings in an engineering setting, by using a computer to evolve solutions to hard problems? Researchers pursuing this idea in the 1950s and '60s gave birth to the domain of evolutionary computation. Four decades later, the domain is flourishing, both in industry and academia, presenting what may well be a new approach to optimization and problem-solving.

Published in 1859, Charles Darwin's *On the Origin of Species by Means of Natural Selection* shook the foundations of not only science but also society at large. Now, with new uses for the evolutionary model coming into being, researchers and scientists are beginning to create hardware that can grow and improve itself over time, evolving steadily as it finds new and better ways to do the tasks it has set before it. And the results may have as much of an impact as Darwin's findings did.

THE ORIGIN OF A NEW SPECIES

Of course, Darwin's theory of random mutation under selective pressure has in effect been utilized for much more than 40 years. Humankind has practiced evolutionary engineering for thousands of years, in the guise of plant breeding and animal husbandry. The move from the natural world to the digital is but a small step further.

A computer can run the evolutionary process all by itself once software has been used to define the thing to be evolved and to create a pool of initial specimens. The software evaluates the existing generation of specimens in accordance with a user-defined fitness criterion, then breeds the next generation by combining and mutating—in line with the laws of probability—the fittest of the current candidates. This process of fitness-based reproduction is reiterated in the hope of finding a solution that meets the user's criteria of acceptability. [See "What is evolutionary computing?" by David Fogel, *IEEE Spectrum*, February, pp. 26–32.]

Essentially, evolutionary computation is a software affair, carried out as a simulation on ordinary hardware such as a PC or workstation. However, it is also possible to take evolutionary algorithms similar to those used by software and embed them directly into hardware. These devices, known as evolvable hardware, are the focus of a growing endeavor to build autonomous, adaptive, and fault-tolerant electronic systems, the present targets being, among others, computers, controllers for medical prostheses, and photographic printers, about which more will be said later.

The field was officially inaugurated only five years ago, when Eduardo Sanchez and Marco Tomassini from the Swiss Federal Institute of Technology, in Lausanne, organized the first Conference on Evolvable Hardware in that city. In the preface to the conference proceedings, entitled *Towards Evolvable Hardware*, they outlined the field's mission: "The remarkable increase in computational power and, more recently, the appearance of a new generation of programmable logic devices have made it possible to put into actual use models of genetic encoding and artificial evolution; this has led to the simulation and ultimately the hardware implementation of a new brand of machines.

"We have crossed a technological barrier, beyond which we no longer need content ourselves with traditional approaches to engineering design; rather, we can now evolve machines to attain the desired behavior."

Sanchez and Tomassini went on to state boldly that "...we are witnessing the nascence of a new era, in which the terms 'adaptation' and 'design' will no longer represent opposing concepts."

A VITAL DISTINCTION

Engineers using evolutionary computation depart from the classical design route, and allow the computer to search automatically through the "space" of all possible designs. But just how can evolution be used to make physical devices evolve—how is evolvable hardware obtained? The simple answer is: make some desired behavior of the device the goal of the evolutionary process.

To some, evolvable hardware may seem merely the offspring of the marriage of computer hardware and evolutionary software. To the authors and their colleagues, however, it differs fundamentally from evolutionary computation. The definition of evolvable hardware hinges on whether or not electronic circuits play a fundamental role in the evolutionary process; the hardware is in the loop, so to speak, as opposed to the entire evolutionary process being run as a software simulation. These circuits may be off-the-shelf digital devices, such as field-programmable gate arrays (FPGAs) [see "Self-improvement for ICs," p. 62], which until recently have been the main workhorse of evolvable hardware. But other types of commercially available circuits, including analog, may be used, if they can be molded, or configured.

To clarify the difference from evolutionary computation, consider some recent work using evolutionary techniques to design electronic circuits. Over the past few years, a brand of evolutionary algorithms called genetic programming has been applied to the evolution of analog electrical circuits by Stanford University professor John Koza, along with his colleagues Forrest Bennett III, David Andre, and Martin Keane. The result has been a plethora of circuits for a diversity of functions—filters, amplifiers, and computational circuits (squaring, cubing, and logarithmic), to name a few.

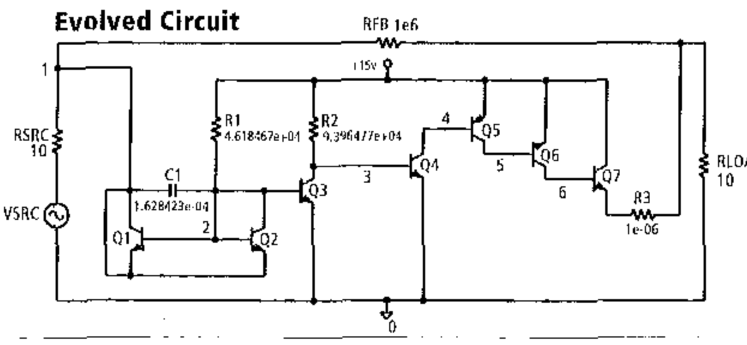
Even more recently, a novel evolutionary algorithm was developed by two other researchers also seeking to evolve analog circuits, Jason Lohn and Silvano Colombano from the NASA Ames Research Center, in Moffet Field, Calif. Their approach is simpler in some respects than the one used by Koza and his colleagues [Fig. 1].

In all cases, the entire evolutionary process was carried out as a software simulation, with fitness of the candidate designs calculated by means of the Spice circuit simulator, the *de facto* standard for analog circuits. At no time did any hardware undergo an evolutionary change. Classified as evolutionary circuit design, this work is a subdomain of evolutionary computation. It is an important area of application that includes a multitude of hard problems and promises exciting results;

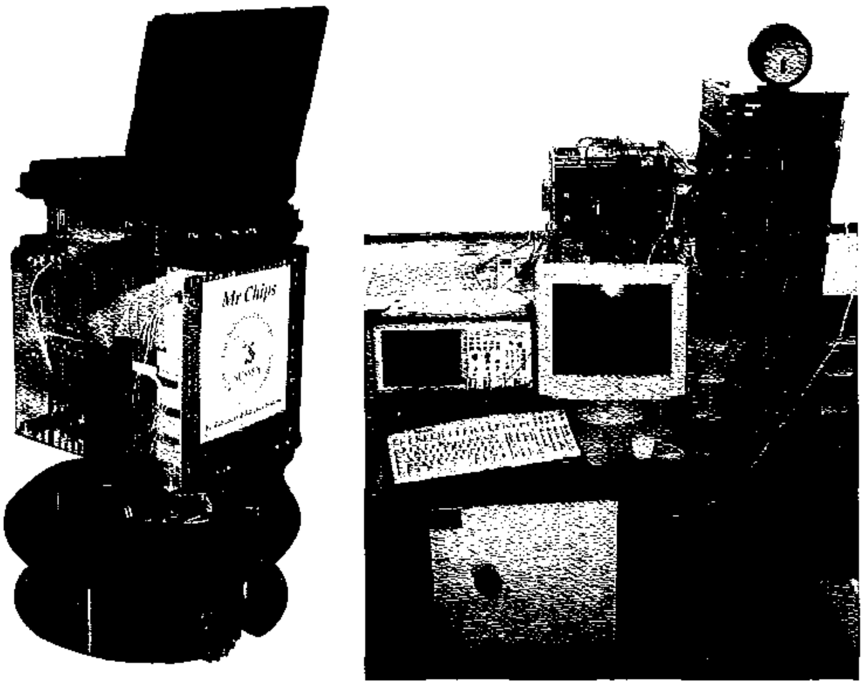
Genetic Instructions

```
transistor(N, ACTIVE_NODE, NEW_NODE, INPUT_NODE);
transistor(N, BASE, ACTIVE_NODE, PREVIOUS_NODE);
resistor_cast_to_ps(4.618467e+04);
capacitor_cast_to_input(1.628423e-04);
transistor(N, NEW_NODE, ACTIVE_NODE, GROUND_NODE);
resistor_cast_to_ps(9.396477e+04);
transistor(N, NEW_NODE, ACTIVE_NODE, GROUND_NODE);
transistor(P, NEW_NODE, ACTIVE_NODE, PS_NODE);
transistor(N, NEW_NODE, ACTIVE_NODE, PS_NODE);
transistor(N, PS_NODE, ACTIVE_NODE, NEW_NODE);
resistor_move_to_output(1e-06);
```

Evolved Circuit



[1] This genetic code at top created the circuit below it. Conjuring up new circuits is merely a matter of mixing the code lines and generating new values for the circuit elements. As the code is relatively simple, the rules for reproduction and mutation can also be simple.



[2] Mr. Chips, the wheeled robot [far left] developed by Adrian Thompson at the University of Sussex, provides the platform on which to check out circuits for navigating a small area without bumping into walls. The control circuit is evolved on an off-line system [near left] and downloaded into Mr. Chips, thereby enabling researchers to check the viability of a design in a real environment quickly.

but even with construction of the evolved circuit that emerges from the simulation, it is not evolvable hardware.

For an application to qualify as evolvable hardware, the presence of real electronic circuits, rather than software simulation, is a must. This is not just a chauvinistic attitude toward software on the part of hardware designers: using real hardware fundamentally changes the evolutionary process—and its results.

For one thing, there is no need to transfer the result of a simulation to hardware, whereas this step is a problem for several software-based efforts. Designing a car, a robot, or an electronic circuit by simulation frequently presents unpleasant surprises when the device is built. Sometimes, these surprises are trifling; at other times, they can be quite nasty, preventing the designers from reaching their intended goal. Using the actual hardware—instead of a simulation—during the evolutionary process makes the end result more predictable.

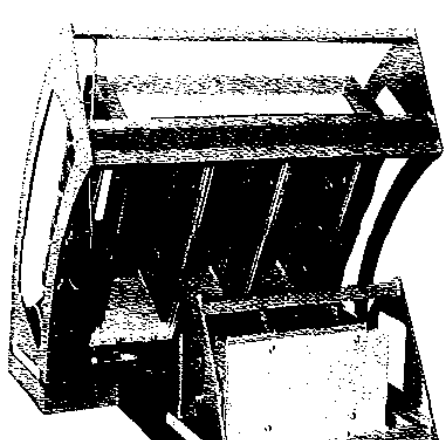
EVOLUTION, OFF- AND ON-LINE

To qualify as evolvable hardware, then, the design process must go beyond simulation and use an electronic circuit. There are two ways of doing this: off-line and on-line. With off-line evolvable hardware, the circuit is merely a reconfigurable servant used for measuring fitness, while the evolutionary algorithm runs on a master computer.

In a simple off-line application within the field of evolutionary robotics, Adrian Thompson, of the University of Sussex, Brighton, England, evolved an onboard controller for a robot [Fig. 2]. The goal of the robot was to move about on wheels within a rectangular arena 2.9 by 4.2 meters in size without bumping into walls. The setup consisted of a single robot, used to test the specification for each controller, referred to using a biological analogy as a genome. (A genome is the deoxyribonucleic acid, or DNA, of an organism, which includes the genes that determine its structure and functioning.) The population of candidate robot controllers or genomes was kept off-line, stored as configuration descriptions on a standard workstation connected to the robot. Each fitness evaluation involved downloading the respective genome onto the (single) robot, allowing it to run about so that performance statistics could be collected, and then computing the fitness score in accordance with these statistics.

Meanwhile, and for some years now, the evolution of artificial neural networks has absorbed Hugo de Garis, until recently part of the Advanced Telecommunications Research Institute's Human Information Processing Research Laboratories in Kyoto, Japan (www.hip.atr.co.jp), and his colleagues. Their project, dubbed brain building, involves a special-purpose slave machine, incorporating hundreds of very large-scale ICs, so that the master computer may quickly test the fitness of individual neural networks [Fig. 3]. The chips are field-configurable devices.

Unlike off-line evolution, in which an external, supervisory computer carries out the evolutionary process, on-line evolvable hardware embeds the process in the target device. Thus the target device maintains the evolving population, evaluates the fitness of individuals, makes selections, and applies the genetic operators of crossover and mutation. In 1996, an on-line evolvable device called

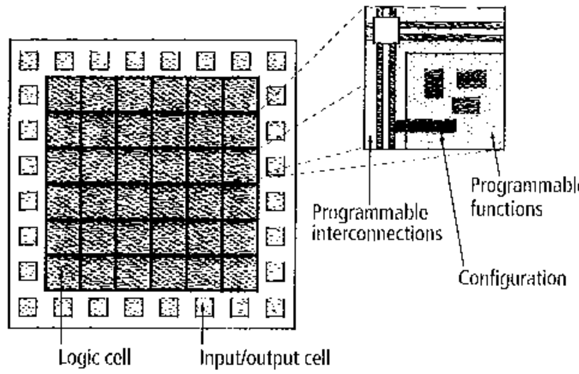


[3] The ideas of Hugo de Garis and his colleagues about implementing a cellular automata machine (CAM) have led to the construction of the CAM-Brain Machine (CBM) by start-up Genobyte Inc., of Boulder, Colo.

Up to 64 000 modules can be placed in the CBM, resulting in billions of cells that can each be updated thousands of times a second, fast enough for real-time control of robots.

The aim of the project is to build a billion-neuron artificial brain by 2001.

Self-improvement for ICs



In a field-programmable gate array, the functional logic cells provide one independent level of programmability, the interconnections between the cells provide another, and the input/output cells provide a third.

"We have crossed a technological barrier beyond which we no longer need content ourselves with traditional approaches to engineering design...," wrote Eduardo Sanchez and Marco Tomassini in the proceedings of the first ever Conference on Evolvable Hardware, held in Lausanne, Switzerland.

What technological barrier did they mean? In part it was the need for a fundamental grasp of the theory and application of evolutionary computation—a need that had been met in the last few years. The other part was the lack of malleable hardware, which had been overcome by progress in computer hardware, and especially in field-programmable gate arrays (FPGAs).

An FPGA is a large, fast integrated circuit that can be modified, or reconfigured, almost at any point by its end-user [see figure]. Physically, it consists of arrays of logic cells linked by an infrastructure of interconnects and each using an array of transistors to realize some circuit function.

The device can be configured at any or all of three levels: the function of each logic cell, the interconnections between cells, and the cell inputs and outputs. All three levels are programmed with a string of bits that can be loaded from an external source repeatedly; hence the IC is reconfigurable.

A basic distinction created by the novel technology is one between programmable and configurable circuits. A programmable circuit iterates ceaselessly through a three-phase loop: an instruction is fetched from memory, decoded, and then passed to the execute phase. The process may call for several clock cycles, and is repeated for the next instruction, and the next, and so on.

A configurable circuit, on the other hand, can be regarded as having but a single, noniterative fetch phase. The so-called configuration string, fetched from memory, requires no interpretation and is used just as it is to set up the hardware for a given task. No further phases or iterations are needed. The ability to control the hardware in such a direct manner gives the user access to a far wider range of functions, but at the price of a more arduous design task.

Within the domain of configurable computing, two types of configuration strings can be distinguished: static and dynamic. A static string, which adapts the circuit to perform a certain function, is loaded once at the outset, thereafter changing not at all during execution of its task. Static applications are mainly aimed at attaining the classical goal in computing: improved performance, be it in terms of speed, resource utilization, or area usage.

Conversely, a dynamic configuration string is able to change during task execution. Dynamic systems adapt to and keep on functioning through variable circumstances, forming an excellent substrate for implementing adaptive, evolvable systems.

The behavior of an FPGA circuit changes every time a novel configuration string is downloaded into the configuration register. In a typical engineering application, the circuit is designed to behave in a fixed manner, in accordance with preordained specifications. Borrowing biological terminology, we could say that we are given the phenotype desired—the mature organism—and asked to find its genotype—the underlying genetic specification.

Engineers usually solve the phenotype-to-genotype problem by resorting to various analytical and empirical tools that have accumulated over the years; logic design is full of them (ECAD, for example). With evolvable hardware, the phenotypic behavior is given and evolution is used to find the underlying genotype.

—M.S. & E.M.A.R.

the Firefly machine was built by Moshe Sipper and his colleagues from the Swiss Federal Institute of Technology, Lausanne, Switzerland [Fig. 4]. They designed the circuit board to improve the way it performed a synchronized-oscillation task—similar to the way a swarm of fireflies will attempt to pulse their lights on and off in unison. It was an early proof of concept of the idea of on-line evolvable hardware.

More practically, several on-line chips for specific applications have been designed and fabricated by Tetsuya Higuchi and his colleagues at the Evolvable Systems Laboratory of the Electrotechnical Laboratory in Tsukuba, Japan. They have built both digital and analog circuits aimed at commercial devices: an analog chip for cellular phones, a clock-timing architecture for gigahertz systems, a neural-network chip capable of autonomous reconfiguration, and a data-compression chip for electrophotographic printers.

Higuchi and his colleagues have also built a general-purpose chip for on-line evolvable hardware and used it to implement a controller for an artificial hand controlled by electric pulses from the nerves in the arm muscles [Fig. 5]. In normal conditions, a disabled person trains for over a month before being able to manipulate a prosthetic hand with ease. Reversing this scenario, Higuchi and his colleagues had the artificial hand adapt itself to the disabled person, instead of having the person adapt to the hand. The idea was that the on-line controller should accept signals from the nerves in the arm and map them to desired hand actions. Because those signals vary greatly between individuals, it is impossible to design such a circuit in advance. But with the evolvable-hardware controller, the hand usually requires less than 10 minutes to adapt to its owner through on-line training, in which the person repeats various hand movements—a notable improvement over the one month required when the owner is the one doing the adapting.

Off-line and on-line evolvable hardware each offer distinct advantages. The off-line approach allows the engineer to use all the computing power available in his laboratory and to employ sophisticated evolutionary algorithms to tackle the design problem at hand, while keeping actual hardware in the loop. The on-line approach holds out the possibility of setting loose an evolving device in its target environment, where it will adapt to perform its intended function. For instance, the hand-controller chip has to be placed in a lightweight prosthetic hand. Similarly, in an on-line robotic application (as opposed to the off-line one described above), the goal is to send the cumbersome workstation packing and let the robot roam about with an evolvable chip in what passes for its belly.

Some form of on-line adaptation is, in fact, the only possibility where dynamic environments are concerned: a robot sent to explore the surface of Jupiter cannot be preprogrammed entirely in advance, and will therefore have to adapt on-line.

FINDING THE UNKNOWN

Up to this point, the "hard" engineering aspects of evolvable hardware have been the focus, namely, applications within the existing "species" of analog and digital devices. Now the question is whether it is possible to generate new species, entirely novel designs, whose underlying structure and functionality are more than evolutionary—they are revolutionary.

Unconventional electronic design through artificial (as opposed to natural) evolution has been explored at the University of Sussex in Brighton, where Adrian Thompson, Paul Layzell, and Ricardo Zebulum evolved a number of *a priori* digital circuits, including the aforementioned robot controller and a tone discriminator.

In these cases, evolution was unfettered by standard digital concerns. No spatial-structure constraints (such as limiting the number of recurrent connections) were placed on the evolving configurable device, an FPGA. Nor were there any impositions on modularity (such as insisting that two functions need be co-resident on the same module) nor any dynamical constraints (such as the insistence on having a synchronizing clock or handshaking between modules). The evolved circuits resembled nothing that an engineer would design. Thompson, Layzell, and Zebulum concluded: "What initially seemed daring hypotheses are now either matter-of-fact, or within reach. From the vastness of design space, practically useful novel regions beckon."

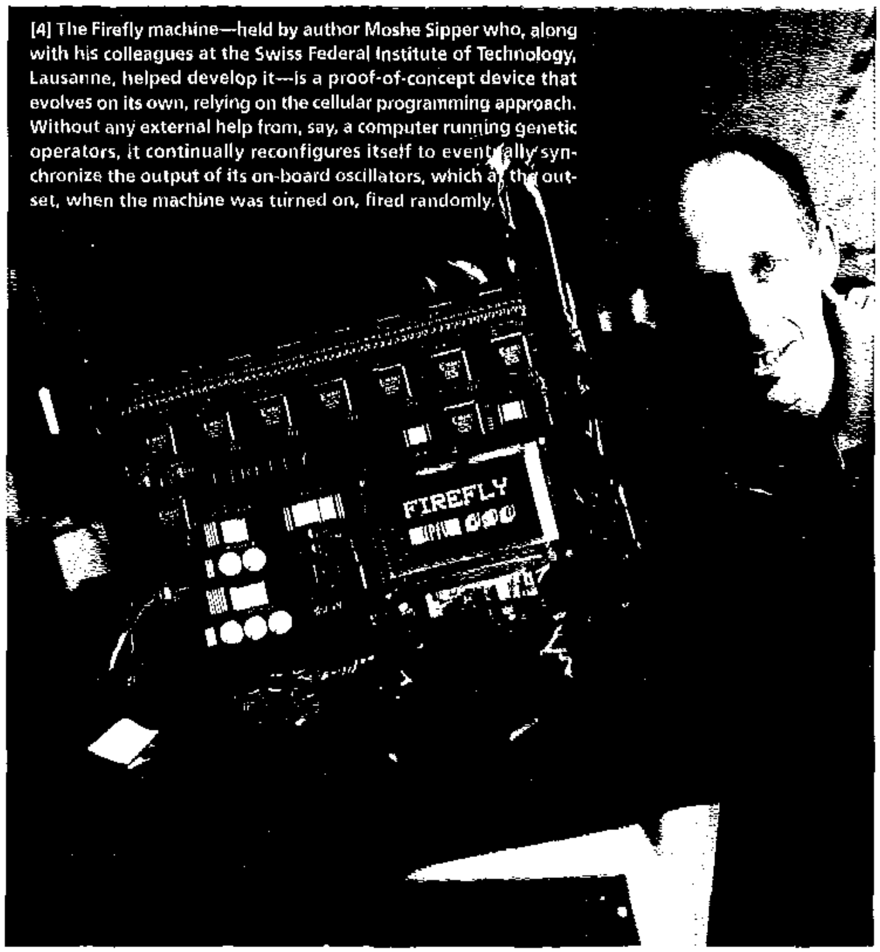
Elsewhere in Britain, an attempt was made to find not just novel circuits but novel circuit types. Julian Miller at the University of Birmingham and Peter Thomson of Napier University in Edinburgh are evolving arithmetic circuits, such as 2- and 3-bit binary multipliers using evolutionary circuit design. "Arithmetic circuits are interesting...," they wrote in the paper they presented at the Second International Conference on Evolvable Systems in 1998, "since there are well known conventional designs with which the evolved solutions can be compared."

But why should anyone wish to evolve such thoroughly well-known types of circuits? Well, one new circuit design is all it takes: finding a truly novel 2-bit multiplier (faster, smaller, more energy efficient) or some novel design principle could mean riches beyond imagination—literally. The evolutionary design system devised by John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane [see To Probe Further, p. 64] has rediscovered a number of previously patented—and thus innovative according to the rules of patent law—circuits, and they are continuing their search for as yet unpatented inventions.

These works are somewhat controversial, and engineers often eye them with suspicion. One major issue has to do with robustness: the circuits evolved by Thompson and his colleagues, for example, often exploit haphazard analog characteristics (such as slight, unintentional differences in resistance between two devices that could be used as an *ad hoc* means to adjust current) during evolution. This renders it impossible to transfer the evolved designs to other chips of the same type (the Sussex group has recently enhanced its paradigm to address the robustness issue.) Nonetheless, this novel use of simulated evolution may lead one day to the evolution of novelty: new electronic species.

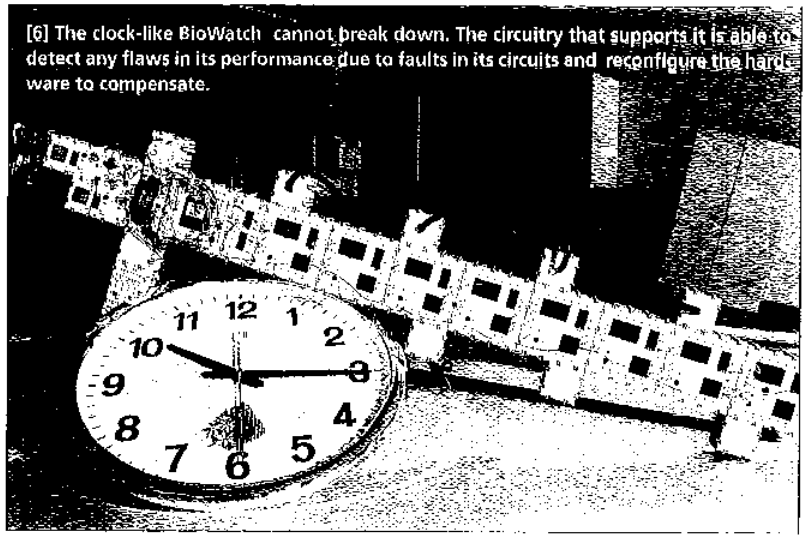
Over the past few years, a growing number of computing scientists and engineers have been turning to nature, seeking inspiration in its workings in order to augment the abilities of their artificial systems. So far, this article has only considered one biological source of ideas: evolution.

There are in fact three prime sources of biological inspiration—evolution (or phylogeny, to use the Greek-derived term favored by biologists), development (ontogeny), and learning (epigenesis). Evolvable hardware thus inhabits one realm of this broader biological terrain, dubbed the POE model. Like evolution, learning hard-



[4] The Firefly machine—held by author Moshe Sipper who, along with his colleagues at the Swiss Federal Institute of Technology, Lausanne, helped develop it—is a proof-of-concept device that evolves on its own, relying on the cellular programming approach. Without any external help from, say, a computer running genetic operators, it continually reconfigures itself to eventually synchronize the output of its on-board oscillators, which at the outset, when the machine was turned on, fired randomly.

[5] Researchers at the Electrotechnical Laboratory in Tsukuba, Japan, used evolvable hardware to develop a prototype artificial hand that adapts to a patient's unique arm-muscle activity. Typically, the adaptation is the other way round: people control the hand by learning to modify their muscle activity.



[6] The clock-like BioWatch cannot break down. The circuitry that supports it is able to detect any flaws in its performance due to faults in its circuits and reconfigure the hardware to compensate.

ware is another busy area of research and application. Today, it mainly involves hardware implementation of artificial neural networks, about which much has been written.

GROWTH OPPORTUNITIES

The third source of inspiration, ontogeny, is the one least studied to date by engineers. Ontogeny is the process by which multicellular organisms develop, through the successive divisions of a fertilized mother cell, known as the zygote.

Drawing inspiration from this fundamental natural process, a Swiss group has been developing the embryonics project. Among their number are Daniel Mange from the Swiss Federal Institute of Technology, Lausanne, and Pierre Marchal from the Centre Suisse d'Electronique et de Microtechnique in Neuchâtel.

Embryonics, for embryonic electronics, aims at developing very large-scale ICs, capable of self-repair and self-replication. These two properties of natural organisms are attained in hardware by implementing in ICs certain features of natural ontogeny.

The ability to self-repair enables an electronic "organism" to recover from minor faults, while the ability to self-replicate allows such an organism to recover from a major fault, by creating a novel, faultless clone. The Swiss group has built a number of prototypes to date, one of which is the BioWatch - a self-repairing watch [Fig. 6].

Be it evolution, learning, development, self-repair, self-replication or any other biological process, the tendency is clear: biological inspiration in engineering is on the rise; machines are becoming more adaptive. Many open issues and research avenues await exploration, yet it seems safe to predict a future rife with adaptive, bio-inspired hardware. The applications of such systems are bounded only by human imagination.

Which is, of course, unlimited. ◆

TO PROBE FURTHER

The *IEEE Transactions on Evolutionary Computation* and the newly created journal *Genetic Programming and Evolvable Machines* (Kluwer Academic Publishers, Norwell, Mass.) are fundamental sources of work on evolvable hardware. A perspective on the differences between evolvable hardware and evolutionary computing appeared in the April 1997 issue of the *Transactions*: "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," by Moshe Sipper, Eduardo Sanchez, Daniel Mange, Marco Tomassini, Andrés Pérez-Uribe, and André Stauffer (Vol. 1, no. 1, pp. 83–97).

The special *Transactions* issue of September 1999 (Vol. 3, no. 3) was entitled "From Biology to Hardware and Back." Edited by Moshe Sipper and Daniel Mange, it contained such articles as "Real-world applications of analog and digital evolvable hardware" by Tetsuya Higuchi, *et al.* (pp. 220–35), and "Explorations in design space: Unconventional electronics design through artificial evolution," by Adrian Thompson, Paul Layzell, and Ricardo S. Zebulum (pp. 167–96) and in "A circuit representation technique for automated circuit design" by Jason D. Lohn and Silvano P. Colombano (pp. 205–19).

The evolution of electrical circuits by genetic programming is described in *Genetic Programming III: Darwinian Invention and Problem Solving* by John R. Koza, Forrest H. Bennett III, David Andre, and Martin A. Keane (Morgan Kaufmann, San Francisco, 1999).

Evolution of Parallel Cellular Machines: The Cellular Programming Approach by Moshe Sipper (1997) is considered a basic text in this field. So, too, is *Towards Evolvable Hardware*, edited by Eduardo Sanchez and Marco Tomassini (1996). Both books are published by Springer-Verlag, Heidelberg, Germany, as are the proceedings of the biennial International Conference on Evolvable System.

For more information on the use of FPGAs in evolvable hardware, see "Static and dynamic configurable systems" by Eduardo Sanchez, Moshe Sipper, Jacques-Olivier Haenni, Jean-Luc Beuchat, André Stauffer, and Andrés Pérez-Uribe, published in the *IEEE Transactions on Computers*, Vol. 48, no. 6, pp. 556–64.

As for the introductory quotation, it was taken from an article by anthropologist Melvin Konner, "A piece of your mind," in *Science*, July 1998, Vol. 281, pp. 653–54.

ABOUT THE AUTHORS

Moshe Sipper is a senior researcher at the Swiss Federal Institute of Technology in Lausanne, Switzerland. His chief interests involve the application of biological principles to artificial systems, including evolutionary computation, cellular computing, bio-inspired systems, evolvable hardware, complex adaptive systems, artificial life, and neural networks.

Edmund Ronald is an affiliate researcher at the Center for Applied Mathematics of the Ecole Polytechnique, in Paris, and a visiting researcher at the Swiss Federal Institute of Technology in Lausanne. His interests include experimenting with collective robotics and the philosophical underpinnings of artificial life.

Spectrum editor: Richard Comerford