Computer Design Meets Darwin

A silicon version of natural selection generates unexpected new circuit designs by mutating, recombining, and testing actual hardware over many generations

Robot designers and artificial-intelligence researchers have long been inspired by living things. Now, computer science is starting to mimic some of the most fundamental features of life. As the stories in this Special News Report show, computer chips are now reproducing, mutating, and evolving. In some laboratories, they are even trading in their simple digital behavior for the complex responses of living neurons.

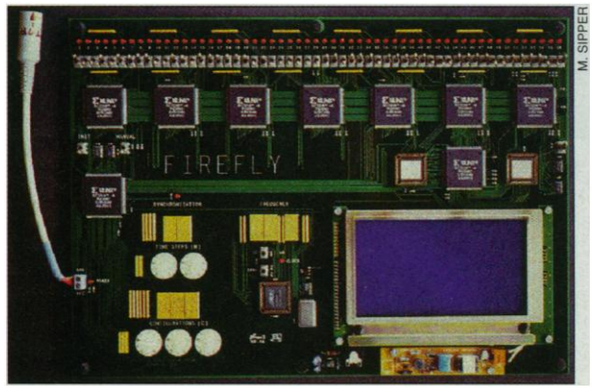
The innovation arrived, so the story has it, as an epiphany. Hugo de Garis, a computer scientist who works for Advanced Telecom Research in Japan and who describes his avocation as building brains, was visiting George Mason University in Fairfax, Virginia, in the summer of 1992, when he had a discussion with an electrical engineer. This "E.E. guy," as de Garis calls him, was telling him about computer chips known as field programmable gate arrays, or FPGAs, which are, in effect, pieces of hardware that can be reconfigured by software. "You can send in a software instruction, and it tells this programmable hardware how to wire itself up," says de Garis.

De Garis then had a moment of visionary clarity: "The idea occurred to me," he says, "that if you could send in a software instruction to a piece of hardware to wire it up, maybe you could look on that software instruction as the equivalent of a genetic chromosome. You could breed it, by mating chromosomes, mutate it randomly, maybe actually evolve the hardware. So then I started asking this guy, 'Could you do this infinitely; could you just keep sending in instructions, rewriting the hardware again and again and again?' And he said that for some FPGAs that was possible. . . ."

That was the beginning. Five years later, some 50 researchers, working in a handful of labs, are pursuing the vision of the computer chip as a biological entity that can be set to evolve at electronic speeds through Darwinian natural selection. The process should yield a configuration of logic gates that perform a desired task, even if the programmer had little or no idea how to go about configuring the chip to begin with. Researchers have already used the technique to create simple proof-of-principle devices and are now trying to evolve chips for practical applications like data compression and communications. The ultimate endpoint for evolvable hardware—at least for the extreme technological optimists—is the creation of artificial intellects.

Whether this visionary idea is more than an evolutionary dead end in the natural selection of computer science still remains to be seen. But the vision is a powerful one. "Eventually, we will need to know how to design hardware when we have no idea how to do it," says David Fogel, chief scientist of Natural Selection Inc. in La Jolla, California, and editor-in-chief of the IEEE *Transactions on Evolutionary Computation*. "This presents an alternative way. It may not be the only way; it may end up not even being the most practical way, but it's a reasonable thing to pursue."

The vision of evolvable hardware builds on a much older one: genetic or evolutionary algorithms, a software concept that dates back to the 1950s. Genetic algorithms simulate evolution in a computer in order to solve a problem in, say, design. The algorithms start with a string of bits—the binary equivalent of a chromosome. The string specifies one possible solution, and the algorithm includes steps for simulating the performance of the string, measuring its "fitness"—how good a solution it is—and producing new variants.



M. SIPPER

Evolvable hardware. The Firefly machine, which consists of circuits that evolve the ability to flash on and off in tandem.

For example, says John Koza, a Stanford University computer scientist, "you can code most engineering design problems into strings of bits." Take the design of an ideal turbine blade: One of the bits might indicate the material of the blade; the next three bits might specify its thickness; the next four, its twist. Each set of bits defining a particular aspect of the blade can be thought of as a gene. "Then," says Koza, "you find the best design by searching for the best possible arrangement of bits [using] this genetic algorithm."

The algorithm creates a population of perhaps 1000 individuals—blade designs, in this case—each defined by a slightly different bit string, and simulates the performance of each bit string on some desired task. The better individuals are then bred together by combining their genes, and the offspring are tested: "It's as if you're breeding animals, and you're doing animal husbandry," says Koza. The algorithm repeats this process of breeding and testing over hundreds of generations. To make the analogy to evolution even more powerful, a few bits are randomly changed in each new individual, introducing slight variations on the blade designs.

Hardware genetics

While genetic algorithms have evolved into powerful tools for optimizing designs or schedules, they are, at best, simulations run on general-purpose computers—painfully slow at tracing the evolution of thousands of individuals through hundreds of generations. What de Garis realized 5 years ago is that for circuit design, at least, there's a faster way: build a special-purpose chip that could wire itself up to evaluate a design in an instant, throw it out, and then rewire itself to execute another. "Instead of executing 100,000 consecutive software instructions in serial," says Koza, "you embody the individual in hardware," and the hardware itself evolves an answer.

That's where the FPGA comes in. As de Garis conceived and Tetsuya Higuchi of the Electrotechnical Laboratory in Tsukuba, Japan, first showed experimentally, these reprogrammable circuits are the ideal setting for exercising genetic algorithms in hardware. Each FPGA combines a memory with an array of thousands of identical cells, each of which represents a potential logic gate that can be configured or reconfigured—changed from an OR to an AND gate, for example—by setting "configuration bits" in memory. Because the bits can be set in a nanosecond, the entire chip can be reconfigured online.

To understand how a genetic algorithm could guide the evolution of an FPGA, imagine trying to create a chip that could recognize

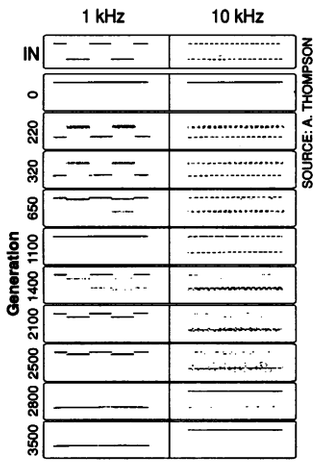
the letters of the alphabet. The starting point would be sets of configuration bits specifying perhaps 1000 possible designs for this pattern-recognizing chip, and the measure of fitness would be how well each "individual" recognized letters. The genetic algorithm would download each configuration onto the FPGA, test the resulting circuit's ability to recognize letters, and then download and test the next individual, taking a few microseconds per cycle. After the entire 1000 individuals had been graded, the best would be kept and bred together, and the bad ones discarded.

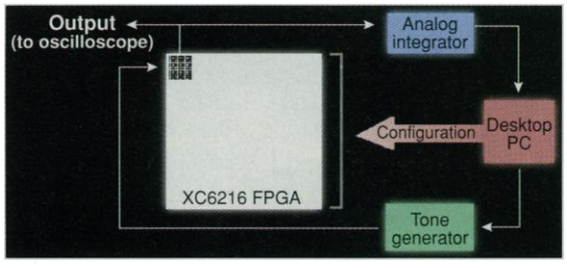
The approach has one immediate advantage over the existing techniques of circuit design, which rely either on human artistry or circuit design algorithms. Because the design actually emerges in hardware, says Fogel, it avoids a potential pitfall of designing a circuit in a software simulation: "When you go to build it, you might get something that didn't work. You'd find you hadn't taken into account the real physics of the device."

So far, notes Moshe Sipper, a computer scientist with the Swiss Federal Institute of Technology in Lausanne, evolvable hardware schemes still rely heavily on software—to breed and grade the bit strings, for instance, and control the process of evolution. No one has done what Koza calls the "whole enchilada"—putting everything on the chip, including the genetic algorithm—which would increase the speed exponentially. If that could be done, Koza says, evolvable circuits by the thousands could be assembled into a huge parallel computer, with the evolutionary steps built in locally: "You could presumably then develop brainlike self-learning."

The evolvable hardware that researchers have created so far amounts to little more than mathematical parlor games—proof-of-principle demonstrations. In Lausanne, Sipper and his colleagues created what they call the FireFly machine, which has 54 cells that evolved to flash on and off in unison. And at Stanford, Koza and Forrest Bennett have evolved an FPGA to sort seven numbers by size. They generated 1000 random combinations of "compare swap" networks, which look at any two numbers and put the smaller one first, to serve as the individuals in the genetic algorithm. They then ran the evolutionary process over 50 generations, which took a few hours. The result, he says, was a sorting algorithm wired onto the FPGA that was considerably faster and more efficient than one patented 35 years ago.

Higuchi's lab is trying to "realize more practical industrial applications," he says, by evolving the circuit at a higher level of function—in a biological analogy, by altering whole cells at each step, rather than individual genes. The group uses an FPGA whose configuration bits, rather than resetting individual logic gates, change entire functions, such as addition, multiplication, or finding sines or cosines. "With function-level evolution," Higuchi says, "you can get much more useful hardware functions." Higuchi and his colleagues have been working on chip designs for ATM networks, mobile communication, and data compression.





Silicon family history. A circuit evolves to distinguish two sound frequencies. Part of a reprogrammable chip (FPGA) was tested for its ability to deliver a 1-volt output for the low tone and a 5-volt output for the high, then reconfigured and retested. The oscilloscope traces *(top)* represent its output at various stages in the process, which continued for 3500 generations.

Evolution unlimited

Perhaps the most intriguing variant of the evolvable hardware idea is one pursued by Adrian Thompson and his colleagues at the University of Sussex in the United Kingdom. Thompson thinks computer scientists are restricting the powers of evolution unnecessarily by putting it to work only on digital logic gates. "Silicon has a much richer dynamical behavior than just flicking between ones and zeros," he says. Chip components can adopt a whole range of values intermediate between the standard 1 and 0. So, rather than making their FPGAs follow the rules of digital design, says Thompson, "we don't tell evolution anything about how we expect it to work. We let it find the best way."

Thompson's demonstration task was to feed an FPGA a single input—a 1-kilohertz or 10-kilohertz audio tone—and evolve it to generate a signal identifying the input: 1 volt for 1 kilohertz, 5 volts for 10 kilohertz. "This is actually quite a hard task," says Thompson. "The problem is the logic gates are incredibly fast. They respond on a time scale of 2 or 3 nanoseconds. The input is orders of magnitude slower. So we were asking incredibly fast components to do something much slower and produce very nice, steady output," he says. "All evolution had [for the task] was this little bit of silicon on a chip. ... It had to use the natural physical properties to get the job done."

The system that resulted from the evolutionary process worked efficiently, says Thompson, but he didn't have the "faintest idea how it worked." Back-engineering failed to decipher it, but what he did learn was that the circuit seemed to be relying on only 32 of the 100 available logic gates to achieve its task, and some of those working gates were not connected to the rest by their normal wiring. "It was either electromagnetic coupling," Thompson explains, "which is basically radio waves between components sitting right next to each other, or they were somehow interacting through the power-supply wiring."

Thompson notes, though, that such unconstrained evolution can produce something so exquisitely adapted to specific conditions that it could never be replicated on other chips. "A lot of the physical properties vary from chip to chip and with temperature as well," he says. "If I change the temperature, these evolved chips start going wrong, which is bad news for logic applications." Thompson now has a 3-year grant to study how to evolve fault-tolerant and temperature-tolerant chip designs. His new approach is to run the evolution on five different FPGA chips from five different manufacturers, and to change the temperature as the evolution proceeds. "It will be pressure for evolution to produce really robust strategies," he says.

As for the ultimate value of those strategies, the forecasts range from de Garis's vision—"making tens of thousands of circuits and connecting them up in humanly designed, artificial brain architectures"—to the sober views of Tom Knight, a computer scientist at the Massachusetts Institute of Technology. He suggests that evolutionary hardware is likely only to be "good in applications where there are no other ways to solve the problem" and expects such niches to be rare, considering the power of modern software approaches to circuit design.

But evolvable hardware researchers say that evolution has an insurmountable advantage over conventional circuit design. "If your application changes in time," says Higuchi, "conventional hardware cannot be changed to deal with it. You have to design another hardware device." Evolvable hardware faces no such limits, says Moshe Sipper. "What we have is this very powerful methodology for creating chips, electronic hardware machines, that will be able to change on the fly—dynamically. Then the question will be, what do we use them for?"

–Gary Taubes