Promises and Challenges of Evolvable Hardware

Xin Yao, *Senior Member, IEEE*, and Tetsuya Higuchi

***Abstract*— Evolvable hardware (EHW) has attracted increasing attention since the early 1990's with the advent of easily reconfigurable hardware, such as field programmable gate arrays (FPGA's). It promises to provide an entirely new approach to complex electronic circuit design and new adaptive hardware. EHW has been demonstrated to be able to perform a wide range of tasks from pattern recognition to adaptive control. However, there are still many fundamental issues in EHW that remain open. This paper reviews the current status of EHW, discusses the promises and possible advantages of EHW, and indicates the challenges we must meet in order to develop practical and large-scale EHW.**

***Index Terms*— Adaptive hardware, evolutionary algorithms, evolutionary computation, field programmable gate arrays (FPGA's), reconfigurable hardware.**

I. Introduction

EVOLVABLE hardware (EHW) refers to hardware that can change its architecture and behavior dynamically and autonomously by interacting with its environment. At present, almost all EHW use an evolutionary algorithm (EA) as their main adaptive mechanism. One of the key motivations behind EHW is to learn from nature since she has done so well in evolving wonders such as ourselves (i.e., human beings) without external forces. However, *learning* from nature is quite different from *copying* it. There are many new challenges in front of us if we want to harness the power of evolution in EHW. This paper discusses the promises and challenges of EHW in more detail in later sections.

There are different views on what EHW is, depending on the purpose of EHW. One view regards EHW as "applications of evolutionary techniques to circuit synthesis" [1, abstract]. Another view regards EHW as hardware that is capable of online adaptation through reconfiguring its architecture dynamically and autonomously [2]. Although these views are closely related and quite similar to each other, they emphasize different aspects of EHW. The former one uses simulated evolution as an alternative to conventional specification-based electronic circuit design, while the later uses it as an adaptive mechanism. However, the line between the two is gray.

EHW is fundamentally different from the hardware implementation of EA's, in which the hardware architecture does not change and is used to implement EA functions, such as selection, recombination, and mutation [3]–[5]. The main motivation for hardware implementation of EA's is to speed up the execution of *EA functions*. Such speedup, however, does not necessarily imply a faster EA application because it does not speed up fitness evaluation, which is often the most time-consuming part of an EA application. Discussion of EA's hardware implementation is beyond the scope of this paper.

EHW involves two major aspects—simulated evolution and electronic hardware. According to different EA's, e.g., genetic algorithms (GA's), genetic programming (GP), evolutionary programming (EP), and evolution strategies (ES's), and different electronic circuits, e.g., digital, analogue, and hybrid circuits, used, we could classify EHW into different categories along these two dimensions. There are, however, at least two other important dimensions we should consider in investigating EHW, i.e., how the simulated evolution is realized and what the simulated evolution is used for, because they have a direct impact on the future research and development of EHW.

EHW is usually implemented on programmable logic devices (PLD's), such as field programmable gate arrays (FPGA's). The architecture of a PLD and thus its function are determined by a set of architecture bits that can be changed (i.e., reconfigured). In EHW, the simulated evolution is used to evolve a good set of architecture bits to solve a particular problem. According to de Garis [6], EHW can be classified into two categories, i.e., extrinsic and intrinsic EHW. Extrinsic EHW simulates evolution by software and only downloads the best configuration to hardware in each generation; i.e., the hardware is only reconfigured once. Intrinsic EHW simulates evolution directly in its hardware; i.e., every chromosome will be used to reconfigure the hardware. The EHW will be reconfigured the same number of times as the population size in each generation. Hirst [1] wrote a good survey paper along this line. Sanchez *et al.* [7] described a phylogeny, ontogeny, and epigenesis (POE) model that nicely captured some of the characteristics of various EHW's.

In this paper, we take a much broader view on EHW and address some important issues not covered in Hirst's and Sanchez *et al.*'s surveys. We argue that there is a difference between EHW used as an alternative to circuit design and EHW as online adaptive hardware. Although the techniques used to develop them may be very similar, the criteria used to evaluate them are different. For EHW that is used as an alternative to conventional circuit design, there are two distinct phases. One is the evolutionary design phase, and the other is the execution phase, which usually does not require online adaptation (although it is possible). To achieve online adaptation, EHW must adapt its architecture *while operating in the real environment*. Many new issues arise when online adaptation is required.

Manuscript received April 21, 1997; revised January 15, 1998.

X. Yao is with the Computational Intelligence Group, School of Computer Science, University College, The University of New South Wales, Australian Defence Force Academy, ACT 2600 Canberra, Australia (e-mail: xin@cs.adfa.oz.au).

T. Higuchi is with the Computation Models Section, Electrotechnical Laboratory, Tsukuba, Ibaraki, 305 Japan (e-mail: higuchi@etl.go.jp).

Publisher Item Identifier S 1094-6977(99)00105-4.

The rest of this paper is organized as follows. Section II reviews the work of evolving hardware as an alternative to designing it from specifications, as is done in conventional electronic circuit design. Section III discusses the attempt of developing EHW for online adaptation. Sections IV and V present some views toward EHW and other nonevolutionary approaches to EHW. Finally, Section VI concludes this paper with a summary of the paper and some remarks.

II. Evolutionary Design of Electronic Circuits

Although EHW is a relatively new term, evolutionary design of electronic circuits have been attempted for more than a decade [8]–[10]. These early attempts did not design the architecture or function of a circuit. They were only used to optimize certain aspects of electronic circuit boards, e.g., cell placement [9], [10] and compaction of symbolic layout [8]. In essence, such work is better described as combinatorial optimization by EA's.

Recent EHW work concentrates on evolutionary design of electronic circuits, although the ultimate goal is to develop online adaptive hardware. So far, few studies have been reported on EHW, which adapts its architecture and function while operating in a real physical environment.

According to the level of chromosome representation, the design approach to EHW can be classified into the direct and indirect ones. The direct approach to EHW encodes circuit's architecture bits as chromosomes, which specify the connectivity and functions of different hardware components (often at the gate level) of the circuit. In contrast, the indirect approach does not evolve architecture bits directly. It uses a higher level representation, such as trees or grammars, as chromosomes. These trees or grammars are then used to generate circuits.

A. Indirect Approach to Evolutionary Circuit Design

1) Evolving Digital Circuits: A typical example of the indirect approach is the evolution of a binary adder using Hardware Description Language (HDL) programs [11]. In this case, programs written in Structured Function Description Language (SFL) were encoded as chromosomes and subject to evolution [12]. The chromosome representation is a derivation tree generated from a context-free grammar, which is called a rewriting system [11]. Each tree can generate one SFL program deterministically if the tree is "well-structured" [11, p. 372]. Programs generated by different derivation trees can cover all possible programs in the SFL language, which is defined by the grammar.

The root of a derivation tree is the start symbol of the grammar. The internal nodes are nonterminal symbols of the grammar, and the leaves are terminal symbols. The crossover and mutation operators applied to derivation trees are similar to those used in GP [13], [14], but with some constraints. A branch (i.e., subtree) in a derivation tree can only be replaced (through either crossover or mutation) by another one with the same nonterminal node as the root (of the subtree). This is equivalent to replace a production (i.e., rewriting) rule in the grammar with another one having the same left-hand side. Such restriction ensures that all offspring produced are legal programs of the language. In addition to crossover and mutation, gene duplication and deletion were also used to modify derivation trees.

The idea of evolving the grammar itself was mentioned [11]. It was hoped that as the tasks to be performed by the hardware became more and more complex, the grammar itself would evolve to cope with the increasing complexity. A grammar was represented by a production diagram, which is a directed graph. The main genetic operator proposed was a kind of node splitting operation [11].

Software simulation of evolving a binary adder using SFL programs was carried out [11]. The task can be described as follows.

> The target is two input and one output circuit; inputting two sequences of binary numbers from lowest figure, the circuit produces the sum of the binary numbers from the lowest figure in the output terminal. The correct circuit must consider the carry from the lower bit, so it belongs in a class of sequential circuits.

The fitness of each individual (i.e., a derivation tree/program) was calculated by evaluating its correctness in adding two 1536-bit numbers, which include all possible combinations of two 4-bit numbers [11, p. 376]. A complete binary adder circuit was found in the experiment.

2) Related Work on Evolving Derivation Trees: The chromosome representation scheme used in evolving the binary adder was also studied by Whigham [15]–[17] independently in a very different context. Whigham [15], [16] used a grammar to incorporate biases into GP to learn difficult and complex concepts. The major concern was to introduce declarative biases into GP under the general framework of inductive learning. There was no direct connection with EHW.

Whigham [15], [16] used similar crossover and mutation operators to those used by Hemmi *et al.* [11]. A schema theorem under the derivation tree representation and the crossover and mutation was given [18]. The idea of evolving the grammar itself was mentioned but not tested.

Whigham's work provides a different view toward the evolution of grammars. Such evolution can be regarded as the evolution of biases; i.e., certain knowledge or heuristics about what kind of circuits should be evolved. Such biases would be accumulated and obtained through the evolution of SFL programs and used to guide the evolution of grammars at a higher level. The importance of biases in inductive learning has long been recognized in the machine learning field and will not be repeated here.

3) Evolving Analogue Circuits: In comparison with digital circuits, analogue circuits are more difficult to design. Recent work on evolving analogue circuits using GA's [19] and GP [20]–[24] shows an alternative to analogue circuit design using the evolutionary approach. One of the key issues in such evolutionary design is to find a suitable chromosome representation of analogue circuits. This problem is quite similar to that in evolutionary artificial neural networks (EANN's) [25]–[27], in which a good chromosome representation of EANN's is also very important.

In the GP approach to analogue circuit design, trees are used to construct circuits. These circuit-constructing trees are evolved by GP [20]–[24]. Each tree can contain connection-modifying functions, component creating functions, and automatically defined functions. A number of circuits, such as a low-pass "brick wall" filter, an asymmetric bandpass filter, and an amplifier using transistors, have been evolved successfully [20], [23].

The work on evolving analogue circuits described here does not belong to EHW in a strict sense because the evolution was all implemented and simulated by software. The simulation was carried out on a parallel computer system consisting of 64 Power PC 601 processors (80 MHz) arranged in a toroidal mesh [20].

B. Direct Approach to Evolutionary Circuit Design

Instead of evolving indirectly HDL programs or trees that specify circuit architecture and function, direct evolution of architecture bits of PLD's, such as FPGA's, has also been proposed [28], [29], [2]. The architecture bits of an FPGA refer to those bits that specify its logic function and interconnections. The architecture bits uniquely determine the architecture and function of an FPGA. By evolving these bits (i.e., chromosomes), hardware can be evolved.

It is worth pointing out here the distinction made earlier in Section I of this paper between EHW used as an alternative to circuit design and EHW as online adaptive hardware. Higuchi *et al.* [28], [29], [2] have explicitly emphasized the latter although some of the techniques they proposed can also be used in evolutionary circuit design. The hardware evolution described in this subsection is at the gate level since all hardware functional units are simple logic gates, such as AND and OR gates [30].

A software simulation of evolving a GAL16Z8 chip for the six-multiplexor problem has been carried out to show the potential of the gate level evolution [28]. The chromosome used in the simulation had 108 bits, of which 12 bits were used to specify the function of the logic cell [an output logic macro cell (OLMC)] and 96 bits used to specify a fuse array that determines interconnections between inputs and the logic cell. A generational GA with uniform crossover, bit-flipping mutation, and roulette wheel selection was used to evolve a population of 100 such chromosomes. The fitness of each chromosome was calculated by evaluating the gate array on *all* 64 possible inputs. A correct six-multiplexor circuit was evolved after about 2000 generations in one run. Experiments on evolving other circuits, such as the exclusive-OR circuit, a 3-bit counter, and a four-state finite state machine, have also been reported [29], [2].

Thompson *et al.* [31]–[34] emphasized the importance of unconstrained evolution of electronic circuits (including both spatial and temporal constraints), and they provided both theoretical arguments and experimental evidences to support their points. They viewed electronic circuits more as dynamic systems than as static ones. Such a view enabled them to explore a wide range of potentials of EHW. It also revealed some fundamental issues faced by EHW in general.

One of the experiments carried out by Thompson was the evolution of a slow electronic oscillator using high-speed logic gates [31], [33]. The aim of such an experiment was to find out whether "the high-speed components can somehow be assembled to give rise to slower dynamics, without explicitly providing large time-constant resources or slow-speed clocks." [31] The experiment was quite different from others in that both spatial and temporal constraints were removed. The evolving circuit operated entirely in an asynchronous mode without any clocks. The delay at each gate was assigned a real-value "selected uniformly randomly from the range 1.0 to 5.0 nanoseconds." The delay of connections was ignored, i.e., set to zero.

Thompson [31] fixed the number of logic gates (also called nodes) at 100 in his experiment and defined the fitness of an individual as follows.

> The objective was for node number 100 to produce a square wave oscillation of 1 kHz, which means alternately spending 0.5×10^{-3} s at logic **1** and at logic **0**. If k logic transitions were observed on the output of node 100 during the simulation, with the nth transition occurring at time t_n s, the average error in the time spent at each level was calculated as

$$\textit{average error} = \frac{1}{k-1}\sum_{n=2}^{k} |(t_n - t_{n-1}) - 0.5 \times 10^{-3}|. \quad (1)$$

> For the purpose of this equation, transitions were also assumed to occur at the very beginning and end of the trial, which lasted for 10 ms of simulated time. The fitness was simply the reciprocal of the average error.

Each node (i.e., logic gate) required a genotype segment of 24 bits in the chromosome representation, which encoded the node function and the sources of its inputs. Each chromosome had a total of 101 segments, i.e., 2424 bits. The GA used was a "generational one with elitism and linear rank-based selection." The population size was 30 [31].

In the fortieth generation of one run, the GA was able to evolve an oscillator with approximately 4 kHz, while the best individual in the random initial population was one with approximately 18 MHz. The experiment did not continue after 40 generations due to "excessive processor time needed to simulate this kind of network," although "fitness was still rising." [31] A total of 68 gates were used in the 4-kHz oscillator evolved by the GA.

C. Function Level Evolution

As can be seen from the experiments on the gate level evolution, the size of chromosomes grows rapidly as the size of EHW increases. According to Higuchi *et al.*'s estimation [28], "FPGA's require from 2000 to 30 000 architecture bits to configure their circuits." Evolution of chromosomes of such sizes is inefficient even in hardware.

To address the issue, Higuchi *et al.* [30], [35], [36] proposed the function level evolution for EHW. In the function level evolution, high-level hardware functions, such as addition, subtraction, sine, etc., rather than simple logic functions are used as primitive functions in the evolution. Much more powerful circuits can be evolved using the function level evolution [30], [35], [37], [38], [36]. Since the function level evolution aims at online adaptation by EHW, it will be discussed in more detail in Section III.

D. Advantages of Evolutionary Design

EHW has been used as an alternative to conventional circuit design although the ultimate goal might be to develop EHW that adapts in a real physical environment. Such an evolutionary design approach offers a number of advantages over the conventional one used by human designers although there are some important issues that remain open.

First, the evolutionary design approach can explore a much wider range of design alternatives than those that could be considered by human beings. This has been shown by many experiments in other design tasks, such as evolutionary design of neural networks [39]–[43], building architectures [44], and arts [45]. These experiments demonstrated how evolutionary techniques could be applied to evolving novel designs that were difficult to discover by human beings. However, all of these experiments were carried out by software simulation although some of the techniques used in these software simulations will also be useful for EHW.

Second, the evolutionary design approach does not assume *a priori* knowledge of any particular design domain. It can be applied by users without resorting to domain experts. It can be used in domains in which little *a priori* knowledge is available or in which such knowledge is very costly to obtain. As the complexity of circuits increases, it becomes extremely difficult to fully understand interactions among various components of the circuits and their dynamics. The conventional design approach tends to break down in such cases, while the evolutionary approach would excel. In essence, the conventional design approach specifies *how* to design and implement a circuit, while the evolutionary approach only specifies *what* the circuit should implement, i.e., what required function or behavior the circuit should have without worrying how to achieve it.

Third, the evolutionary design approach can work with varying degrees of constraints and special requirements, if necessary, by incorporating them in chromosome representation and fitness function. As mentioned above, the evolutionary approach can work with little *a priori* domain knowledge. However, if some domain knowledge is available, it can be used to improve the efficiency of the evolutionary design. Using domain knowledge to improve EA's has been shown to be achievable and effective [46], [47].

E. Some Issues in Evolutionary Design of Electronic Circuits

1) Scalability of EHW: The importance of scalability has been recognized by several researchers [28], [48]. It is a tough problem faced not only by EHW researchers, but by other researchers in the fields of evolutionary computation, artificial neural networks, and artificial intelligence in general. To our best knowledge, all EHW experiments conducted so far have been on a small scale. That is, the EHW is small with much fewer components in comparison with the circuits designed by the conventional method. Even for these small EHW, researchers have already experienced the high computational cost of evolving circuits [28], [48].

The scalability of EHW could be divided into two related parts. The first part deals with the scalability of the chromosome representation of electronic circuits. At present, the length of chromosomes can be a couple of thousand bits for 100 logic gates [31]. For a circuit with 1000 logic gates, the expected length of chromosomes would be tens of thousands of bits, which is very inefficient to process by the current evolutionary techniques. Roughly speaking, if no constraint is imposed on the connectivity of EHW, i.e., any connectivity is possible, then the length of chromosome would grow in the order of $O(n^2)$, in which n is the number of functional components (such as logic gates) in the EHW. If connectivity is constrained to certain local neighborhood around a functional component, $O(n)$ would be achievable. However, this comes with the cost of *constraining the EHW*, something that we tried to avoid at the beginning when we embarked on EHW.

The second part of EHW's scalability concerns with the computational complexity of an EA. This is a much more important issue, which still remains open, than the scalability of chromosome representation. Neither the worst nor average case time complexity has been established for any EA. At present, it is not unusual to carry out an EHW experiment that runs for days. Yet the EHW used in these experiments contained only 100 functional components or so. The question is: how long will it take to evolve an EHW with 10 000 functional components using the current techniques?

2) Danger of Relying Too Much on Hardware Speed: Using hardware to increase the speed of evolution seems to be an answer to combat the high computational cost. While hardware does offer limited temporary relief on the high computational cost, it does not solve the problem. The sheer speed of dedicated hardware is not the answer to a time complexity issue. The importance of the time complexity and the irrelevance of hardware speed can be seen clearly from the following artificial example. Assume the average time complexity of an EA applied to an EHW is $O(2^n)$, where n is the number of functional components in the EHW. If the EHW with ten functional components requires $1024 = 2^{10}$ nanoseconds ($\approx 10^{-6}$ s) to evolve (in hardware of course), a similar EHW with only 100 functional components would need $2^{100} \approx 10^{30}$ nanoseconds ($>10^{13}$ years) to evolve. That is certainly not the time we would like to spend on EHW.

The above example shows the danger of relying too much on hardware speed while overlooking the fundamental issue. Fortunately, the time complexity of $O(2^n)$ assumed in the artificial example is not based on any theoretical or empirical evidences. Unfortunately, there is still no result on the time complexity of EHW. The possibility of an $O(2^n)$ time complexity, albeit small, does exist.

3) Fitness Evaluation and Circuit Verification/Testing: An issue that arises in the evolutionary design of electronic circuits is how to verify that the evolved circuit, i.e., EHW is correct. This would not be an issue if a fitness function could be defined such that a maximum fitness corresponds to a perfectly correct circuit. For example, in the six-multiplexor experiment [28], all 64 possible input combinations were used in fitness evaluation. The maximum fitness implies a correct EHW. However, the method will not work for circuits with a large number of inputs since the number of possible input combinations increases exponentially as the number of inputs increases.

Sometimes a fitness function that guarantees the circuit correctness is very difficult to find without incurring heavy computational cost in fitness evaluation. For example, in the sequential binary adder experiment [11], the correctness of evolved circuits had to be confirmed by human beings through “reading the description of the program” [11, p. 376]. The maximum fitness value did not guarantee the correctness of a circuit. The fitness of a circuit in the sequential binary adder experiment was defined by considering “all possible combinations of two 4 bit numbers” [11, p. 376]. However, this does not imply that a sequential binary adder that operates correctly on all possible combinations of two 4-bit numbers will be correct on all possible combinations of two five or more bit numbers. This seems to be a complex problem related to the fitness evaluation and stopping criteria used in EHW. It is difficult for EHW to know when a correct circuit, not just the one with the maximum fitness value, has been evolved because the simulated evolution only manipulates the *syntax* not *semantics* of encoded circuits.

Another example is the fitness definition used in evolving a slow oscillator [31]. The fitness value of a circuit depends on the value of k in (1). A maximum fitness for a particular k does not imply the circuit will operate correctly for larger k’s. If a large k is used in the fitness evaluation, the computational cost will no doubt increase. The correctness issue addressed here is related to the generalization ability of EHW if we viewed EHW as a learning device not an alternative to circuit design.

Unconstrained hardware evolution can cause additional problems in terms of circuit correctness since it exploits every characteristic of electronic circuits and the environment in which it is evaluated, regardless of whether a characteristic is relevant. As Thompson pointed out [31], the behavior of EHW may depend on such factors as fluctuations in temperature and power supply. Exploitation of hardware must be traded against EHW’s sensitivity to variations. It was suggested that EHW could be evaluated under various situations to “make sure” that it is not sensitive to small variations [31]. However, it is not a simple task to achieve this. First, all characteristics exploited by EHW must be varied. Second, the number of variations for each characteristic must be sufficiently large. A very high computational cost has to be paid for all of these. In addition, it is difficult to find out what characteristics would be exploited by EHW in the first place before we could vary them.

4) Termination of Evolution: The difficulty in defining a good fitness function, as mentioned in the previous subsection, also leads to the difficulty in defining a stop criterion for the simulated evolution. EHW does not know when it has found a correct solution and thus should stop since a maximum fitness value does not necessarily guarantee a correct circuit. In the existing EHW experiments [28], [29], [2], [11], [31], the correctness of evolved circuits was established by human beings. Another thing that is unclear from all of these experiments is whether the correct circuit is the result of only one run or multiple runs. If every single run can guarantee to produce a correct circuit, there would be no problem. If not, how many runs on average do we have to perform to get a correct circuit? When should we stop? Although there are quite a few papers analyzing the behaviors of an evolved circuit and showing they are correct, it is unclear whether a circuit with similar behaviors could be evolved from another separate run.

5) Other Issues: Other issues that need addressing in the current EHW research include maintainability and understandability of evolved circuits. Circuits evolved by EA’s are often very difficult to understand and thus very difficult to maintain by human beings. They are basically black boxes. In order to use EHW successfully in a real-world environment, the EHW must be maintainable. If the maintenance is carried out by human experts, they must be able to understand the EHW, which is an extremely difficult task. An entirely different approach to maintainability would be to have the EHW itself to detect and repair its faults. Mange *et al.*’s work [49]–[52] on self-repairing hardware might be a direction to go in for the future.

III. EHW for Learning and Online Adaptation

The real attractiveness and power of EHW comes from its potential as an adaptive hardware that can change its behavior and improve its performance while executing in a real physical environment (as opposed to simulation). Such online adaptation is very difficult to achieve. The difficulty is not caused by EHW, but by the *online* requirement. In other words, online adaptation would still be very difficult even if a different approach from EHW is adopted.

At present, EHW has mostly been studied in terms of off-line adaptation, except for a few examples [53]–[55]. That is, EHW is not used in an execution mode while evolving. For example, it is not used to control a real robot in a real physical environment while evolving. This can be regarded as the off-line learning phase of EHW. One of the reasons why off-line learning is used is because of the trial-and-error nature of EA’s. It is possible to produce very poor individuals by random mutation or crossover in EA’s. These poor individuals could cause severe damages or disasters to EHW or the physical environment in which it is being evaluated, if there is no additional technique to prevent them from happening. For example, an EHW evolved to control a real robot could produce such a poor controller that the robot would hit an obstacle badly. This is certainly not the way to get a fitness value of the controller in a real physical environment.

A. EHW Controllers

EHW controllers refer to those EHW that are used primarily as controllers for robots or any other devices (such as ATM switches or multiplexors) [37], [38], [56]–[60]. Mizoguchi *et al.* [56] used EHW to control an artificial ant to follow the John Muir Trail. The trail was placed on a grid. The controller of the artificial ant, which was implemented by EHW through simulation, had one input and two outputs. The input contained information about whether the trail exists in the cell in front of it. The outputs controlled the actions of going straight, turning left and turning right.

The technique used to evolve the EHW controller is the same as that used by Hemmi *et al.* [11], which is described briefly in Section II-A. Each configuration of the controller was specified by an SFL program that was produced by a derivation tree (i.e., a rewriting tree). Derivation trees generated from the SFL grammar were represented as chromosomes and evolved by production genetic algorithms (PGA's) proposed by Mizoguchi *et al.* [56]. PGA's employed six operators: selection, crossover, mutation, duplication, insertion, and deletion. The operators guarantee that all offspring will be legal trees defined by the grammar. The fitness of an artificial ant was determined by the number of cells on the trail that were traversed within a time limit and the number of time steps used. Traversing more cells on the trail with less time steps within a time limit gave higher fitness. All cells of the trail, which were fixed, were used in fitness evaluation. No testing on the generalization ability of EHW was performed. As pointed out by Mizoguchi *et al.* [56], their system "represents one approach to designing hardware." Adaptivity and generalization would not be the major concern.

Another experiment on robot control was carried out by Thompson [61], in which a real hardware robot controller was evolved for wall-avoidance behavior. The controller's input came from two sonar heads pointing left and right, respectively. Its output went to the motors for controlling two wheels. For the hardware evolution, architecture bits (also called "configuration memory") of the EHW controller, which was implemented in FPGA's, were used as genotypes. They determined functions of the functional blocks in the FPGA and their interconnections. In other words, they determined the whole function and thus behavior of the EHW controller.

In Thompson's experiment [61], a genotype *directly* encoded all details of the EHW controller, including the clock information. For the simple wall-avoidance behavior, the length of genotypes was 32 bits. A GA was used to evolve a population of 30 such genotypes. Each genotype was evaluated by evaluating how well the EHW controller performed for four trials of 30 s each. The worst performance out of four was used to determined the fitness. "For the final few generations, the evaluations were extended to 90 seconds, to find controllers that were not only good at moving away from walls, but also *staying* away from them" [61].

Although Thompson [61] evolved real hardware controller to control a real physical robot, simulation was still used in fitness evaluation. However, his reason for using simulation appears to be different from our concern about potential risks of evaluating poor controller in a physical environment.

> For convenience, ... The real evolving hardware controlled the real motors, but the wheels were just spinning in the air. The wheels' angular velocities were measured, and used by a real time simulation of the motor characteristics and motor dynamics to calculate how the robot would move. The sonar echo signals were then artificially synthesized and supplied in real time to the hardware DSM. Realistic levels of noise were included in the sensor and motor models, both of which were constructed by fitting curves to experimental measurements, including a probabilistic model for specular sonar reflections [61].

Such an experiment belongs to the category of evolving real hardware in a simulated environment [31, Section 13]. How close the simulated environment (or models) is to the real physical one will have a major impact on the performance of evolved hardware in the real physical environment. The good result achieved by Thompson [61] on the transfer from the simulated to the real environment shows that a simulated environment might be a solution to avoiding the potential risks of evaluating poor controller in a physical environment.

B. EHW Recognizers and Classifiers

EHW recognizers and classifiers refer to those EHW that are used primarily for pattern recognition and classification. Higuchi *et al.* [28], [29], [2], [35], [30], [62] have carried out a number of experiments using EHW to perform various recognition and classification tasks. They used both the gate and function level evolution.

For the gate level evolution, an EHW pattern recognition system was developed to recognize noisy binary input patterns [2], [30], [62]. The input pattern consisted of 8×8 pixels, which were represented by 64 bits. There were three output classes that were represented by 3 bits. During the learning phase, the EHW recognizer was presented with the training patterns. The chromosome representation scheme used was different from that previously adopted by Higuchi *et al.* [28], [29]. A variable-length chromosome representation scheme was used, which only encoded nonempty (nonnil) entries in the connectivity matrix of EHW (FPGA's) [62], [63]. Such a representation generated substantially shorter chromosomes for sparsely connected FPGA's. The GA used was similar to messy GA's with cut and splice operators [64]. The only difference was that duplicated genes would be removed after the splice operation.

The fitness of each individual (i.e., EHW recognizer) was determined by both the error and complexity of the EHW according to the MDL principle [65]. EHW with lower error and lower complexity had higher fitness. Unlike most of the experiments described previously, the EHW recognizer was tested on a separate testing set, which was not used in training. "The test data set consists of 30 patterns which are made by adding some noises into the training patterns. One to five pixels are selected randomly, and the values of the selected pixels are inverted." [62]. Fairly good results, which were average over ten runs, were obtained from the experiment [62].

The gate level evolution was also adopted to develop an EHW comparator used in a V-shape ditch tracer of an industrial welding robot [2]. The EHW was used as a backup system for the conventional logic comparator. It would take over control from the conventional logic comparator only when the conventional logic comparator failed due to circuit faults [2].

For the function level evolution, experiments were carried out with four well-known problems, i.e., the two intertwined spirals, the Iris data set, 2-D image rotation, and synthesis of a four-state automaton [35], [30]. For all of these experiments, an FPGA model consisting of 100 programmable floating processing units (PFU's) was used, which were arranged on a 5×20 grid in a feedforward fashion. That is, the output from one column of PFU's would only be fed into the next column. The two inputs to the FPGA could, however, be fed into any PFU's. A chromosome encoded the information about the function selected by each PFU and the interconnections between PFU's. The variable-length chromosome representation scheme proposed by Kajitani *et al.* [63], which was mentioned above, was used in the experiments. However, the GA used did not have any crossover operators. Only three types of mutations were adopted: operand mutation, function mutation, and insertion [35].

The fitness of each EHW in the function level evolution only considered the error information [35]. The MDL principle was not used. All four problems were investigated from the point of view of EHW's generalization ability. Testing results were given along with the training results. Such experiments were quite different from those aiming at EHW as a design alternative. It was pointed out clearly that the final goal was to achieve online adaptation, although the current work was only concerned with offline adaptation [30], [35].

Other work on the function level evolution [36] include EHW for adaptive equalization in digital mobile communications and lossy data compression.

The driving force behind the function level evolution was to partially address the problem of scalability suffered by the gate level evolution, especially for EHW that would be used in industrial applications.

C. Cellular Programming

A research area closely related to EHW is cellular programming, i.e., evolving cellular automata (CA) by simulated evolution [66]. Sipper *et al.* [66], [55], [67], [68] and Mitchell *et al.* [69], [70] have used EA's to evolve, rather than design by hand, CA's that display complex behaviors and perform complex computations. A hardware implementation was described in [55] and [54].

D. Some Issues and Related Work in Adaptive EHW

Although adaptive EHW might be accused of being a "seductive" phrase, it is used here to distinguish it from evolutionary design of hardware and refer to the EHW that requires generalization ability and online adaptation. There are some fundamental and interesting issues in adaptive EHW that are worth probing further. A comparison with some related work would also help to foster cross fertilization between EHW and other research areas and identify the potential niches of EHW, where its advantages could be fully exploited.

1) Online Adaptation: In spite of the high hope of EHW, no work has been reported on online adaptation by EHW. Only offline adaptation by EHW has been achieved, in which adaptation happens during the learning phase of EHW. It should be noted that online adaptation means adaptation of EHW, while it is executing in a real physical environment. In a sense, online adaptation can also be viewed as real-time adaptation. The meaning of "online" here is different from that used in other contexts, such as online update of connection weights for a backpropagation neural network.

Online adaptation requires EHW's learning to be incremental and responsive. Such requirements do not seem to be met by population-based evolutionary learning, which is used by all EHW at present. The current population-based evolutionary learning is not incremental because, if a new situation occurs as a result of an environmental change, it would have to relearn the new *as well as* old situations to deal with both.

Evolutionary learning at the population level is slow in responding to environmental changes without local learning at the individual level. The population-based learning is global because learning can only be achieved through interactions among different individuals, although it is possible to restrict such interactions to a neighborhood.

It appears that "pure" population-based evolutionary learning would not be sufficient to cope with the requirements of online adaptation. Local learning at the individual level could be introduced to supplement it. Local learning can respond much faster to environmental changes since such response can be made at the local individual level. It has been shown in the area of EANN's that combining evolutionary learning at the population level with local learning at the individual level is feasible and beneficial [71], [25]–[27].

2) Generalization: Generalization is a key issue for any learning or adaptive systems, including EHW. However, studies on this topic are relatively few in the area of EHW. Some experiments on EHW did not address the issue since the same training and testing data set was used, e.g., the experiments with the artificial ant [56] and the four-state automaton [35]. It is unclear how well the EHW could generalize to different situations in these cases. In essence, such experiments demonstrated the effectiveness of EHW as an alternative to circuit design, but not necessarily as an adaptive or learning system.

Most work on testing the generalization ability of EHW was done by Higuchi *et al.* [2], [62], [35], [30]. For the EHW pattern recognizer described in Section III-B, its performance was tested on a noisy test data set different from the training set [62]. For the Iris data set, different training and testing sets were also used [35].

An issue that arises here is whether the maximum fitness value corresponds to the best generalization. The issue is somewhat similar to that raised in Section II-E3. For example, a solution learned by the EHW pattern recognizer for identifying three patterns was $O_0 = I_{50}$, $O_1 = \overline{I_{13}}$, and $O_2 = I_{37}$, where O_i's $(0 \leq i \leq 2)$ were output and I_j's $(0 \leq j \leq 63)$ were input [62]. This was apparently not a good generalization because the output class was determined by a single pixel value. It meant that a single-bit noise at that particular position would change the output of classification. The patterns used in learning EHW recognizer were digits and letters. Recognizing a digit or letter based on the presence or absence of a particular pixel value does not seem to be correct. The fact that the learned EHW had high fitness but not best generalization implies that the EHW recognizer did not learn what we wanted it to learn. It is possible that the training set used in training the EHW recognizer did not contain data of sufficient variety. A better training set should improve EHW's generalization.

Evaluating EHW's generalization can be a difficult task due to different implementations. This difficulty is closely related to that of evaluating the generalization ability of evolutionary learning systems in general. It is not uncommon to read papers that only report a good system evolved at a certain number of generations. It is unclear, however, whether such a good system is the result of one particular run or the average of multiple runs. Statistical analysis of the experimental results seems to be missing. In addition, it is unclear how to decide when to stop to get the good system. A more disciplined approach to experimental studies of generalization in evolutionary learning will greatly help EHW's research.

3) Adaptive EHW and EANN's: EANN's refer to a class of ANN's in which evolution is another form of adaptation in addition to learning [71], [25]–[27]. In particular, EANN's that adapt their architectures through simulated evolution and their weights through learning (training) have been shown to be successful in dealing with a number of benchmark problems [39]–[43], [72].

Adaptive EHW is closely related to EANN's. For example, both the function level EHW (FEHW) [30], [35] and EPNet [39]–[43] evolve feedforward architectures. Both can have different node functions in an architecture [35], [73]. However, node functions in FEHW usually have more variety. There is currently no local learning in FEHW. No weights are associated with connections in FEHW. FEHW's adaptation relies heavily on different compositions of its node functions. In contrast, EPNet uses weights and local learning, but less variety of node functions. It is unknown at present whether FEHW with more node functions without weights would be better than that with less node functions with weights in terms of adaptation and hardware implementation. (It should be pointed out that EPNet is a software package, and it is not targeted at hardware implementation.)

Although EPNet is implemented in software, there are some techniques that might be useful for EHW. For example, EPNet uses validation sets and the order of mutations to improve the generalization ability of learned systems. It grows an ANN by splitting an existing node rather than adding a random one. The process is similar to cell splitting in biology. It deletes or adds a connection by evaluating the importance of the connection first. It also maintains a close behavioral link between parents and offspring, which is important for online adaptation in which we do not want large fluctuations.

Local learning through adjusting weights could be introduced into FEHW since each PFU (i.e., node) in FEHW (implemented by function level FPGA's) has four constant generators that can be used to produce weights for up to four inputs. Such local learning at the individual level can be realized easily. The approach adopted in EPNet could be borrowed or tailored to FHEW. The potential problem might be the speed of such learning. The various types of node function used in EHW will have a major impact on the speed.

4) Adaptive EHW and Genetic Programming: While FEHW and EPNet manipulate acyclic, weighted, and directed graphs by simulated evolution, GP [13] mainly manipulates trees. They are closely related to each other because a tree can be regarded as an acyclic directed graph and an acyclic directed graph can be transformed into a tree by duplicating nodes and branches.

FEHW and GP share the similarity that both adapt function compositions and/or combinations without weights and local learning. But their representations are different. FEHW manipulates acyclic, weighted, and directed graphs by simulated evolution, while GP mainly manipulates trees. Although a tree can be regarded as an acyclic directed graph and an acyclic directed graph can be transformed into a tree by repeating nodes and branches, FEHW is more flexible and general as it can deal with cyclic graphs without much added complexity.

Just as the case in GP, FEHW also requires predefining a set of primitive functions that can be used by each node. One question, which was mentioned briefly in the previous subsection, is why we need more than one node function and what the benefits would be. GP systems that use only one type of node function but with weights, such as the STRONGANOFF system [74], seem to work quite well.

5) Disaster Prevention in Real-Time Online Evolution: It was mentioned in the beginning of Section III that evaluating an EHW in a real physical environment could cause severe damages or disasters to EHW or the physical environment. This potential risk restricts possible applications of EHW in domains in which evaluating EHW in a real physical environment is impractical and an accurate simulation model of the physical environment is difficult to obtain. In most EHW applications, fitness evaluation is the most time-consuming part of the whole evolutionary process. The distinction between intrinsic and extrinsic EHW does not seem to capture this characteristic of EHW. It is only concerned with whether an EHW is reconfigured once or multiple times for each generation [6].

The aforementioned risk stems from the trial-and-error nature of EA's and the black-box approach used by EA's. An EA only evolves chromosomes *syntactically* not *semantically*. It does not understand evolved systems and the environment as no explicit models are used. A possible way to get around this problem is to develop a knowledge-based adaptive EHW, where constraints and knowledge about the environment in which EHW will be evaluated are incorporated into fitness evaluation as its front end, such that any poor individuals that may cause damages to themselves or the environment could be detected and prevented from being passed to the real physical environment.

IV. Behavioral View of EHW

Evolving electronic circuits faces many challenges and open issues. Most of them are caused by the confusion between evolving circuits and evolving circuit behaviors. This confusion should be cleared before any further progress can be made in EHW research. On the surface, it does not seem to make much difference when circuits are evolved or their behaviors are evolved. However, conceptually it is inappropriate to evolve circuits. It is circuit behaviors that should be and can be evolved. It is inherently hard to evolve circuits. Why?

Simulated evolution uses a fitness function to evaluate an EHW individual. What does it actually evaluate? Is it really the EHW circuit (connectivity, functional cells, etc.)? The answer is *no*. It is the circuit's behaviors that are evaluated. The fitness function knows nothing directly about the circuit, and it is not supposed to know it. Since it is the circuit behaviors that are evaluated, the fitness value must depend on the environment in which the EHW circuit is evaluated. Hence, the fitness value is only a measurement of how good the circuit is in *that* environment. It says nothing about the circuit's behaviors in a different environment. This is where the generalization and circuit verification issues start coming in and bothering the EHW research, as discussed in Sections III-D2 and II-E3.

In short, EHW should be regarded as an evolutionary approach to behavior design rather than hardware design. Such a behavioral view of EHW requires a different thinking on EHW circuit design. It is no longer appropriate to talk about what architecture or function a circuit should have. We should start thinking of what behaviors are required from a circuit in certain environments. Then EHW would become a powerful means to evolve and implement such behaviors.

V. EHW that Does Not Change Its Configuration by an EA

This paper is primarily concerned with EHW that uses simulated evolution (notably EA's) to evolve hardware. There are other types of hardware that also use some biological ideas other than evolution. For example, Mange *et al.* [75]–[78], [50]–[52], [49] have been working on self-reproducing and self-repairing hardware based on some ideas from molecular biology (genetics and embryology). The approach used was built on von Neumann's pioneering work on self-reproducing automata [79].

de Garis [80]–[84] used CA in the CAM-BRAIN project exclusively. The aim of the project is to grow and evolve CA-based neural networks (i.e., artificial brains). However, it is unclear what the neural networks are used for since little information has been disclosed about experimental studies on CAM-BRAIN. It is also unclear how the artificial brain is going to learn or evolve after it has "grown up."

VI. Conclusion

This paper reviews the current research on EHW. Emphasis is given to EHW that employs simulated evolution to evolve hardware. A number of issues are raised and discussed. In particular, EHW research needs to address issues, such as scalability, online adaptation, generalization, circuit correctness, and potential risk of evolving hardware in a real physical environment. It is argued that a theoretical foundation of EHW should be established before rushing to large-scale EHW implementations.

This paper also points out some related work, in which some of the techniques could be applied to EHW. Two such related areas are EANN's and GP. EHW is a new research field in the intersection between evolutionary computation and electronics. New progresses in these two fields will continue to provide EHW with new opportunities. For example, recent work on fast hardware for computing exponential and trigonometric functions [85] and work on built-in self-test (BIST) test pattern generators [86] will no doubt widen the range of possible EHW applications.

References

[1] A. J. Hirst, "Notes on the evolution of adaptive hardware," in *Proc. 2nd Int. Conf. Adaptive Comput. Eng. Design (ACEDC'96)*, I. Parmee, Ed.

[2] T. Higuchi, M. Iwata, I. Kajitani, H. Iba, T. Furuya, and B. Manderick, "Evolvable hardware and its applications to pattern recognition and fault-tolerant systems," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 118–135.

[3] P. Graham and B. Nelson, "A hardware genetic algorithm for the traveling salesman problem on splash 2," in *Proc. 5th Int. Workshop Field Programmable Logic Applicat.*, Oxford, U.K., Aug. 1995, pp. 352–361.

[4] S. D. Scott, A. Samal, and S. Seth, "HGA: A hardware based genetic algorithm," in *Proc. ACM/SIGDA 3rd Int. Symp. FPGA's*, 1995, pp. 53–59.

[5] M. Salami and G. Cain, "Implementation of genetic algorithms on reprogrammable architectures," in *Proc. 8th Australian Joint Conf. Artif. Intell. (AI'95)*, X. Yao, Ed. Singapore: World Scientific, 1995, p. 581.

[6] H. de Garis, "LSL evolvable hardware workshop report," ATR, Japan, Tech. Rep., Oct. 1995.

[7] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, and A. Stauffer, "Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 35–54.

[8] M. P. Fourman, "Compaction of symbolic layout using genetic algorithms," in *Proc. 1st Int. Conf. Genetic Algorithms Their Applicat.*, J. J. Grefenstette, Ed. Pittsburgh, PA: Carnegie Mellon Univ. Press, 1985, pp. 141–153.

[9] J. Cohoon and W. Paris, "Genetic placement," in *Proc. Int. Conf. Computer-Aided Design*. New York: IEEE Press, 1986, pp. 422–425.

[10] R. M. Kling and P. Banerjee, "ESP: Placement by simulated evolution," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 245–256, Mar. 1989.

[11] H. Hemmi, J. Mizoguchi, and K. Shimohara, "Development and evolution of hardware behaviors," in *Artificial Life IV: Proc. 4th Int. Workshop Synthesis Simulation Living Syst.*, R. Brooks and P. Maes, Eds. Cambridge, MA: MIT Press, 1994, pp. 371–376.

[12] H. Hemmi, T. Hikage, and K. Shimohara, "AdAM: A hardware evolutionary system," in *Proc. 1997 IEEE Conf. Evolutionary Computat. (ICEC'97)*. Piscataway, NJ: IEEE Press, pp. 193–196.

[13] J. R. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[14] ——, *Genetic Programming II*. Cambridge, MA: MIT Press, 1994.

[15] P. Whigham, "Grammatically-based genetic programming," in *Proc. Workshop Genetic Programming: From Theory to Real-World Applicat.*, J. Rosca, Ed. New York: Morgan Kaufmann, July 1995, pp. 33–41.

[16] ——, "Inductive bias and genetic programming," in *Proc. Inst. Elect. Eng. 1st Int. Conf. Genetic Algorithms Eng. Syst.: Innovat. Applicat.*, Sept. 1995, pp. 461–466.

[17] P. Whigham and R. I. McKay, "Genetic approaches to learning recursive relations," in *Progress in Evolutionary Computation*, vol. 956, X. Yao, Ed. Heidelberg, Germany: Springer-Verlag, 1995, pp. 17–27.

[18] P. Whigham, "A schema theorem for context-free grammars," in *Proc. 1995 IEEE Int. Conf. Evolutionary Computat. (ICEC'95)*. Piscataway, NJ: IEEE Press, vol. 1, pp. 178–182.

[19] W. Kruiskamp and D. Leenaerts, "DARWIN: CMOS opamp synthesis by means of a genetic algorithm," in *Proc. 32nd Design Auto. Conf.*. New York: ACM, 1995, pp. 433–438.

[20] J. R. Koza, F. H. Bennett, III, D. Andre, and M. A. Keane, "Four problems for which a computer program evolved by genetic programming is competitive with human performance," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computat. (ICEC'96)*. Piscataway, NJ: IEEE, pp. 1–10.

[21] J. R. Koza, D. Andre, F. H. Bennett, III, and M. A. Keane, "Use of automatically defined functions and architecture-altering operations in automated circuit synthesis using genetic programming," in *Genetic Programming 1996: Proc. 1st Annu. Conf. (GP'96)*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, pp. 132–140.

[22] J. R. Koza, F. H. Bennett, III, D. Andre, and M. A. Keane, "Automated WYWIWYG design of both the topology and component values of electrical circuits using genetic programming," in *Genetic Programming*

1996: Proc. 1st Annu. Conf. (GP'96), J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA, MIT Press, pp. 123–131.

[23] F. H. Bennett, III, J. R. Koza, D. Andre, and M. A. Keane, "Evolution of a 60 Decibel op amp using genetic programming," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware (ICES'96)*, vol. 1259, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 455–469.

[24] J. R. Koza, F. H. Bennett, III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 109–128, Feb. 1997.

[25] X. Yao, "A review of evolutionary artificial neural networks," *Int. J. Intell. Syst.*, vol. 8, no. 4, pp. 539–567, 1993.

[26] ——, "Evolutionary artificial neural networks," *Int. J. Neural Syst.*, vol. 4, no. 3, pp. 203–222, 1993.

[27] ——, "Evolutionary artificial neural networks," in *Encyclopedia of Computer Science and Technology*, vol. 33, A. Kent and J. G. Williams, Eds. New York: Marcel Dekker, 1995, pp. 137–170.

[28] T. Higuchi, T. Niwa, T. Tanaka, H. Iba, H. de Garis, and T. Furuya, "Evolving hardware with genetic learning: A first step toward building a Darwin machine," in *Proc. 2nd Int. Conf. Simulation Adaptive Behavior (SAB'92)*. Cambridge, MA: MIT Press, pp. 417–424.

[29] T. Higuchi, H. Iba, and B. Manderick, "Evolvable hardware," in *Massively Parallel Artificial Intelligence*, H. Kitano and J. Hendler, Eds. Cambridge, MA: MIT Press, 1994, pp. 398–421.

[30] T. Higuchi, M. Iwata, I. Kaijitani, M. Murakawa, S. Yoshizawa, and T. Furuya, "Hardware evolution at gate and function level," in *Proc. Int. Conf. Biologically Inspired Autonomous Syst.: Computation, Cognition Action*, Durham, NC, Mar. 4–5, 1996.

[31] A. Thompson, I. Harvey, and P. Husbands, "Unconstrained evolution and hard consequences," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 136–165.

[32] A. Thompson, "Silicon evolution," in *Genetic Programming 1996: Proc. 1st Annu. Conf. (GP'96)*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, pp. 444–452.

[33] ——, "An evolved circuit, intrinsic in silicon, entwined with physics," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 390–405.

[34] I. Harvey and A. Thompson, "Through the labyrinth evolution finds a way: A silicon ridge," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 406–422.

[35] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, "Hardware evolution at function level," in *Proc. Int. Conf. Parallel Problem Solving from Nature (PPSN'96)*, 1996.

[36] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. Liu, and M. Salami, "Evolvable hardware at function level," in *Proc. 1997 IEEE Conf. Evolutionary Computat. (ICEC'97)*. Piscataway, NJ: IEEE Press, pp. 187–192.

[37] W. Liu, M. Murakawa, and T. Higuchi, "ATM cell scheduling by function level evolvable hardware," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 180–192.

[38] M. Murakawa, S. Yoshizawa, and T. Higuchi, "Adaptive equalization of digital communication channels using evolvable hardware," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware (ICES'96)*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 379–389.

[39] X. Yao and Y. Liu, "A new evolutionary system for evolving artificial neural networks," *IEEE Trans. Neural Networks*, vol. 8, pp. 694–713, May 1997.

[40] ——, "Toward designing artificial neural networks by evolution," *Appl. Math. Comput.*, vol. 91, no. 1, pp. 83–90, 1998.

[41] ——, "Evolving artificial neural networks through evolutionary programming," in *Evolutionary Programming V: Proc. 5th Annu. Conf. Evolutionary Programming*, L. J. Fogel, P. J. Angeline, and T. Bäck, Eds. Cambridge, MA, MIT Press, 1996, pp. 257–266.

[42] ——, "Evolutionary artificial neural networks that learn and generalize well," in *1996 IEEE Int. Conf. Neural Networks*, Washington, DC. New York: IEEE Press, pp. 159–164.

[43] Y. Liu and X. Yao, "A population-based learning algorithm which learns both architectures and weights of neural networks," *Chin. J. Adv. Softw. Res.*, vol. 3, no. 1, pp. 54–65, 1996.

[44] M. A. Rosenman, "An evolutionary model for nonroutine design," in *Proc. 8th Australian Joint Conf. Artif. Intell. (AI'95)*, X. Yao, Ed. Singapore: World Scientific, pp. 363–370.

[45] K. Sims, "Artificial evolution for computer graphics," *Comput. Graph.*, vol. 25, no. 4, pp. 319–328, 1991.

[46] J. J. Grefenstette, "Incorporating problem specific knowledge into genetic algorithms," in *Genetic Algorithms and Simulated Annealing*, L. Davis, Ed. San Mateo, CA: Morgan Kaufmann, 1987, pp. 42–60.

[47] L. Davis, *Handbook of Genetic Algorithms*, New York: Van Nostrand Reinhold, 1991.

[48] H. Hemmi, J. Mizoguchi, and K. Shimohara, "Development and evolution of hardware behaviors," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 250–265.

[49] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand, "Embryonics: A new family of coarse-grained field programmable gate array with self-repair and self-reproducing properties," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, pp. 197–200, 1996.

[50] G. Tempesti, D. Mange, and A. Stauffer, "A self-repairing FPGA inspired by biology," in *Proc. 3rd IEEE Int. On-Line Testing Workshop*, IEEE Comput. Soc., 1997, pp. 191–195.

[51] D. Mange, A. Stauffer, and G. Tempesti, "Self-replicating and self-repairing field-programmable processor arrays (FPPA's) with universal computation," in *Proc. IJCAI'97 Workshop Evolvable Syst.*, T. Higuchi, Ed., Nagoya, Japan, pp. 7–11.

[52] ——, "Self-replicating and self-repairing field-programmable processor arrays (FPPA's) with universal construction," in *Proc. IJCAI'97 Workshop Evolvable Syst.*, T. Higuchi, Ed., Nagoya, Japan, pp. 13–18.

[53] A. Pérez-Uribe and E. Sanchez, "Neural network structure optimization through on-line hardware evolution," in *Proc. World Congr. Neural Networks (WCNN'96)*, San Diego, CA, pp. 1041–1044.

[54] M. Sipper, M. Goeke, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini, "The firefly machine: Online evolware," in *Proc. 1997 IEEE Conf. Evolutionary Computat. (ICEC'97)*. Piscataway, NJ: IEEE, pp. 181–186.

[55] M. Goeke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, and M. Tomassini, "Online autonomous evolware," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, vol. 1259, pp. 96–106.

[56] J. Mizoguchi, H. Hemmi, and K. Shimohara, "Production genetic algorithms for automated hardware design through an evolutionary process," in *Proc. 1st IEEE Conf. Evolutionary Computat. (ICEC'94)*, Z. M. *et al.*, Eds. Piscataway, NJ: IEEE, vol. I, pp. 661–664.

[57] D. Keymeulen, M. Durantez, K. Konaka, Y. Kuniyoshi, and T. Higuchi, "An evolutionary robot navigation system using a gate-level evolvable hardware," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu , Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 195–209.

[58] T. Naito, R. Odagiri, Y. Matsunaga, M. Tanifuji, and K. Murase, "Genetic evolution of a logic circuit which controls an autonomous mobile robot," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 210–219.

[59] J. Yamamoto and Y. Anzai, "Autonomous robot with evolving algorithm based on biological systems," in *Proc. 1st Int. Conf. Evolvable Syst.: From Biology to Hardware*, T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, vol. 1259, pp. 220–233.

[60] H. H. Lund, J. Hallam, and W.-P. Lee, "Evolving robot morphology," in *Proc. 1997 IEEE Conf. Evolutionary Computat. (ICEC'97)*. Piscataway, NJ: IEEE, pp. 197–202.

[61] A. Thompson, "Evolving electronic robot controllers that exploit hardware resources," in *Proc. 3rd Eur. Conf. Artif. Life (ECAL'95)*. Berlin, Germany: Springer-Verlag, pp. 640–656.

[62] M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi, "A pattern recognition system using evolvable hardware," in *Proc. Int. Conf. Parallel Probl. Solving Nature (PPSN'96)*.

[63] I. Kajitani, T. Hoshino, M. Iwata, and T. Higuchi, "Variable length chromosome GA for evolvable hardware," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computat. (ICEC'96)*. Piscataway, NJ: IEEE Press, pp. 443–447.

[64] D. E. Goldberg, K. Deb, H. Kargupta, and G. Harik, "Rapid, accurate optimization of difficult problems using fast messy genetic algorithms," in *Proc. 5th Int. Conf. Genetic Algorithms (ICGA'93)*, S. Forrest, Ed. New York: Morgan Kaufmann, pp. 56–64.

[65] J. Rissanen, *Stochastic Complexity in Statistical Inquiry*. Singapore: World Scientific, 1989.

[66] M. Sipper, "Designing evolware by cellular programming," in *Proc. 1st Int. Conf. Evolvable Syst/: From Biology to Hardware*, vol. 1259,

T. Higuchi, M. Iwata, and W. Liu, Eds. Berlin, Germany: Springer-Verlag, 1997, pp. 81–95.

[67] ——, "Co-evolving nonuniform cellular automata to perform computations," *Physica D*, vol. 92, pp. 193–208, 1996.

[68] M. Sipper and E. Ruppin, "Co-evolving cellular architectures by cellular programming," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computat. (ICEC'96)*. Piscataway, NJ: IEEE Press, pp. 306–311.

[69] M. Mitchell, J. P. Crutchfield, and P. T. Hraber, "Evolving cellular automata to perform computations: Mechanisms and impediments," *Physica D*, vol. 75, pp. 361–391, 1994.

[70] M. Mitchell, P. T. Hraber and J. P. Crutchfield, "Revisiting the edge of chaos: Evolving cellular automata to perform computations," *Complex Syst.*, vol. 7, pp. 89–130, 1993.

[71] X. Yao, "Evolution of connectionist networks," in *Preprints Int. Symp. Artif. Intell., Reasoning Creativity*, T. Dartnall, Ed. Queensland, Australia: Griffith Univ. Press, 1991, pp. 49–52.

[72] A. Pérez-Uribe and E. Sanchez, "FPGA implementation of an adaptable-size neural network," in *Proc. 6th Int. Conf. Artif. Neural Networks—ICANN'96*, C. V. D. Malsburg, J. C. V. W. V. Seelen, and B. Sendhoff, Eds. Berlin, Germany: Springer-Verlag, vol. 1112, pp. 383–388.

[73] Y. Liu and X. Yao, "Evolutionary design of artificial neural networks with different nodes," in *Proc. 1996 IEEE Int. Conf. Evolutionary Computat. (ICEC'96)*, Nagoya, Japan. New York: IEEE, pp. 670–675.

[74] H. Iba, H. de Garis, and T. Sato, "A numerical approach to genetic programming for system identification," *Evolutionary Computat.*, vol. 3, no. 4, pp. 417–452, 1996.

[75] D. Mange and A. Stauffer, "Introduction to embryonics: Toward new self-repairing and self-reproducing hardware based on biological-like properties," in *Artificial Life and Virtual Reality*, N. M. Thalmann and D. Thalmann, Eds. Chichester, U.K.: Wiley, 1994, pp. 61–72.

[76] P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand, "Achieving von Neumann's dream: Artificial life on silicon," in *Proc. IEEE Int. Conf. Neural Networks (ICNN'94)*, vol. IV, pp. 2321–2326.

[77] S. Durand, A. Stauffer, and D. Mange, "Biodule: An introduction to digital biology," Logic Syst. Lab., Swiss Federal Inst. Technol., Lausanne, Switzerland, Tech. Rep., Preliminary Rep., Sept. 1994.

[78] D. Mange, S. Durand, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, and C. Piguet, "A new self-reproducing automaton based on a multicellular organization," Logic Syst. Lab., Swiss Federal Inst. Technol., Lausanne, Switzerland, Tech. Rep. 95/114, Apr. 1995.

[79] J. von Neumann, *The Theory of Self-Reproducing Automata*. Urbana, IL: Univ. Illinois Press, 1966.

[80] H. de Garis, "Evolvable hardware: Genetic programming of a Darwin machine," in *Proc. Int. Conf. Artif. Neural Nets Genetic Algorithms*, R. F. Albrecht, C. R. Reeves, and N. C. Steele, Eds. Innsbruck, Austria: Springer-Verlag, 1993, pp. 441–449.

[81] ——, "An artificial brain," *New Generation Computing*, vol. 12, pp. 215–221, 1994.

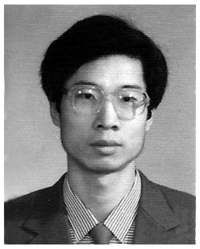
[82] ——, "CAM-BRAIN—The evolutionary engineering of a billion neuron artificial brain by 2001 which grows/evolves at electronic speeds inside a cellular automata machine (CAM)," in *Proc. Int. Conf. Artif. Neural Nets Genetic Algorithms, Alés*, D. W. Pearson, N. C. Steele, and R. F. Albrecht, Eds. France, Springer-Verlag, 1995, pp. 84–87.

[83] ——, "CAM-BRAIN: The evolutionary engineering of a billion neuron artificial brain by 2000 which grows/evolves at electronic speeds inside a cellular automata machine (cam)," in *Toward Evolvable Hardware: The Evolutionary Engineering Approach*, vol. 1062, E. Sanchez and M. Tomassini, Eds. Berlin, Germany: Springer-Verlag, 1996, pp. 76–98.

[84] ——, "CAM-BRAIN issues: Implementation and performance-scaling issues concerning the genetic programming of a cellular automata based artificial brain," in *Proc. IEEE Int. Conference Neural Networks (ICNN'94)*, vol. III, pp. 1714–1720.

[85] V. Kantabutra, "On hardware for computing exponential and trigonometric functions," *IEEE Trans. Comput.*, vol. 45, pp. 328–339, Mar. 1996.

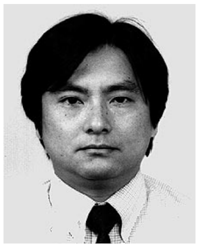
[86] C.-A. Chen and S. K. Gupta, "BIST test pattern generators for two-pattern testing—Theory and design algorithms," *IEEE Trans. Comput.*, vol. 45, pp. 257–269, Mar. 1996.



Xin Yao (SM'96) received the B.Sc. degree from the University of Science and Technology of China (USTC) in 1982, the M.Sc. degree from the North China Institute of Computing Technologies (NCI) in 1985, and the Ph.D. degree from USTC in 1990.

He held postdoctoral fellowships in the Australian National University (ANU) and the Commonwealth Scientific and Industrial Research Organization (CSIRO). He is an Associate Professor in the School of Computer Science, University College, the University of New South Wales (UNSW), Australian Defence Force Academy (ADFA), Canberra. He has published a number of papers in the fields of evolutionary computation and neural networks.

Dr. Yao is the program committee Co-Chair of the IEEE ICEC'97 in Indianapolis, IN, and Co-Vice-Chair of IEEE ICEC'98 in Anchorage, AK. He is/was also a program committee Chair/Co-Chair of the Third International Conference on Computational Intelligence and Multimedia Applications (ICCIMA'99), the Second Asia-Pacific Conference on Simulated Evolution And Learning (SEAL'98), ICCIMA'97, SEAL'96, the Eighth Australian Joint Conference on AI (AI'95), and a program committee member of many other international conferences. He is an Associate Editor of the IEEE TRANSACTIONS ON EVOLUTIONARY COMPUTATION, a member of the IEEE NNC Technical Committee on Evolutionary Computation, and the second Vice-President of the Evolutionary Programming Society.



Testuya Higuchi received the B.E., M.E., and Ph.D. degrees, all in electrical engineering from Keio University, Japan, in 1978, 1980, and 1984, respectively.

He heads the Evolvable Systems Laboratory in the Electrotechnical Laboratory, MITI, Tsukuba, Japan. His current research interests include evolvable hardware systems, parallel processing architectures in artificial intelligence, and adaptive systems. He is also in charge of the Adaptive Devices Group in the MITI national project, Real World Computing Program. He was the General Chair of the First International Conference on Evolvable Systems: From Biology to Hardware (ICES'96) and the IJCAI'97 Workshop on Evolvable Hardware.