Chapter 12

EVOLVING HARDWARE

Timothy G. W. Gordon and Peter J. Bentley
University College London

1 INTRODUCTION

In the hundred years since John Ambrose Fleming invented the diode at University College London and gave birth to the field, electronics has become a well-understood engineering discipline. This solid grounding of knowledge has allowed the commercial semiconductor industry to grow at a remarkable rate in the intervening years, both in volume and in the complexity of hardware. As a result, the now-famous Moore's Law has held true for almost forty years [85]. But problems are beginning to emerge. For the industry to flourish, the growth in hardware complexity must continue, but it is becoming clear that current design methodologies applied to silicon-based technologies can no longer support the present rate of scaling.

In the medium term, the requirement for new and innovative designs is set to grow as it becomes necessary to squeeze more and more out of the technologies we already have. The long-term solution is likely to lie in the development of new circuit medium technologies. But even when new circuit media do eventually become commercially feasible, they are likely at best to require features in our designs that our current circuit methodologies are not aimed at providing, such as fault tolerance, and at worst require a complete rewriting of the design rules. So it is clear that there is a significant requirement for innovative circuit designs and design methodologies, and the cost of developing these in man-hours of research and design is likely to be considerable.

Over the past decade, a new field applying evolutionary techniques to hardware design and synthesis has emerged. These techniques may be able to give us a new option. We can use evolution to design automatically, or at least aid in the design and realization of innovative circuits. This field has been coined *evolutionary electronics*, *hardware evolution*, and *evolvable hardware*, amongst others. Here it will be referred to as *evolvable hardware*.

The field of evolvable hardware draws inspiration from a range of other fields, as shown in Figure 12.1. For many years computer scientists have modeled their learning algorithms on self-organizing processes observed in nature. Perhaps the most well-known example is the artificial neural network (ANN) [93]. Others include the collective decision-making of ant colonies [12], the adaptive ability of immune systems [98], the growth of self-similar structures in plants [64], and of course Darwinian evolution [19]. Collectively, work on such algorithms is known as *bio-inspired software*, which is shown at the intersection of Computer Science and Biology in Figure 12.1.

Ideas from nature have also been used in electronic engineering for many years; for instance, simulated annealing algorithms are used in many circuit partitioning algorithms. (Simulated annealing algorithms are based on the physical phenomenon of annealing in cooling metals.) Interest in using ideas from nature has grown in recent years to the extent that the field of bio-inspired hardware is now firmly established in its own right. This field uses many of the ideas adopted from nature by software developers, and some new ones, to allow fault tolerance, reconfigurability, and even automatic circuit design in modern hardware. The field of evolvable hardware is shown at the intersection of Computer Science, Biology, and Electronic Engineering in Figure 12.1. The focus of this chapter is in this central area.

The interrelationships between areas of hardware design and synthesis, and evolutionary computation are shown in Figure 12.2. Digital hardware synthesis is traditionally a combination of two processes. First, a human-designed circuit specification is mapped to a logical representation through the process of logic synthesis. This is represented as the lower right-hand set in Figure 12.2. This netlist then undergoes further combinatorially complex optimization processes in order to place and route the circuit to the target technology. This area is represented as the lower left-hand set in Figure 12.2. Many modern electronic design automation (EDA)[1] tools use intelligent techniques in these optimization algorithms, and research into the use of evolution for these purposes abounds [18, 74]. Hence we see the set representing evolutionary design intersect with that of

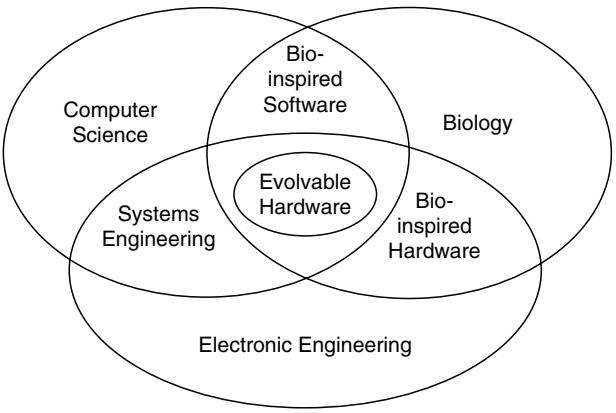

Figure 12.1. The field of evolvable hardware originates from the intersection of three sciences

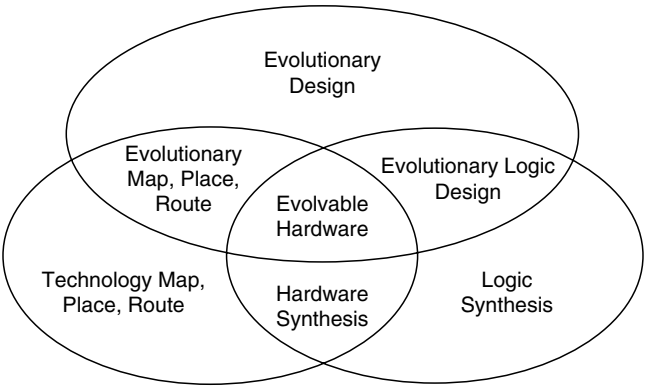

Figure 12.2. Evolvable hardware can include aspects of hardware design and optimization techniques

technology mapping, placement, and routing in Figure 12.2 to yield evolutionary mapping, placement, and routing. However, circuit design, along with some optimization decisions during the synthesis process, is still in the domain of the human designer. It has only been recently that significant interest has developed in implementing evolutionary techniques higher up the VLSI design flow at circuit design, a move that can allow evolution to generate creative designs that can rival or improve on human ones. The most widespread examples of this have been to use evolution for the design of logic, as represented by the intersection of the areas of evolutionary design and logic synthesis in Figure 12.2. Some of the work in this intersection falls into the field of evolvable hardware. However, much work at the logical level is carried out in the spirit of evolving programs or other forms of logic, and so is beyond the scope of this chapter.

The rest of the chapter is organized as follows. Section 2 begins with a brief discussion of how evolvable hardware can be realized, and surveys its areas of application. The field is still young, and there are several problems that must be tackled before large-scale commercial use of the techniques will become viable. Section 3 discusses key past and current research into evolvable hardware by focusing on the two largest and most actively researched of these problems, namely, generalization and evolvability, along with the most important benefit of evolvable hardware in our eyes: innovation. We use "level of abstraction," "learning bias," and "evolving platform" as the main features to map out this research.

A distinction commonly made is the difference between *extrinsic evolution*, where candidate circuit designs are evaluated in simulation, and *intrinsic evolution*, where candidate circuit designs are synthesized and evaluated directly on programmable logic devices (PLDs). In the case of circuits evolved intrinsically, the choice of platform used can have a profound effect on evolution's performance. Criteria for choosing a suitable platform are discussed at the end of Section 3, along with an appraisal of platforms that have been used for evolvable hardware to date. Section 4 presents some of our recent work into a new branch of evolvable hardware, developmental hardware evolution, which has the potential to solve many of the evolvability issues in the field. Finally, a summary will be given in Section 5.

2 EVOLVABLE HARDWARE IN PRACTICE

Evolutionary Computation is the field of solving problems using search algorithms inspired by biological evolution. These algorithms are collectively known as *evolutionary algorithms*. They model the principles of selection, variation, and inheritance that are the basis of the theory of Darwinian evolution, and have been applied to a huge spectrum of problems, from classic optimization [52] to the creation of original music [5]. Typically they work on a population of prospective solutions in parallel. Each member of the population is evaluated according to a problem-specific *fitness function* that tests how well each solution performs a required task and then assigns that solution a fitness score. A *selection* operator then probabilistically chooses solutions with higher fitness from the population to form the basis of a new generation of solutions. These solutions are then varied, commonly by randomly altering each solution to model *mutation* and/or by recombining two solutions in some way to model sexual reproduction—a procedure commonly called *crossover*. The process is then iterated for a number of generations until a stopping condition is met, for instance, the discovery of a solution with a given fitness or the completion of a predefined number of generations.

This chapter concerns the application of evolutionary algorithms to the automatic design of electronic circuits. In order to familiarize the reader with how circuits might be evolved, an example is now presented.

2.1 An Example of Evolvable Hardware

The class of evolutionary algorithms most commonly used in evolvable hardware is the *genetic algorithm*. Most commonly, these operate on a fixed-size population of fixed-length binary strings called *chromosomes*. Each chromosome encodes a common set of parameters that describe a collection of electronic components and their interconnections. Thus, each set of parameter values represents an electronic circuit. The set of all possible combinations of parameter values defines the *search space* of the algorithm, and the circuits that they represent define the *solution space* of the algorithm. Traditionally, every parameter set in the search space encodes a unique circuit description in the solution space. For every chromosome/circuit pair, the chromosome is called the *genotype* and the circuit is called the *phenotype*.

An example of evolvable hardware is shown in Figure 12.3. The algorithm begins by initializing the bits of each chromosome with random values. The chromosomes are then evaluated in turn by creating a circuit based on the parameter values, either as a simulated model of the circuit or as a concrete circuit embodied in reconfigurable hardware (an example of which is shown in Section 5). The circuit's fitness for performing the target task is then measured by passing to it a set of test values and evaluating the veracity of the circuit's output. The selection operator then probabilistically populates the next generation of chromosomes such that chromosomes with high fitness are more likely to be selected. There are many methods to achieve this, a common approach being *two-member tournament selection* [19]: the operator selects two individuals at random and compares their fitness. Only the individual with the highest fitness is inserted into the next

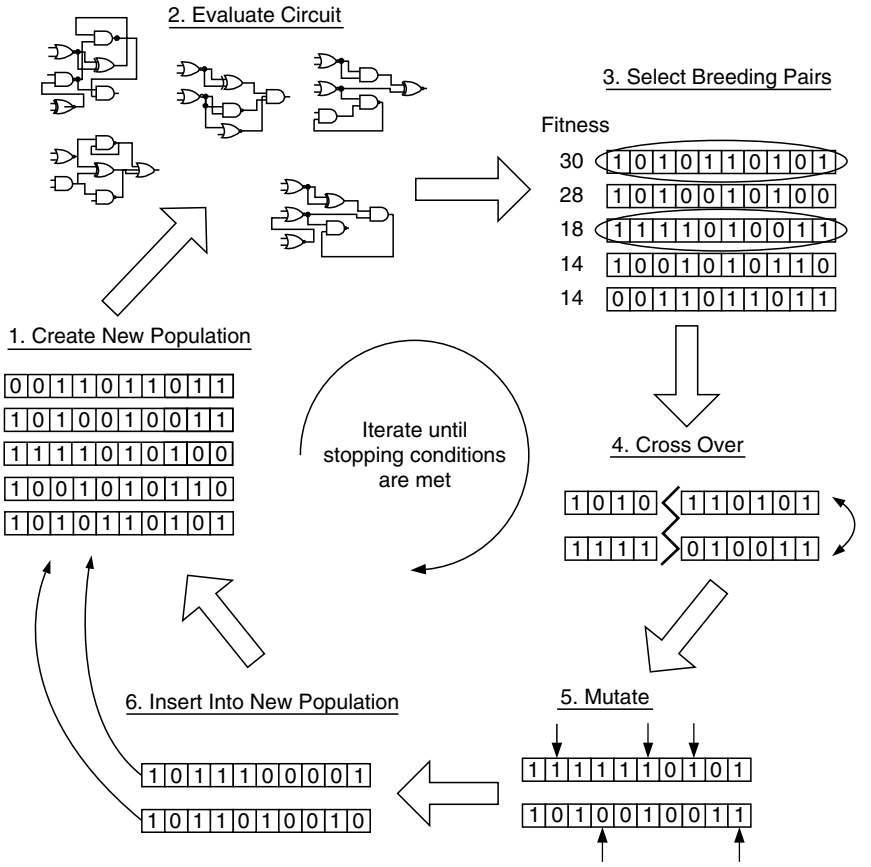

Figure 12.3. An example of evolvable hardware

generation. If the two have equal fitness, the individual to be inserted is chosen at random. Once the new population has been selected, it is varied. Common variation operators are *one-point crossover* and *point mutation* [19]. One-point crossover recombines two chromosomes by choosing a position at random along the chromosome and swapping every bit beyond this point between the strings. It is stochastically applied according to a fixed probability. Point mutation independently inverts each bit in the chromosome according to a fixed probability. These operators are applied to all members of the new population. Often, in addition to these operators, the best member of the original population is copied into the new population unchanged, a strategy called *elitism* [19]. The new population is now complete, and the algorithm then iterates the steps of evaluation, selection, and variation until a circuit that functions adequately is found or a prespecified number of generations is completed.

Using evolution to design circuits in this way brings a number of important benefits to electronics, allowing design automation and innovation for an increasing range of applications. Some of the more important areas where evolvable hardware can be applied include the following:

- Automatic design of low-cost hardware
- Coping with poorly specified problems
- Creation of adaptive systems
- Creation of fault-tolerant systems
- Innovation in poorly understood design spaces

The remainder of this section will explore research in these areas in a little more detail.

2.2 Automatic Design of Low-Cost Hardware

Automation has been used in circuit synthesis for many years. Traditional digital design involves the mapping of an abstract human-designed circuit to a specific technology through the application of simple minimization, placement, and routing rules. As our capability for synthesizing more complex circuits has grown, so has the need for more resourceful processes to handle the combinatorially complex mapping procedures. Intelligent techniques such as simulated annealing [97] and ANNs [133] have been routinely used to search these exploding spaces of mappings and minimizations for some time. More recently, so has evolution [18, 74].

Evolvable hardware allows us to take the automation of circuit production a step further, automating how to generate the actual circuit design from a behavioral specification and simultaneously automating the circuit synthesis process. The behavioral specification presented to the evolvable system may be as simple as a series of circuit input signals that the system must match to a corresponding predefined set of output signals, although other representations of circuit behavior may be used, often including environmental conditions or simulated error test cases or depending on the requirements of the circuit. How the representation and contents of the circuit specification affect the functionality of circuits is currently the center of much interest and is discussed in more detail under the heading of *generalization* in Section 3.

In applications where a suitable behavioral specification has been found, evolvable hardware can remove the necessity for a designer, or at least reduce the design time that is required, thus reducing production costs. This advantage is particularly useful when design costs are a significant proportion of total cost, for instance for hardware that is produced in low volumes. Evolvable hardware even allows us to evolve designs to suit an individual. Many medical applications have not been suitable for hardware solutions owing to the expense of personalization. Evolvable hardware allows cheap, fast solutions to such applications. For example, a system has been developed to control a prosthetic hand by recognition of patterns of myoelectric signals in a user's arm [45]. The implementation is an entirely hardware-based solution with reconfigurable logic, a hardware genetic algorithm unit, a CPU core for evaluation, a chromosome memory, and a random number generator implemented on the same integrated chip.

Evolution can also be used to reduce production costs on larger scales by optimizing circuits that fail to meet their required specifications due to variations during fabrication. For instance, [88] corrected variations in the frequency of

intermediate frequency filters by using evolution to control the output of a series of transconductance amplifiers. This is a useful tool in the case of analogue circuit designs, where individual component behaviors can vary quite markedly, and in particular for designs where power and size are important, since tuning the components in this way allows smaller, low-power components to be used. In light of this, intermediate frequency filters tuned using this technique are already in use in commercial mobile communications products [87]. The idea of using evolution to counteract fabrication variations has also been explored for digital circuits. For instance, Takahashi, Kasai, et al. incorporated programmable delay elements in the registers of a memory test pattern generator [112], thus allowing the evolved circuits to compensate for not only clock skew but also any variations in data delays throughout the circuit. Simulation results demonstrated that an almost 50% improvement in production yield was possible using this method. Such improvements in yield can reduce production costs considerably.

2.3 Poorly Specified Problems

For some problems, it is difficult to specify functionality succinctly but easy to specify a behavioral description. Computer scientists have used evolution to handle problems with such poor specifications for many years. ANNs have been applied to problems such as noisy pattern recognition [93]. Evolvable hardware techniques have similarities with and advantages over ANNs, as noted by Yao and Higuchi [132]. Both can be feed-forward networks, and both can learn nonlinear functions successfully. But, in addition, hardware is by nature a fast medium, and in many cases, such as when restricted to feed-forward networks, evolved hardware designs are more easily understood than ANNs. Therefore this approach is often suited to problems usually tackled with ANNs but that require fast operation and good solution tractability. Evolvable hardware suitable for such purposes has already been developed for industrial use [88].

One problem where evolved hardware can rival ANNs is pattern recognition. For example, Sekanina has successfully evolved image noise filters that rival the best traditionally designed circuits [100]. One of the advantages of evolutionary systems is the ease with which learning biases can be incorporated. For instance, Higuchi et al. have evolved high-speed robust classifiers [34, 42] Good generalization characteristics were incorporated into the solutions by specification of a bias based on machine learning theory. More recently, do Amaral et al. evolved fuzzy functions that can be used as building blocks in the construction of fuzzy logic controllers [11].

2.4 Adaptive systems

With sufficient automation (i.e., real-time synthesis provided by PLDs), evolvable hardware has the potential to adapt autonomously to changes in its environment. This ability can be very useful in situations where real-time manual control over systems is not possible, such as on deep space missions. It could be particularly useful when unexpected conditions are encountered.

Stoica et al. have noted that current lack of validation for online evolutionary systems means that critical spacecraft control systems, and other mission-critical

systems, cannot currently be placed under evolutionary control [104]. Greenwood and Song have proposed using evolutionary techniques in conjunction with formal verification techniques to circumvent this problem [22]; however, to date only noncritical systems such as sensor processing systems have been explored, for example, adaptive data compression systems [15]. Other systems that could benefit from the ability to autonomously evolve are power management systems and controller deployment mechanisms for booms, antennae, etc. [91]

Several other adaptive hardware compression systems have also been developed. Two systems have been developed at the Electrotechnical Lab. (ETL), both using predictive coding. The first predicts each pixel, using a standard prediction function, from a subset of surrounding pixels selected by a genetic algorithm. It has proved successful in compressing bi-level images for high precision electrophotographic printers, outperforming JBIG, the ISO standard for bi-level image compression, by an average of around 50%. Since then the method has been proposed as a new ISO standard [94]. The second approach breaks images into smaller sections and uses evolution to model a function for each section [95]. They also suggested that a similar system could be used for real-time adaptive compression of video signals. A similar technique was used by Sekanina to evolve adaptive circuits that filter image noise in changing environments [99].

Many other adaptive filters have been evolved, including digital finite impulse response (FIR) filters, commonly used in audio applications such as noise and echo cancellation [125, 131] and their more complex but less reliable counterparts, infinite impulse response (IIR) filters [100]. Analogue adaptive filters have also been evolved. For example, Zebulum et al. presented signal extraction filters capable of adaptively amplifying the strongest component of the input signal while attenuating others, thus improving a hypothetical signal/noise ratio [135]. Through evolution, these circuits could be adapted to new input profiles.

Online scheduling hardware has also been developed, most notably adaptive cell scheduling systems for ATM networks, that responds to changes in traffic flow [59, 65]. In a related field, Damiani et al. have developed an online adaptive hashing system that could be used to map cache blocks to cache tags dependent on the access patterns of the data over time [9].

2.5 Fault-Tolerant Systems

Ongoing advances in component miniaturization have not been complemented by improvements in fabrication reliability. This means that many modern VLSI circuit designs must be tolerant to fabrication faults. It is expected that this issue will become even more important in future circuit technologies. Miniaturization also exposes components to a greater risk of operational faults—for instance, due to the effects of power fluctuations or ionizing radiation. Reliability is of paramount importance for many systems, such as medical equipment and transport control systems. Military and spacecraft systems are particularly susceptible to reliability problems, as they are regularly subjected to harsh conditions. Current techniques for fault tolerance rely on the presence of additional redundant components and thorough testing, either at the point of manufacture or online, and add considerable cost and design complexity. Fortunately, evolvable hardware provides a number of mechanisms to introduce fault tolerance into circuits.

A class of adaptive system that was not mentioned in Section 2.4 comprises circuits that can adapt to faults in their own hardware, thus providing a mechanism of fault recovery. An early demonstration of this ability was that of Higuchi et al. [34], where an adaptive hardware system learned the behavior of an expert robot controller by example using a genetic algorithm. More recently, Vigander demonstrated that a simple evolutionary system could restore most but not all functionality to a 4-bit × 4-bit multiplier that had been subjected to random faults [130]. Complete functionality could be restored by applying a voting system to select between several alternative circuits that had been repaired by evolution. Sinohara et al. used a multiobjective evolutionary algorithm that allowed essential functionality to be restored at the expense of secondary behavior that was not deemed to be important by the designer, such as power dissipation [102]. This algorithm was demonstrated in the repair of NOR gates and inverters. Hounsell and Arlsan have explored the repair of an evolved FIR filter after the injection of multiple faults [38]. They examined two different recovery methods. The first was to recall the final population of the evolutionary run that created the original filter design, and the second was to seed a new random population with a copy of the original design. Both mechanisms recovered functionality faster than rerunning evolution with a completely random population, with population seeding outperforming population recall by a small margin. Zebulum et al. demonstrated evolutionary recovery with a 4-bit DAC that had initially been evolved using traditionally designed operational amplifiers and smaller DACs evolved in earlier experiments as building blocks. Faults were introduced into one of the operational amplifiers. The recovered circuit outperformed the circuit that had initially been evolved. It was suggested that operational amplifiers were not useful building blocks for evolution. Gwaltney and Ferguson investigated fault recovery in an evolved analogue motor controller [26], again by re-evolving the population that gave rise to the best nonfaulty controller after introducing faults. They discovered that evolution could recover from faults in some components better than others, although at least some functionality was restored in all cases.

Louis [70] combined an evolutionary approach with a case-based memory, where partial solutions to similar, previously attempted problems were inserted into the evolutionary population. This process demonstrated that better quality solutions to parity problems could be evolved in less time than when using evolution alone and suggested that this method might prove useful for fault recovery.

Most evolutionary fault recovery systems that have been demonstrated to date have only explored recovery from errors introduced into the logic of the circuit. However, Lohn et al. have demonstrated an evolutionary fault recovery system that can repair routing in addition to logic [66], which they suggest is important for modern routing-rich programmable devices. Another type of fault is the failure of a component at extreme temperatures. Stoica et al. have observed that multipliers, Gaussian curve generators, and logic gates that have evolved under standard conditions degrade or fail at extreme temperatures. However, when re-evolved at those temperatures, the circuits regained functionality in all cases.

Fault detection is traditionally dealt with by incorporating additional hardware into a design to perform a built-in self test (BIST). Garvie and Thompson have demonstrated that evolution can be used to design small adders and multipliers that incorporate BIST at very low additional cost by sharing components

between BIST and the circuit function [17]. Innovative circuit designs such as this will be discussed in Section 2.6.

A number of other bio-inspired online autonomous hardware, fault-tolerance mechanisms have been developed for both fault detection [7] and recovery [72, 126]. Although these have been proposed as a platform for evolutionary experiments, they do not use evolution as an adaptive repair mechanism, and so will not be considered further here.

Fault *tolerance* refers to systems that are inherently tolerant to faults, rather than systems that can detect and/or recover from faults. Evolution has proved an ideal candidate for the exploration of fault-tolerant systems and is discussed under the heading of *generalization* in Section 3.3.

2.6 Design Innovation in Poorly Understood Design Spaces

Traditional circuit designers tend to work on a problem from the top down, decomposing the problem into smaller subproblems that have limited interactions and then repeating the process until only a number of small problems remain that are well understood in the field of circuit design and have known solutions. Each decomposition carefully directs the process towards these solutions by using formal design rules. Evolution works differently. It works from the bottom up, adding components together to make partial solutions to the design problem, which are in turn combined and tinkered with, until the solution meets the design criteria. This idea is discussed more fully in Section 3. For now, we shall discuss when this approach might be useful.

The clearest cases for application are design spaces for which we have very limited knowledge of how components will interact, and so design rules have not yet been developed. Advances in electronic engineering are beginning to generate new kinds of circuit technologies for which the design spaces are often very poorly understood. In these cases, evolution can prove a useful technique in searching for innovative designs, since it can be guided purely by the behavior of the evolving circuit rather than by relying on domain knowledge. An example of this is the field of nanoelectronics, where Thompson and Wasshuber have successfully evolved innovative (but at this stage not particularly useful) single-electron NOR gates [122].

There is also a set of current technologies for which traditional logic synthesis techniques have not yet been designed but that are becoming increasingly important for circuit designers. Many programmable logic technologies provide XOR gates and multiplexers, but digital design rules are best suited to generating sum-of-products solutions that do not map well to these elements. In these cases, an evolutionary approach can work directly with a design abstraction suitable for the technology and potentially search areas of space that a traditional designer would miss if using the techniques above, and this approach may discover more parsimonious solutions, as has been demonstrated by Miller et al. [76].

Beyond these technologies, there are design spaces where the interactions are so complex that it has not been possible to develop formal methods to partition and decompose the design space. For instance, when compared to the design space of digital logic, analogue design is much less well understood. Hence circuit

design in this domain requires more expert knowledge. Evolutionary algorithms have proved very successful in discovering human-competitive (and better) analogue circuit designs [1, 51].

Perhaps the most successful application of evolution to complex design spaces is the automatic design of antennas. Traditional antenna designs are based on a handful of known, regular topologies. Beyond these, the interactions between elements become too complex to abstract. Linden has demonstrated that evolution is capable of discovering an array of highly unconventional, irregular antenna designs [63] and has shown that evolved antennas can be evolved and operate effectively in real-world settings using transmission of real data [61] and transmission where the signal path is obstructed [61]. Such is evolution's performance when applied to antenna design that an evolved antenna is undergoing flight qualification testing for NASA's upcoming Space Technology 5 mission [69], and if successful will be the first evolved hardware in space.

A more subtle and perhaps surprising point is that evolution searches an inherently different area of search space than traditional designers do. Because of this difference, it is possible for evolution to discover innovative solutions even for well-understood design spaces, since some useful circuits lie beyond the areas of solution space we would normally explore if we were to tackle the problem. This outcome, of course, demands that evolution is allowed to work without the design constraints that we would normally place on circuit designs, as was first demonstrated by Thompson. Currently, this approach has not yet yielded any significant real-world applications, but the concept has prompted a great deal of research, as discussed in Section 3.

Current evolutionary techniques only works well for small problems, since the search spaces can become vast for large circuits. A great deal of research is currently directed at scalability, which is discussed later in this chapter. That said, we can still make use of evolution by finding small yet innovative designs that are evolved to produce limited interactions and so can be used by traditional designers as building blocks for larger circuits. Such building blocks have been found for both analogue and digital designs [1, 78]. This approach has also been advocated for use at higher abstractions [101], where it was suggested that evolved or evolvable IP cores could now be provided for commercial use in programmable logic devices. It has also been suggested that previously evolved building blocks may help evolution discover larger circuits [135].

Finally, evolution has proved to be very successful at the generation of circuits that incorporate several functions within one set of shared components, a task for which there is little domain knowledge. We have described an example of this in Section 2.5, where adders and multipliers were evolved to incorporate a BIST function. A related idea is that of polymorphic electronics [108], where a circuit is evolved to perform multiple functions using a shared set of components, with each function becoming apparent under different environmental conditions. For example, a circuit might perform as an AND gate at one temperature and an OR gate at another. Such circuits might prove very useful for military and intelligence purposes.

Design innovation is, in our eyes, the most significant benefit of evolvable hardware; hence, research in this area is discussed in more detail in Section 3.

3 RESEARCH IN EVOLVABLE HARDWARE

Having discussed the benefits of evolvable hardware, and some of the applications that these benefits allow, this section reviews the main thrusts of research in this field. The research is decomposed into three areas: innovation, generalization and evolvability.

3.1 Innovation

Traditional circuit designers decompose a problem from the top down, iteratively splitting the task into smaller and smaller subproblems by applying constraints on their interactions. This partitioning is *actively directed* to reach a set of subproblems contained within the reservoir of electronics and materials knowledge, and is known as *abstraction*. These subproblems can then be individually modeled as an encapsulated physical device, without the need to understand its complex internal interactions. An example is a digital memory device, which can be mapped to an array of analogue circuits that use different techniques to achieve similar input/output characteristics. When the subproblems are reassembled, care must be taken to ensure that the constraints made during the partitioning process are adhered to. For example the digital memory device mentioned above is often constructed of high-gain analogue components, so we must ensure that its output is allowed to saturate before it is passed to another part of the circuit.

Evolution uses a different approach. It works from the bottom up, attempting to find correlations between sets of components that consistently improve the behavior of a circuit with respect to the problem at hand. Unlike traditional design, the direction of its search does not have to be directed by previous knowledge. If evolution is set up in such a way that it can *exploit* correlations between the components it manipulates and the observed external behavior of a circuit, then circuit designs can be discovered using this behavior alone as a guide, regardless of the complexities of the interactions within the circuit.

In Section 2 we discussed four areas of application for innovative evolutionary design: familiar design technologies with relaxed abstractions, programmable logic abstractions, complex design technologies, and new design technologies. These are now discussed in turn.

3.1.1 Relaxing Abstractions

Seminal work on the relaxation of design abstractions was carried out by Thompson. He first set out to show that evolution could successfully manipulate the dynamics and structure of circuits when the dynamical and structural constraints that traditional designers depend on heavily had been relaxed. He demonstrated this [120] by evolving a complex recurrent network of high-speed gates at a netlist level abstraction to behave as a low-frequency oscillator. Fitness was measured as an average error based on the sum of the differences between desired and measured transition periods. Circuits were evaluated in simulation using an asynchronous digital abstraction. Hence a search space containing only circuits

that used behavior modeled by the simulator was searched, with the space strictly partitioned into units of logic gates. However, as the simulator allowed the gates to interact asynchronously, the selection operator could explore the asynchronous dynamics of the model, being free to make use of any such behavior or ignore it as it saw fit.

The required behavior of the circuit was successfully evolved, showing that it is possible for evolution to search without the constraints (in this case, synchronous constraints) usually needed by traditional designers. Further, a graph-partitioning algorithm showed that the structure of the circuit contained no significant structural modules, as would be seen through the successive abstraction approach of a traditional top-down approach. Thompson also showed that the circuit behavior relied on methods that would not have been used by traditional designers. So not only had evolution found a solution by searching the space beyond conventional circuit design space but also it had found a solution that that actually *lay within* this space.

Thompson went on to show that evolution with relaxed restrictions on circuit dynamics was possible in physical hardware, rather than simulation [120]. The hardware was a finite-state machine for a robot controller. However, whether the states were controlled synchronously by a given clock or not was under genetic control, an architecture Thompson termed a *dynamic state machine (DSM)*. The evolved robot controller used a mixture of synchronous and asynchronous behavior and interacted with the environment in a complex dynamical manner to produce behavior that would not have been possible using the finite-state machine abstraction with such limited resources. Importantly, he suggested that the ability of such a parsimonious controller to interact in such a complex manner with its environment was not attributable to the DSM architecture. Rather, it arose from the ability of evolution to *exploit* it. Again, evolution had found a circuit that traditional design techniques could not generate by avoiding a traditional design constraint, which in this case was the synchrony imposed on the finite-state machine abstraction. But in addition, evolution had found a circuit that used the rich dynamics that can arise by relaxing design constraints to perform a real task, demonstrating that such dynamics can give rise to *useful behavior* in the real world.

Thompson also carried out the first intrinsic evolution of a circuit evaluated on an FPGA. A 10×10 area of a Xilinx XC6126 bitstream was evolved. Almost all bits in the bitstream corresponding to this area were evolved directly as the bits of the chromosome of a genetic algorithm [116]. Thereby Thompson set about evolving a circuit at the lowest level of abstraction possible with the device he had—that of the *physical behavior* of the target technology. The task was to evolve a circuit to discriminate between 1 kHz and 10 kHz signals. Fitness was calculated by subjecting each circuit to five 500 ms bursts of each signal in a random order, and awarding high fitness to circuits with a large difference between the average voltage of the output during these bursts. The average voltages were measured with an analogue integrator. The only input to the circuit was the 1 kHz/10 kHz signal—no clock was given, and hence the task required that a continuous-time arrangement of components be found that discriminated between signals many orders of magnitude longer than the delay afforded by each individual component. The resulting circuit used a fraction of the resources that a

traditional designer would need to achieve the same task. Following months of analysis, Thompson and Layzell described the functionality of the circuit as "bizarre," and to date, the nature of some of the mechanisms it uses are still not completely understood, although the authors postulated that the circuit made use of the *underlying physics of the substrate* in a way that traditional design would consider too complex to consider.

Later, Thompson and Layzell carried out a similar experiment, this time providing the circuit with a 6 MHz oscillator signal that could be used or ignored as evolution required [121]. The prime motivation for the experiment was to investigate robustness, and so evaluation was carried out under a range of conditions specified by an operational envelope. Hence the constraints to the system were the same as before, except that a soft bias towards robust behavior had been added through the fitness function. However, an additional dynamical resource had been provided. The resulting circuit made use of the clock, and the design was simulated by using the PSpice digital simulator. The simulated design behaved exactly like that of the real circuit, showing that evolution had found a solution within the digital design abstraction of the simulator, even through the constraints did not explicitly require that. However, analysis of the simulation waveforms showed a large number of transient signals. This finding allows us to conclude that potentially useful circuits lie within the digital abstraction that are *undiscoverable* using traditional digital design methodologies owing to their greedy, top-down nature, and that at least some of these circuits *can* be discovered using evolution.

3.1.2 Programmable Logic Abstractions

In Section 2 we noted that most digital circuit design methodologies are geared towards producing logic in the canonical sum-of-products form. However, many programmable logic devices support additional components that are not easily utilized by such an abstraction, such as XOR gates, multiplexers, and lookup tables (LUTs). Miller et al. have conducted research into the discovery of innovative circuits, one of their main motivations being the derivation of new design principles that could be applied to logic abstractions such as those found in programmable logic devices. They note [78] that Boolean or other algebraic rules can map from a truth table of required circuit behavior to an expression in terms of that algebra. They then suggest that a bottom-up evolutionary approach could search not only the class of expressions that the algebraic rules map to but also a larger space of logical representations beyond commonly used algebras.

In an attempt to demonstrate this idea, they successfully evolved one- and two-bit adders based on the ripple adder principle using a feed-forward netlist representation of AND, OR, NOT, XOR and MUX gates. This space lies beyond the commonly used Boolean and Reed–Muller algebra spaces but is of interest since the multiplexer is available as a basic unit in many technologies. This argument is very similar to Thompson's in principle—that the discovery of innovative circuits can be facilitated through the modification of design abstractions implemented through representational biases.

Many of the circuits reported in this and other work [83, 76] were unusual but interesting because of their efficiency in terms of gate count. They lay in the space

of circuits making use of multiplexers and XOR gates, outside the space of traditional atomic Boolean logic units. The authors argued that these circuits were unlikely to be found using traditional algebraic methods, and so evolutionary "assemble-and-test" was a useful way that such a space can be explored. The work continued with the evolution of two-bit and three-bit multipliers. All work was carried out using gate-level logic simulation. Similar work has been carried out with multiple valued algebras [46].

Another aspect of this group's work is the contention that *design principles* useful to traditional designers could be discovered by searching for patterns in evolved circuits. In particular, they hypothesis that by evolving a series of modules of increasing size, design principles that the modules have in common may be extracted from them. The authors [78], [83] evolved many one and two bit adders, and by inspection deduced the principle of the ripple adder. Although knowledge of this principle already exists in the domain, they went on to argue that evolution discovered and made use of it with no prior knowledge or explicit bias. Since the design principle could be extracted a comparison of one- and two-bit adders that had evolved to use the principle, they asserted that evolution could be used as a method of design principle discovery.

More recent work in this area has concentrated on developing an automatic method of principle detection [76, 77]. Having successfully evolved two- and three-bit multipliers that are much more compact than those of traditional design, the authors have integrated a data mining procedure to search for design principles [43]. The learning algorithm used for the data mining process is an instance-based learning technique called Case Based Reasoning [84], (Chapter 8). We shall argue in our discussion on scalability in Section 3.4 that by modeling biological development we might be able to allow evolution to automatically encapsulate such design principles without the need to resort to other learning techniques and to use evolution itself to select for design principles that are inherently evolvable.

3.1.3 Complex Design Technologies

In Section 2, we noted that there are complex design spaces for which it has not been possible to develop formal methods to partition and decompose the design space, and that evolutionary algorithms offer an alternative approach to the use of a human expert. An example of this kind of design space is that of analogue circuit design.

One traditional technique of simplifying an analogue design space is to fix the topology of the circuit to a design with well-known characteristics and to modify only parameters relating to the components within the design. A good deal of work using evolutionary algorithms in analogue circuit design takes this approach, and can be considered to have more in common with evolutionary optimization than evolutionary circuit design [3]; [88]. However, as the field of evolvable hardware has developed, researchers have begun to allow evolution to explore analogue circuit topologies. For instance, Grimbleby developed a hybrid genetic algorithm/numerical search method that used the genetic algorithm to search topologies and a numerical design optimization method to select parameter values for the evolved topologies [23]. Additionally, Koza et al. and Lohn and

Columbano have both developed evolutionary circuit design methods that explore both topology and component parameters [67]; [50]. These two methods are of particular interest to us since they do not use a fixed mapping of genotype to phenotype. The benefits of using such an approach, and details of these two examples in particular, are discussed at length in Section 3.4.

With the advantages of evolutionary design in mind, Gallagher has recently advocated a return to the development of analogue computers [16], which today have been almost completely replaced by their digital counterparts. He distinguished two classes of analogue computers. The first is *direct* computers, which are designed to reproduce the behavior of a physical system directly. The example he gave was of a serial RLC circuit. This can be considered as directly modeling a damped harmonic oscillator, where inductance is equivalent to mass, capacitance is equivalent to the inverse of spring elasticity, and resistance is equivalent to frictional damping. Indirect analogue computers simply implement complex mathematical functions using building blocks that embody simple mathematical functions, such as adders and integrators. Gallagher suggests that the demise of the analogue computer is mostly due to a combination of the difficulty in discovering direct implementations of required computations and the difficulty in constructing accurate indirect models due to compound errors in component precision. He went on to point out that intrinsic evolution actually discovers direct implementations, since the circuit is designed purely to replicate a specified behavior rather than to perform a mathematical function, and that for applications where size and power are vital, evolving direct analogue models should be considered as a serious alternative to digital models of analogue computations.

An impressive example of evolution's ability to manipulate interactions that are too complex for human designers to fathom is that of antenna design. We have already mentioned in Section 2 that evolution is capable of discovering an array of highly unconventional, irregular antenna designs. Early work in this field used simulation; however, Linden [60] went a step further. He intrinsically evolved an array of wires connected with reed switches, which are mechanical switches that are closed by an induced magnetic field, controllable from a computer. The antennas that he evolved made use of the complex electromechanical coupling between wire segments that resulted from the fields of the reed switches. Human designers would be unable to exploit such complex nonlinear physical interactions in a controlled manner.

3.1.4 New technologies

In Section 2, we briefly discussed that evolutionary design is likely to be a useful tool for new circuit design technologies for which no domain knowledge exists. Thompson [119] suggested that until a model of a new technology is derived, only a blind search technique such as evolution can be of use to design circuits in it. He first noted that as we move towards nanoscale circuitry, we cannot continue to suppress quantum effects so that our macroscopic models fit; rather, we must make use of them. He then described a system of this third class. The system consisted of an array of quantum dots between which electrons could only pass by quantum mechanical tunnelling. The task was to evolve a NOR gate by

modifying effectively only the size, shape, and position of the dots. Thus, evolved circuits would rely on tunnelling effects to perform a logical function. (The task was carried out in simulation, but the concept is unaffected.) The evolved circuit used a property called *stochastic resonance* in which the thermal energy of the electrons allows stochastic transmission of a signal. This is an innovative property never before considered for the design of electronic circuits, be they single-electron or not. That evolution discovered this property demonstrates its ability to blindly design in the absence of any useful design rules.

There are also hopes to exploit quantum effects in another way: through quantum computing. Quantum computers do not process bits. Instead, they process qbits, which exist in a superposition of states. This allows n coupled qubits to represent a superposition of 2^n states, and operators acting upon the qubits operate on the superposition of states in parallel. This means that, as the number of superposed bits the operators operate upon increases, the processing power of the device increases exponentially with respect to traditional computing devices. Once quantum circuits are developed that can operate on superpositions of even tens of bits, they are likely to have enormous computing power. Theory has pointed to a number of rudimentary quantum gates that could be used to develop quantum circuits, although practice suggests that the number of interconnected gates is likely to become a limiting factor in their design. This realization has led a number of researchers to begin searching for innovative parsimonious sets of quantum gates using evolutionary algorithms [71]; [111].

Several researchers have also suggested that the field should be designing new technologies to suit evolutionary algorithms rather than the reverse. Miller and Downing have noted that all of today's electronic components have been designed specifically for top-down design methodologies and that researchers in hardware evolution have been "abusing" these components [75]. They argue that biological evolution is clearly capable of evolving extremely complex structure by *exploiting the physics* of the surrounding environment, and so we should be looking for substrates that exhibit rich, complex internal interactions and must be reconfigurable, ideally by small applied voltages. They suggest that substances that exist in a state on the edge of disorder would be good candidates, as they would exhibit the rich interactions necessary while being able to quickly relax to a homogeneous quiescent state. The Candidates they have suggested include liquid crystals, electroactive polymers, and voltage-controlled colloids.

Amorphous computers have also recently been suggested as a substrate amenable to evolution. Amorphous computers are essentially large collection of simple, wireless units that perform computations. These units are unreliable, not geometrically aligned, and can only communicate locally, but they are likely to be relatively easy to synthesize in extremely large arrays, as compared with other future technologies. However, no computational paradigm exists that can take advantage of their massively distributed function. Future nanoscale devices are also likely to have an amorphous structure, as Miller and Downing have pointed out [75]; hence, this could be a major issue for upcoming computational devices. Haddow and van Remortel have suggested that, by combining the principles of biological development and evolvable hardware, it may be possible to realize designs for amorphous computers [28].

3.2 Generalization

In the section above, we have discussed what we believe to be the primary motivation for work on evolvable hardware, namely, its ability to create innovative hardware. In this and the next section, we discuss the two greatest hurdles to evolvable hardware's viability for general real-world applications. The first of these is the difficulty of generalization.

Inductive learners such as evolutionary algorithms infer hypotheses from observed training examples of some kind. In the case of evolvable hardware, we test prospective circuits by exposing them to different conditions, most commonly a range of input signals, and observing the circuit outputs in order to evaluate fitness. If it is infeasible for all possible training examples to be observed by the learner, then the learner generalizes beyond the cases it has observed. Modern real-world circuits can process hundreds of input signals, and to observe each possible combination of these just once, even at millions of training cases a second, would take longer than the age of the universe. For sequential circuits, the number of training cases is infinite. And as we shall see later in this section, unseen signal inputs are but one (admittedly important) example of unseen operating conditions that we might hope a circuit to generalize across. Clearly, the ability to generalize is vital to the long-term future of evolvable hardware.

Two approaches to applying bias towards generalization can be found in the literature:

1. Introduce domain knowledge about the *structure* of circuits that exhibit the required generalization characteristics, perhaps in the form of a heuristic.
2. Introduce knowledge about the *behavior* of circuits that exhibit the required generalization characteristics, and rely on evolution to learn about the structure of circuits that exhibit the required behavior in addition to the primary task.

We now explore work on generalization, first by considering the special case of generalization across unseen input signal cases.

3.2.1 Generalization Across Input Vectors

Several researchers have explored input generalization under the framework of pattern recognition, a familiar problem in the area of generalization and therefore well suited to the study of this problem. As we mentioned in Section 2, many systems have been developed that demonstrate that evolvable hardware can generalize to unseen test cases for real-world pattern recognition data, such as image and signal classification [34]; [90] and image and signal noise filtering [100]; [131]. Yao and Higuchi have implied that the success of evolvable hardware in problems like these relies in some way on the use of a hard bias towards feed-forward networks of nonlinear processing units, likening their function to ANNs [132]. This bias is an example of case 1 above. Iwata et al. successfully managed to improve upon the generalization abilities of this kind of system by applying additional knowledge, again in the style of case 1 above [42]. They introduced a heuristic commonly used in the machine learning literature to improve generalization. The heuristic emerges from the application of the Minimum Description Length (MDL) principle to the discovery of maximum a posteriori hypotheses in Bayesian settings, and

biases the search towards small circuits. For details of this interpretation of MDL, see Mitchell, [84] Chapter 6).

Miller and Thomson investigated the generalization abilities of a system evolving two- and three-bit multipliers with respect to the size of the input training sets [80, 81] and were far less successful. The task was to evolve a functional circuit from a subset of the truth table. They found that if evolution was presented with a subset of training cases throughout the entire evolutionary run, it was not able to produce general solutions. This finding suggests that in the setting of this problem and algorithm there was no implicit bias towards generality, even though they again enforced a hard representational bias towards feed-forward networks. They also reported that even when evolution was provided with a new set of training cases randomly drawn from the truth table every generation, general solutions were still not found, suggesting that evolution had little memory in the context of this problem.

Miller and Thomson also investigated the evolution of square root functions [80, 81]. In these cases, they discovered that some acceptable solutions were generated when evolution was limited to an incomplete training set. These cases occurred when the missing training cases tested low-order bits, which contributed less to the fitness. This outcome seems to answer the puzzle as to why their earlier experiments failed to generalize, as we shall now explain with reference to another experiment.

Imamura, Foster, and Krings also considered generalization in Miller's multiplier problems [40] and concurred that evolving fully correct circuits to many problems was extremely difficult without access to a full training set. They pointed out that the problem was exacerbated in functions where each test vector contained equal amounts of information relevant to the problem, such as the case of the three-bit multiplier studied by Miller and Thomson. However they suggested that in cases where the data contained a large amount of "don't care" values, evolvable hardware could be successful using a smaller test vector. Real-world pattern classification data contain redundant information, which explains why they succeeded where the multiplier problem failed. Indeed, since many input sets exhibit this property, it seems reasonable to assume that for any real-world problem some level of redundancy is likely to exist, although the problem of how to select test vectors remains. Immamura, Foster, and Krings suggested an adaptive approach of allowing the evolving system to search for useful subsets of test vectors.

3.2.2 Generalizing Across Operating Environments Though Representation

Just as it is unrealistic for the algorithm to train from every conceivable circuit input, in most cases it is unrealistic to train under every conceivable operating environment. Operating environments might include a range of technologies or platforms on which the designed circuit should operate, as well as a range of conditions to which the embodied circuit may be subjected.

Traditional designers usually manage such generalization by imposing hard biases on the nature of the circuit. These biases are again representational abstractions that encode domain knowledge known to produce behavior common across all necessary operating environments. The abstractions are then mirrored

on the physical hardware through some constraint on the hardware's behaviour. A circuit that behaves correctly in all necessary conditions should then follow. For example, a gate-level digital design abstraction requires that the physical gates of the target technology behave as perfect logic operators. In most technologies, these gates are represented by transistors—physical devices that behave like high-gain amplifiers. Timing constraints and operating environment constraints specified by the manufacturer of the physical device are imposed on the real hardware. This ensures that, when an abstract computation takes place, the voltages of the transistors within each logic gate have reached saturation, and any transient behavior generated before saturation has dissipated. From this point forward, the outputs can be treated as logical values. In synchronous systems, these constraints are usually imposed with respect to a clock. The manufacturer will then guarantee that for a range of operating conditions, the device will behave as it appeared to within the design abstraction. The design is then portable across a range of devices and operating conditions.

Evolutionary circuit design often takes a similar approach to the traditional design process by applying design abstractions used by traditional designers. Many circuits have been evolved at levels of abstractions that would limit the search to circuits with good generalization characteristics. However, the only case we are familiar with where representational design abstractions have been imposed *specifically* to ensure good generalization is that of Stoica and colleagues [109], where a very high level of generalization was required. The experiment involved evolving transistor level circuits, and a representational bias was imposed that prevented input signals from connecting to transistor gates rather than to source or drain inputs, thus improving the loading characteristics of the evolved circuits. (The experiment is discussed in more detail in Section 3.3.3.)

3.2.3 Generalization Across Operating Environments by Inference from Examples

In cases where no knowledge is available about the structure of solutions that generalize across all operating environments, the only solution is for evolution to infer this information from examples.

Early work with intrinsically evolved circuits by Thompson focused on design innovation through relaxation of constraints [115, 116, 117]. Thompson successfully evolved a circuit to distinguish between two frequencies, using a Xilinx XC6200 FPGA. However, he then went on to note the lack of robustness to environmental conditions such as temperature, electronic surroundings, and power supply that may occur. He also noted that the design was not portable when moved not only to a different FPGA, but also to a different area of the same FPGA. Similar results have been reported by Masner et al. [73]. Thompson went on to explore how solutions that generalized well across a range of operating environments could be evolved [18]. He took a previously evolved FPGA circuit that discriminated between two tones. He then specified a number of parameters for an operational envelope which, when varied, affected the performance of this circuit: temperature, power supply, fabrication variations, packaging, electronic surroundings, output load, and circuit position on the FPGA. The final population from the previous experiment was then allowed to evolve further, this time on

five different FPGAs maintained at the limits of environmental conditions specified by the operational envelope parameters. Although there was no guarantee that the circuit would generalize to behave robustly under all environmental conditions within the envelope, Thompson found a level of robustness evolved in four out of five cases. Hence, it appears that the biases he had introduced into the evolutionary algorithm were sufficient to promote good operating-condition generalization characteristics for the evolution of the 6200 architecture.

In a similar vein, Stoica et al. [106] explored the operation of circuits in extreme temperatures. Their initial experiment involved testing both traditionally designed circuits and circuits evolved under standard conditions (multipliers, Gaussian curve generators, and logic gates) to see whether they degrade or fail at extreme temperatures. This was primarily an experiment in evolutionary fault recovery, and they demonstrated that all circuits could regain functionality when evolved under extreme conditions. However, it is interesting to note that a population of 50 circuits re-evolved for 200 generations in this manner often exhibited degraded performance under standard conditions, whereas before they had functioned perfectly. This finding suggests that generalization qualities are easily lost if a consistent bias towards them is not asserted during evolution.

A problem closely related to Thompson's exploration of portability is the portability of extrinsically evolved analogue circuits to physical devices. Analogue circuit simulators tend to simulate circuit behavior very closely, and so it might be expected that extrinsically evolved circuits would generalize well to the real circuit. However, this does not happen in practice. One issue is that some behaviors that simulate according to the physics programmed into the simulator may not be feasible in the chosen implementation technology. A common example is that simulators fail to prevent the simulation of extremely high currents, and so evolution is free to take advantage of them in its design. Koza et al. have evolved many circuits extrinsically at an analogue abstraction using the Berkeley SPICE simulator [50], but have found that these circuits are practically infeasible because they rely on extremely high currents. Additionally, analogue simulators use very precise operating conditions. The circuits of Koza et al. are evolved to operate at 27°C, and so there is no explicit bias towards generalization across a range of temperatures.

When evolving networks of transistors intrinsically, Stoica et al. have come across the reverse problem: circuits evolved intrinsically may operate as expected under the conditions prevailing when they were evolved, but may not operate acceptably in software [105]. Their solution to the problem was to evaluate some circuits of each generation intrinsically, and some extrinsically. This they termed *mixtrinsic evolution* [107]. They also suggested that another use of mixtrinsic evolution would be to reward solutions that operate differently in simulation than when instantiated in a physical circuit. This would encourage innovative behavior not captured by simulation. They later developed a method [25] to include several different software models, based on various different processes, analysis tests, and timing resolutions.

The issues of portability discussed above have only dealt with portability between simulation and PLDs. An issue of extreme importance for evolutionary circuit design is whether designs evolved either extrinsically on PLDs or intrinsically are portable to custom application-specific integrated circuits (ASICs),

which cannot be used during mixtrinsic evolution. Until recently, this question had been left unanswered, but Stoica et. al [109] evolved transistor-level gates using a combination of comprehensive fitness testing on each individual and mixtrinsic testing across the population. Comprehensive tests included transient analyses at different frequencies, testing a number of loads. Mixtrinsic tests were SPICE analysis on several process models and a range of voltages and temperatures. Additionally, a representational bias was imposed to improve loading characteristics, as mentioned in Section 3.3.2. Tests that were carried out mixtrinsically during evolution were carried out in full on the final evolved solutions, and revealed that some but not all of the circuits performed robustly across all tests. All circuits exposed to the full range of validation were successfully validated in silicon, showing that with careful validation procedures, portability of evolved designs to ASIC technologies is possible.

The concept of the ability of circuits to function under various environmental conditions can be extended to include the capacity of circuits to operate in the presence of faults. This was first investigated by Thompson [115, 116, 117]. He evolved a DSM-based robot controller problem (discussed in Section 3.2.1) in the presence of single-stuck-at (SSA) faults in the RAM used to hold a lookup table of state transitions for the state machine. Rather than testing each candidate solution exhaustively across all sets of possible faults, he aimed to test only the fault that caused the most degradation in each controller. He recognized that the population was likely to be made up of individuals of various designs, and hence the highest degradation of performance was unlikely to be caused by the same fault in the RAM. To circumvent this problem, at each generation he averaged the RAM bits across the DSMs of the entire population to give what he termed a *consensus individual*. Faults were only introduced once a good solution was found, and then the population was tracked to see how it performed. He found that solutions that were tolerant to most SSA faults existing in the initial population of evolved solutions, for reasons discussed in Section 3.3.4, but as evolution proceeded in the presence of faults, tolerance was lost as the algorithm concentrated on tolerating the single worst fault until eventually solutions tolerant to any single fault were discovered.

Canham and Tyrell extended this work to more complex faults that commonly develop in FPGA architectures [8]. They emulated a Xilinx 6200 series architecture on a Xilinx Virtex FPGA and introduced simulated SSA faults in the logic of the configurable logic blocks (CLBs), and short circuit faults between the inputs and outputs of the CLBs during evolution. The resultant circuits were compared against a set of control circuits that were evolved in the absence of faults and found a large increase in fault tolerance that could not be explained purely by "junk" faults occurring in unused areas of the FPGA.

Hartmann et al. have evolved fault-tolerant circuits using nonperfect digital gates called messy gates [30]. Various levels of noise were injected into digital gate models simulated using SPICE, and digital circuits were evolved. The circuits are manipulated by evolution at the gate level, but the evaluation of circuits was carried out using SPICE. The authors discovered that adders and multipliers could be evolved under high levels of noise. They postulated that the noise smoothed the fitness landscape as highly fit circuits that depended on each gate to perform function were no longer present in the search space.

3.2.4 Inherent Generalization

Another fascinating model for fault tolerance is that the biases of the evolutionary algorithm have an inherent tendency to generate solutions that generalize across certain conditions. Thereby, evolved circuits would exhibit robustness to changes in those particular conditions "for free."

Thompson has also postulated that evolved circuits may be inherently robust to some types of fault. He observed that an evolutionary algorithm will by nature be drawn to optima surrounded by areas of high fitness, and suggested that as a result, a single bit mutation from such an optimum will also tend to also have a high fitness. He then conducted experiments on an artificial NK landscape to demonstrate this. For details of this type of landscape, see work by Kauffman and Levin [47]. He then proposed that such an effect could have beneficial engineering consequences if a mutation were to cause a change in the circuit that is similar to a fault—namely, that the evolved system is likely to be inherently robust to such faults. He went on to highlight this by using the evolution of the DSM robot controller described in Section 3.3.3 as an example. Each bit of the RAM that encoded the controller's state machine was directly encoded in the chromosome, and so mutation of one of these bits had a effect similar to a "single stuck at" (SSA) fault. Examination of the effect of SSA faults on a previously evolved state machine revealed that it was quite robust to faults. However, since state machines for this problem with similar fitness could not be easily generated by any means other than evolution, statistical tests of the evolved machine's resilience to faults could not be carried out.

Following this experiment, Masner et al. [73] carried out studies of the effect of representational bias on the robustness of evolved sorting networks to a range of faults. The aim of the work was to explore the relationship between size and robustness of sorting networks using two representations—tree and linear. They noted that robustness first increases and then decreases with size, and is therefore not due purely to the existence of redundant nonfunctional gates in the sorting networks. They also noted that the linear representation tended to decrease in robustness with respect to size faster than the tree representation.

Layzell has suggested that robustness of solutions can also be generated at the level of populations [55]. In particular, he was interested in the ability of another member of the population to be robust with respect to a fault that causes the original best solution to fail. This outcome he called *populational fault tolerance (PFT)*. He went on to demonstrate that PFT is inherent in certain classes of evolved circuit and to test various hypotheses that could explain its nature. As with Masner et al., he noted that fault tolerance did not seem to be a result of redundant units based on the current design. Instead, he showed that descendants of a previously best and inherently different design were still present in redundant genes in the members of the population. It was these individuals that provided PFT. He demonstrated that this situation did not result from the presence of a diverse range of distinct solutions in the final population when he repeated the experiment using a single hillclimber to evolve solutions and then generated 50 single-bit mutants of this single individual. These individuals presented a similar tolerance to fault, confirming that the fault tolerance was inherent to the *incremental* nature of evolutionary processes in general: the entire

population contained remnants of inherently different solutions that had been explored earlier.

This fact suggests that PFT is somewhat of a misnomer, since one might expect it to refer to tolerance owing to the nature of a population-based search. Tyrrell et al. have explored what might be called "true" populational fault tolerance [127]. Unlike Layzell's work, population diversity was encouraged by evolving oscillators using a population of 16 hillclimbers that did not interact with each other. This setup ensured that the evolved solutions did not share a common evolutionary history, so any fault tolerance observed could not be a result of the effect proposed by Layzell above. When faults were introduced to the oscillators that caused the best member of the population to fail, another member of the population often retained relatively high fitness. This demonstrates that population diversity can also play a role in evolved fault tolerance.

3.3 Performance and Evolvability

A good deal of research in the field of evolvable hardware is devoted to the following:

- improving the quality of solutions that evolution discovers for a given problem
- improving the scalability of evolution to larger and/or more complex problems
- improving the speed with which evolution finds acceptable solutions

These ideas are highly interrelated since they all aim to improve the performance of the evolutionary search in order to achieve slightly different goals.

3.3.1 Representations

Selection of a good representation is crucial to the performance of an evolutionary algorithm. As discussed in Section 2, the representation of an evolutionary algorithm defines how solution space is mapped onto search space. This process affects the performance of the algorithm as it delimits the solutions present in the search space, thereby fixing the density of acceptable solutions in the search space. Many researchers, particularly in the early days of evolvable hardware, believed that performance could be improved by reducing the size of the search space and increasing the density of good solutions lying within it. This approach will be discussed in due course. However, representation has a second effect. In Section 2 we discussed how it partly specifies the order of traversal of search space, since it sets the distance between any given points in space. Hence, it changes the *nature* of the search space. It is becoming increasingly recognized that having a small-sized space is not as important as having a space that allows evolution to discover incremental improvements in fitness that will lead it to a solution [10, 2, 115]. We define a search space that allows this process to occur an *evolvable* search space.

Miller and Thomson have explored how changes in circuit geometry affect the evolvability of a two-bit multiplier [79, 80, 81] and how the functionality-to-routing ratio affects the evolvability of netlist representations of the SBOX

problem space [79, 80, 81]. It appears that evolvability is affected profoundly but erratically by both factors, making it difficult to draw many direct conclusions. Miller and Thomson did note, however, that evolvability was improved by allowing cells dedicated to routing signals between functional cells. However, because these studies may be dependent on the problem, the biases imposed by the specific operators used within the algorithm, and the level of abstraction at which the experiments were conducted, again it is dangerous to read too much into this work.

3.3.2 Function Level Evolution

The function-level approach to improving evolvability was proposed by Murakawa et al. [89] and has since been adopted by many others [124, 99, 123]. Murakawa et al. pointed out that the size of the search space for a binary genetic algorithm increases at a rate of 2^n for every addition n genes, and suggested that as evolution tackles larger problems, the explosion in search-space size prevents the algorithm from searching effectively. One solution they proposed was function-level evolution. Here they suggested that instead of using gate-level representations, domain knowledge could be used to select high-level computational units, such as adders, subtractors, and sine generators, that could be represented directly in the chromosome. thereby reducing the size of the chromosome necessary to represent an acceptable solution. Although this approach has proved to be successful for limited problems, there are several issues that indicate it is not a long-term solution. First is the problem of domain knowledge, which requires an experienced designer to select suitable function-level units for the problem at hand. Furthermore, if little or no domain knowledge exists for the problem, it may not be suitable for a function-level approach. Second, the approach is not scalable to problems of increasingly greater complexity without introducing more domain knowledge through the selection of more powerful functions. Third, once an abstraction has been made through the selection of function-level units, evolution will be limited to search the space of this abstraction, and any innovative solutions at a lower abstraction will be unattainable. Finally, and perhaps most importantly, the functional units are selected using domain knowledge from traditional design processes. As we have discussed throughout this chapter, evolution performs a bottom-up search rather than a top-down design. In Section 3.4.1, we pointed out that there is very little domain knowledge about the *evolvability* of circuit design spaces, and so even functions selected by experienced designers may not be of value when attempting to solve a problem using an evolutionary algorithm.

Indeed, Thompson argued that coarse-grained representations such as those employed by function-level evolution may reduce the evolvability of a hardware design space [115, 116, 117], since the addition to or removal from a circuit design of a complex function is likely to have a more dramatic effect on the overall function of the circuit than simple function. Thompson makes a strong argument that traditional evolution has the capability to search larger spaces than those advocated by Murakawa et al. [89]. In particular, he suggests that there may be features of many hardware design landscapes that allow us to search large spaces beyond

the point where the evolving population has converged in fitness. Such a feature, he suggested, was the neutral network.

3.3.3 Neutral Networks

Neutral networks can be conceived as collections of genotypes with phenotypes of identical fitness that are arranged in search space so as to make pathways or networks that can be navigated by evolution through the application of its genetic operators. It has been suggested that *genetic drift* along such networks can allow evolution to escape local optima that they would otherwise be anchored to [39]. The idea of neutral mutations has been recognized in the field of evolutionary biology for some time but has only in recent years been used as a paradigm for search in evolutionary computation. Harvey suggested that taking full advantage of neutral networks would require a redesign of evolutionary algorithms, and in light of this he proposed the Species Adaptation Genetic Algorithm (SAGA) [31], which advocates incremental changes in genotype length and a much greater emphasis on mutation than is common for genetic algorithms. Thompson, however, managed to prove his point using only a fixed-length genetic algorithm with a SAGA-style mutation rate to search an incredibly large circuit design space (2^{1800}) for good solutions. This he succeeded in doing, and when the algorithm was stopped owing to time constraints, fitness was still increasing even though the population had converged long before [32]. Analysis of the evolutionary process did indeed reveal that a converged population had drifted along neutral networks to more fruitful areas of the search space. He attributed much of this behavior to the increased mutation rate, a change to the static procedural mapping of the algorithm. He went on to speculate that neutral networks might be a feature of a great deal of design spaces, including many hardware design spaces.

Vassiliev and Miller have explored neutrality in the three-bit multiplier logic netlist space. Their work [128, 129] suggests that neutral changes at the start of an evolutionary run occur because of high redundancy in the genotype. As the run continues and fitness becomes higher, redundancy is reduced. However, the number of neutral changes does not drop as quickly, suggesting that selection *promotes* neutral changes in order to search the design space. They then went on to show that when neutral mutations were forbidden, the evolvability of the landscape was reduced. They have also proposed that the search for innovation may be assisted by using current designs as a starting point for evolution, and proposed that a neutral bridge could be used to lead from conventional design space to areas beyond [128, 129].

Much of the work on neutrality uses evolutionary strategies as opposed to the more traditional genetic algorithm. Evolutionary strategies do not use the crossover operator, and because of this, their use in studies of neutral mutations, the driving force of evolution in the neutral network paradigm, simplifies analysis.

3.3.4 Incremental Learning

Following the function-level approach, Torresen proposed another idea based on evolving more complex components to improve scalability. Inspired by results

from the use of automatically defined functions in genetic programming, and recognizing that an incremental, bottom-up process might improve scalability, he suggested that evolution could be handed the task of evolving higher-level functions. He also suggested that the process could be repeated incrementally so as to produce a complex solution based on a series of modules that had been iteratively encapsulated into larger ones. Thus he dubbed the approach *increased complexity evolution*. However, he still needed a mechanism to modularize the problem into less complex subtasks that would each present a more evolvable landscape than that of the entire task.

He suggested that the complexity of the problem could be subdivided by a traditional functional decomposition, and demonstrated the process with a pattern recognition task where a number of character images were to be classified according to character. Each output of the circuit corresponded to an individual character and was to be set high only if the pattern under test corresponded to that character. He manually decomposed the problem into a set of circuits where each would be evolved to detect only a single character. His results showed that there was a significant increase in evolutionary performance when decomposing the problem in this way. Unfortunately, his demonstration implicitly included domain knowledge by applying the idea of top-down decomposition to a problem that is amenable to such an approach. Additionally, he also stopped short of demonstrating the benefits such an approach could bring to scalability, since he did not present a demonstration of evolution at a higher level of abstraction using the evolved circuits as primitives. Finally, the opportunity for an incrementally evolved system to innovate is curtailed by this approach, in this case by the imposition of a traditional top-down design that was implicitly imposed. Although this method does not fundamentally solve the problem of scalability it may be useful when knowledge is available as to how a problem might be decomposed. For example, Hounsell and Arslan [37] decomposed a three-bit multiplier problem by output pins in this manner. In this case, they automatically integrated the individual circuits, which were evolved extrinsically, using standard logic minimization techniques, thereby automating the technique and addressing to some extent the issue of parsimony that Torresen had not touched upon. Kazadi et al. [48] have extended the idea further by removing the requirement of combining evolved circuits using traditional minimization techniques, thereby increasing the opportunities for innovative circuit design. They achieved this by first evolving the correct behavior for a single output and then selecting a single parsimonious solution and encapsulating it as a module. The module was then used as a primitive for another stage of evolution in which correct behavior for an additional output was required. The process was iterated until correct behavior for all outputs was observed. Although this method can automate the generation of a complete circuit, it is still relies on decomposition by output producing evolvable subproblems.

Lohn et al. have compared a number of incremental-type systems. They compared three dynamic fitness functions against a static one [68]. The dynamic fitness functions increased in difficulty during an evolutionary run. One had a fixed increase in difficulty, based on domain knowledge; one had a simple adaptive increase based on the best fitness within the population; and one put the level of difficulty under genetic control by coevolving the problem and the solution. The results showed that the coevolutionary system performed best on an amplifier

design problem, but the static system performed best of all. When discussing potential reasons as to why the incremental systems showed poorer performance, Lohn et al. recognized that the discontinuity in the fitness landscapes resulting from the adaptive nature of the fitness functions might have reduced the evolvability of the systems.

3.3.5 Dynamic Representations

Another proposal from ETL to improve the speed of evolution was to use a variable length representation, with the aim of reducing the size of the search space necessary for a problem. Applied to a pattern recognition problem, performance was improved over an algorithm that did not use variable-length representations, in terms of both solution parsimony and efficacy [44].

A similar approach was taken by Zebulum in an experiment to evolve Boolean functions using a chromosome of product terms that were summed by the fitness function [139]. However, the search order of representation space differed from the ETL experiments. Inspired by the observation that complex organisms have evolved from simpler ones, the population was seeded with short chromosomes. This approach assumes a correlation between complex behavior and complex structure. As we discussed earlier, Thompson has demonstrated that this is not necessarily true, since complexity in behavior can arise from interactions of a simple system with a complex environment [120]. However, the simplicity of the simulation used to evaluate circuit designs in this example may mean that in this case the assumption holds. A new operator was introduced to increase chromosome length, under the control of a fixed parameter. Hence a simple pressure to move from short representations to long ones was set. It was found that a low rate of increase allowed fully functional but more parsimonious solutions to be found over a larger rate.

In both these examples, each gene in the representation was mapped directly to a Boolean function, and the representation space was searched by adding and removing genes guided by evolution in the first case, and by adding genes guided by a simple heuristic in the second case. In both cases, only the size of the space searched was changeable, rather than any arrangement of the order; hence, the *evolvability* of the space remained unaltered.

3.3.6 Development

The use of evolution itself to explore representation space as a meta-search in addition to the search of design space is an attractive idea. This leaves the question of how to do so such that the search of representations achieves the following:

- it allows evolution to explore innovative design spaces;
- it allows evolution to explore design spaces of varying evolvability, not just size.

We have already explained that evolution searches design space from the bottom up, and that this is unlike approaches imposed by traditional top-down

design, allowing evolution to explore innovative areas of design space. We have also already mentioned how we have little understanding of how to make such searches more evolvable.

One approach we can take is to turn to nature to gain some insight into evolvability. The proof that bottom-up evolutionary design can be highly evolvable is all around us in the form of extremely complex biological organisms. However, Dawkins has noted that that the organisms that evolved early in evolutionary history have since then evolved the least [10], since most simple organisms present today are virtually unchanged since their appearance in the fossil record, whereas organisms that have evolved in more modern times have continued to evolve increasingly complex structure. This led Dawkins to suggest that biological evolution has over time discovered evolvable mechanisms that it has used to generate increasingly complex organisms: there has been an evolution of evolvability. This has led us to believe that we should look to differences between the mechanisms that simple and higher organisms employ to map from genotype to phenotype for sources of evolvability. A striking feature of higher organisms is their modularity. The period of evolutionary history in which organisms first made use of complex modular structures, the Cambrian period, heralded the appearance of Metazoan organisms and was marked by an explosion of evolution [49]. This would suggest that the idea of decomposing a problem into modules to improve evolvability is a good one. The mechanisms of problem decomposition previously used to evolve hardware designs relied on top-down human design abstractions. The mechanism by which all Metazoan organisms map genotype to phenotype is quite different. It is the process of development. Development provides a mechanism for evolutionary control over a *bottom-up* modularization process. It allows evolution to make use of any innovative design features it discovers at lower abstractions and to encapsulate them for reuse at a higher level of abstraction.

Development maps genotype to phenotype in an *indirect* process. It provides a series of instructions describing how to construct an organism [4]. It is also a *generative* process. It uses abstraction and iteration to manage the flow of control within the series of instructions [36]. In this sense, it can be likened to a traditional declarative computer program. Developmental systems that employ these ideas in an abstract sense have been explored for a number of years in the context of ANN design. They directly evolve programs that explicitly describe how a system should develop. The language in which the programs are described employ fixed, explicit mechanisms for abstraction and reuse. Such systems have been labeled as *explicit* developmental systems by Bentley and Kumar [4]. One such system is cellular encoding [24]. More recently, the same method has been used by Koza et al. to evolve analogue circuits [50]. The basic technique is to evolve trees of developmental steps using genetic programming (GP). Each developmental step, encoded as a GP node, explicitly codes for a phenotype modification. A fixed "embryonic" phenotype is "grown" by applying a tree of rules to it. Koza used automatically defined functions (ADFs) to explicitly provide modularity, and automatically defined copies (ADCs) to provide iteration. Lohn and Columbano have used a similar approach, but with a linear mapping representation that is applied to an embryonic circuit in an unfolding manner, rather than a circuit-modifying one [67]. The representational power is limited, although some but not all of these limitations have more

recently been removed by introducing new operators [6]. Although both systems have managed to evolve innovative designs, only Koza has demonstrated examples of modularization and reuse in his solutions, and these have been limited to a few examples that do not produce modularization and reuse on the order of that seen in biological organisms. This result might suggest that there are other features of biological development important to evolvability that are not captured by implementing such abstract ideas of modularization, reuse, and growth alone. To benefit from using a developmental genotype–phenotype mapping, the process by which biological development achieves these features should be modeled more closely.

Biological development describes the transformation of a single-celled embryo into a complex adult organism. The entire process is by no means completely understood. It encapsulates a huge array of interactions between genes, their products, and the environment, from microscopic to macroscopic, some of seemingly minor importance, some ubiquitous to all stages of development. One mechanism that has a hand in all stages of development is DNA transcription. Transcription regulates the rate of gene expression through the presence of proteins called transcription factors, which either increase (activators) or decrease (inhibitors) the transcription rate of a particular gene. All transcription factors are proteins that are generated by the expression of other genes. Thus a dynamic, autocatalytic network of gene products specifies which genes are expressed. These networks are called *gene regulatory networks (GRNs)* [103]. Such networks may be arranged as modules, controlled by a master control gene [58]. When activated, the master control gene causes a cascade of activity throughout a GRN module and generates a complex feature in a phenotype.

Evolution is able to manage the flow of control for the developmental program over time by manipulating gene products involved in GRNs. However, another mechanism is required to communicate flow of control over space. To achieve this, biology makes use of two processes: growth and induction. Growth occurs through cellular division; thus, regulatory substances within an ancestor cell can be distributed to all the cell's descendents as they spread through space. Development can control this process, for instance, by constraining the location of a regulatory substance within a cell such that, after cell cleavage, it is present only in one daughter cell. Such regulatory substances are known as *cytoplasmic determinants*. Induction is quite different. Here a cell encodes regulatory information as a chemical signal, which is transmitted to nearby cells. A variety of inductive signal types have been identified [103] that pass information over various localities and at various rates.

Evolutionary design systems that model these processes are termed *implicit* by Bentley and Kumar [4]. Flow of control in implicit systems is commonly modeled by successively rewriting a symbolic description of a simple object according to a set of rewriting rules. The map between genotype and phenotype is specified by a fixed start symbol for the rule rewriting process, and the grammar is evolved. One type of system that models both transcription and growth are L-Systems. These have been explored in the context of circuit design by Haddow and Tufte [27]. The L-System they used was context free; hence, the rules were rewritten such that there was no communication between adjacent structures. Hence, no concept of induction was modeled. Miller has explored a similar growth-based system that

incorporated a limited amount of context [82]. The phenotype consists of a single embryonic cell. The chromosome encodes a set of functions to determine the inputs and function of the cell and whether it should divide to produce two daughter cells. At each developmental timestep, the functions are executed in all current cells, and the process iterates. The arguments of shared functions are the positions of the inputs, current function, and location of that cell. Functions were used to determine the connections and function in the next step of development. Hence a form of communication is captured by the model through the labels of each cell's current neighbours affecting the current cell's next state. However, the communication between cells (and hence the model of induction) is present in a highly abstract and limited sense, and the role of induction in the development of the circuit cannot be separated from the role of growth.

3.3.7 An Example of Developmental Evolutionary Circuit Design

The recent work of Gordon [20] provides an alternative approach. With the eventual goal of evolving complex, functioning circuits, an exploratory system based on the three principles of being generative, implicit, and context-driven was designed. It was decided that a rule-based system could satisfy all three criteria. Like biological organisms, the phenotype is composed of "cells", but unlike biological organisms, the cells in our model are laid out on a two-dimensional grid, mirroring the medium of electronic circuits. This layout has the advantage of being easily mapped to a circuit design for a programmable logic device such as a Field Programmable Gate Array (FPGA), and so was in keeping with our aim of developing a system with as little computational overhead as possible. To update the entire individual for a single developmental timestep, the set of rules that make up the chromosome is tested against the "chemical environment" that is modeled in each of these cells. For each cell, only the rules that match that cell's environment are activated. If the environment differs between cells, it is possible for different rules to be activated in each cell, which leads to their environments being altered in different ways. In this way, different chemical environments can be maintained between cells. By modeling a cell's context with only transcription factors (proteins that activate genes) and ignoring all other chemistry present in biological cells, we were able to keep our model as simple as possible yet encapsulate the key features that provide a generative, implicit, context-driven process.

Transcription factor proteins were modeled as binary state variables. Each gene was modeled as a rule. The precondition of the rule specified which proteins must be present (activators) and which must be absent (inhibitors) in order for that particular gene to activate. The postcondition of the rule defines the protein that is generated if the rule is activated. An example rule is shown in Figure 12.4.

For a rule like this to be activated, the proteins in the environment must match the pattern of proteins specified in the rule precondition. There are five bits in the rule precondition for each protein in the model. The final three bits define the protein concentration that the operator will act upon. Hence a rule can specify concentration values to range from 0 to 7. The first two bits of the protein condition specify the operator—not equal to (00), less than or equal to (01), greater than or equal to (10), or equal to (11). The specific protein to be tested is determined by the locus of these bits. A set of these rules makes up the chromosome

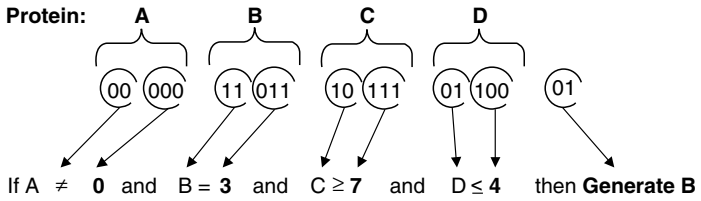

Figure 12.4. An example of a rule

and defines how the proteins interact over time. At each timestep in the developmental process, the environment is inspected to determine which proteins are present, and then each rule is inspected to determine whether the environment matches the rule. If it does, the rule is activated; the protein defined in the rule's postcondition is generated and goes on to make up part of the protein environment of the following timestep.

Context is a key feature of our model—cells must be able to affect their neighbor's environment. In our model, this is achieved through the interaction of proteins. Each cell inspects its neighbors to determine what proteins they are generating. The protein concentration detected by a cell is determined thus: for each protein, the cell sums the total number of neighbors that are generating that protein. If the cell itself is also generating that protein, it adds an additional 3 concentration points to the total. Thus the maximum concentration can be 7 points, since as 4 are contributed by the neighbors and 3 by the cell itself. To simulate this process, the cell model for our exploratory system contains a protein detector and a protein generator in order to record the proteins that are present in the cell and the proteins that are detected by the cell, respectively. To summarize, a complete developmental timestep for a cell proceeded thus:

1. For each protein in the model, the cell's protein detector sends a query to each of its von Neumann neighbors (i.e., the four neighbors to the north, south, east, and west on a 2D grid) to determine if they are generating that protein. It also queries its own generator, and sums the results from the neighbors and itself (with an additional bias towards itself) to give a detected concentration for that protein.
2. The rule set is tested against the pattern of proteins detected by the detector in step 1. As each rule with a precondition matching the cell's current pattern of *detected* proteins is activated, the cell's protein *generator* is updated to represent the protein specified in the rule postcondition.

These two steps are then repeated for a number of cycles, as shown in Figure 12.5, allowing the pattern of proteins formed across the global array of cells to change until a stable state or cycle of states is reached, or until development is halted after a predetermined number of timesteps. Gordon provides Full details [20].

The system described above so far models the process of forming patterns of gene products. What remains is for a mechanism to be introduced by which the patterns of gene products generate a circuit design. Each cell in our cellular array is mapped directly to a configurable logic block (CLB) on a Xilinx Virtex FPGA, and the activity of the genes in each cell are linked to alterations in the functional components in the CLB. This means that in addition to proteins, the models of

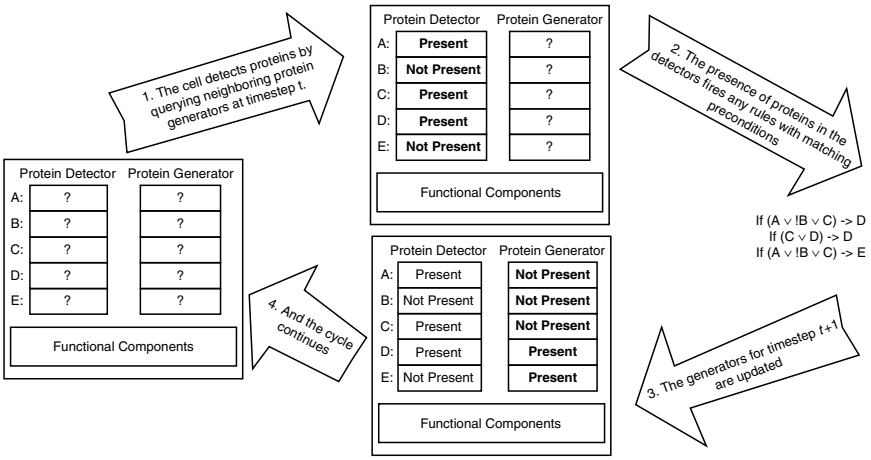

Figure 12.5. A developmental timestep highlighting the protein interaction model with a cell

our cells also contain functional components that map directly to functional components in a CLB. In order to keep the initial model simple, we added as few components as possible to our cell model. Each cell has four input wires that could be driven by its local neighbors, two 4-input lookup tables (LUTs), and an output wire from each LUT. The LUTs map directly to two of the four LUTs in a Virtex CLB, and the input and output wires map directly to manually selected single lines between the CLBs. For details of the Virtex architecture and how this mapping was made, see the work of Gordon and Bertley [21].

To allow these functional components to be altered by gene activity, we introduced new postconditions to the rules. These coded for an alteration to the logic in a CLB. Over the course of development, the activities of these circuit-altering postconditions were recorded by activity counters – one counter in each cell for each circuit-altering postcondition—and once development was complete, the activity counters were inspected in order to determine what alterations should be made to a predefined template circuit on the Virtex. Gordon and Bertley give details of this process [21].

Numerous experiments have been carried out on this model and variations of it [19]. The results showed the importance of good intercellular communication to improve development's ability to generate and maintain a range of patterns. The work has shown that the computational power of this developmental model is sufficient to allow the learning of patterns that map to fully functional adder circuits [20]. This is an important step towards tackling real-world problems with development.

3.4 Platform Research

We have now reviewed most current research into evolvable hardware. We have seen that many researchers believe that working at low levels of abstraction can have advantages. We have also seen that mechanisms to explore evolvability and generalization are being actively investigated. What we have not considered is the availability of platforms for low-abstraction hardware evolution.

In this section, we cover the platforms that have been reported in the evolvable hardware literature. Some are commercially available, and some have been developed by researchers. Commercial devices have not been developed with evolvable hardware as a primary goal, and so most struggle to compete with dedicated evolvable hardware on performance, versatility, and ease of use for our purposes. However, they do have advantages of availability and cost (although some that were used for early research are now no longer available), and so many researchers have explored their use for evolvable hardware.

3.4.1 Criteria for successful evolutionary platforms

Thompson [115] has listed a number of criteria for intrinsic circuit evolution platforms. These are discussed below:

Reconfigurable an unlimited number of times. Many field programmable devices are designed to be programmed only once. Others are designed to be programmed a small number of times, but repeated configuration can eventually cause damage. Evolutionary experiments can require millions of evaluations, and so devices for intrinsic experiments should be able to be reconfigured infinitely.

Fast and / or partial reconfiguration. If millions of evaluations are needed, the evaluation process should be fast. Modern programmable devices have millions of configurable transistors and consequently have large configuration bitstreams. This can mean that downloading the configuration becomes the bottleneck of the evolutionary process. The brute force solution to this problem is to use devices with high bandwidth configuration ports. Another solution is to evaluate many individuals at once, as proposed by Higuchi, Iba, and Manderick, among others [33]. Batch evaluation limits the type of evolutionary algorithm to those with large populations, ruling out the use of steady-state genetic algorithms or low-population evolutionary strategies. A more elegant solution is that of partial reconfiguration, where only the changes from the current configuration need to be uploaded. This yields similar bandwidth use with no constraints on the learning algorithm.

Indestructibility or validity checking. In conventional CMOS technologies, a wire driven from two sources can result a short circuit if one drives the wire to a different voltage level than another. The high currents generated from such an event are extremely undesirable, as they can damage the device, and so should be prevented by hard constraints, rather than the softer ones advocated so far. Some hardware platforms are designed around an architecture with which contention is impossible. For those that are not, there are two options—either an abstract architecture can be imposed on top of the real hardware, or circuits can be tested for contention before they are synthesized, and evaluated by an alternative means if such a condition is detected.

Fine-grain reconfigurability. In order to allow evolution the ability to innovate, evolution must be able to manipulate candidate circuits at a low level of abstraction. Hence a good platform needs fine-grain control over the evolving platform.

Thompson also points out the distinction between fine-grain architectures and fine-grain reconfigurability—namely, that although a device's architecture may be based on repeated large units, if these can be reconfigured at a finer level, then this criterion will be met.

Flexible I/O. The method of supplying input and retrieving output from an evolved circuit can affect the feasibility of successful evolution, so a platform that allows experimentation with this is useful.

Low cost. This is of particular importance when the motive behind using evolution is to lower costs through design automation.

Observability. In order to analyze how evolved circuits work, their internal signals need to be probed. However, when working with low-design abstractions, it may be impossible to avert the potential of signal probes to change the behavior of the circuit, and the probed signal architectures should be chosen with this as a consideration.

3.4.2 Platforms

Bearing these criteria in mind, the platforms that have been used or proposed for use for evolvable hardware experiments are now considered briefly. These can be classified into three groups: commercial digital, commercial analogue, and research platforms. They are tabulated below.

Commercial Analogue Platforms
Zetex TRAC [14]: Based around 2 pipelines of 10 op-amps + programmable capacitors, resistors. Linear and nonlinear functions successfully evolved. Large-grained reconfigurability and limited topology limit worth for evolution.
Anadigm FPAA(Inc. 2003): Up to 4 reconfigurable blocks with programmable interconnect. CABs contain 2 op-amps, capacitor banks, serial approximation register. Large-grained reconfigurability limits worth for evolution. No reports on use for evolvable hardware.
Lattice ispPAC [92]: Designed for filtering applications. Based on programmable amplifiers. Limited reconfigurability (~10,000x) limits suitability for evolvable hardware.
Motorola MPAA020 [136]: 20 cells containing an op. amp, comparator, transistors, capacitors, and SRAM. A range of circuits has been evolved. Much of the bitstream is proprietary. Geared towards circuits based around the op. amp. No longer available.

Commercial Digital Platforms
Xilinx 6200 [115, 116, 117, 50, 121]: Developed for dynamic reconfig. apps. Fast and infinite reconfig., fully or partially. Homogenous fine-grained architecture of MUXes. All configurations valid. Good I/O. Expensive, no longer available.
Xilinx XC4000 [57]: Low cost, infinite but slow reconfig. SRAM LUT based architecture. Damaged by invalid configurations. Parts of bitstream proprietary and undisclosed. Reconfigurable at resource level using Xilinx JBits software. No longer available.

Xilinx Virtex/II/II Pro [35, 56]: Medium cost. SRAM LUT based architecture. Can be reconfigured infinitely and quickly, fully and partially. Can be damaged by random configurations. Some of the bitstream is proprietary and undisclosed, but most hardware resources can be reconfigured using Xilinx JBits software. Virtex II provides embedded multipliers, Virtex II Pro provides embedded CPU core. Widely available.

Research Platforms
FPTA [105, 53, 13]: Reconfigurable at transistor level, additionally supporting capacitors and multiple I/O points. Programmable voltages control resistances of connecting switches for use as additional transistors. Some versions allow variable channel height and width. FPTA2 provides 8×8 array of FPTA cells Fits criteria for evolvable hardware well.
Embryonic Arrays [113, 126]: Bio-inspired fault tolerant FPGA architecture. Programmable cells usually based on MUXtrees. New POEtic tissue designed to support hierachical logical genotype, developmental and phenotype layers. Interesting architecture for developmental hardware evolution.
Palmo [29]: PWM-based signaling rather than true analogue. Based around array of integrators. All configurations valid.
Evolvable Motherboard [54]: Array of analogue switches, connected to six interchangeable evolvable units. Evolution of gates, amplifiers, and oscillators demonstrated using bipolar transistors as evolvable unit. Good I/O. Board-based architecture is not suitable for real-world problems due to size, cost, and number of evolvable units.
FIPSOC [86]: Complete evolutionary system aimed at mixed signal environments. Analogue and digital units. CPU and memory to encode evolutionary algorithm. Analogue units based around amplifiers. Digital units based on LUTs and flip-flops. Context-based dynamic reconfiguration suitable for real-time adaptive systems.
PAMA [96]: Fixed analogue MUX array allowing interconnection of interchangeable evolvable units. Current version implements a 32 16:1 bidirectional low on-resistance MUX/deMUX allowing for random configurations.

4 SUMMARY

The problems of electronic circuit design are increasing as demand for improvements increases. In this review, we have introduced a promising new type of solution to these difficulties: evolvable hardware. This emerging field exists at the intersection of electronic engineering, computer science, and biology.

The benefits brought about by evolvable hardware are particularly suited to a number of applications, including the design of low-cost hardware, poorly specified problems, creation of adaptive systems, fault-tolerant systems, and innovation.

The chapter has also reviewed and analyzed current research trends in evolvable hardware in depth. In particular, the research focusing on innovation, evolvability, and platforms have been described, and a recent example of a developmental evolutionary electronics system designed by the authors has been provided.

Evolvable hardware is still a young field. It does not have all the answers to the problems of circuit design, and there are still many difficulties to overcome. Nevertheless, these new ideas may be one of the brightest and best hopes for the future of electronics.

ACKNOWLEDGMENTS

The authors would like to thank Dr. Peter Rounce for his insights and advice.

REFERENCES

[1] V. Aggarwal (2003): *Evolving sinusoidal oscillators using genetic algorithms*. 2003 NASA/DoD Conference on Evolvable Hardware, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[2] L. Altenberg, (1995): The Schema Theorem and Price's Theorem. *Foundations of Genetic Algorithms 3*. D. Whitley and M. D. Vose. San Mateo, CA, U.S.A., Morgan Kaufmann: 23–49.

[3] T. Arslan and D. H. Horrocks (1995): The Design of Analogue and Digital Filters Using Genetic Algorithms. *15th SARAGA Colloquium on Digital and Analogue Filters and Filtering Systems*, London, U.K.

[4] P. J. Bentley and S. Kumar (1999): Three Ways to Grow Designs: A Comparison of Embryogenies for an Evolutionary Design Problem. *Proceeding of the Genetic and Evolutionary Computation Conference*, Orlando, FL, U.S.A.

[5] J. A. Biles (1994): GenJam: A Genetic Algorithm for Generating Jazz Solos. *Proceedings of the 1994 International Computer Music Conference*, San Francisco, CA, U.S.A., International Computer Music Association.

[6] J. P. B., Botelho, L. B. Sa, et al. (2003): An experiment on nonlinear synthesis using evolutionary techniques based only on CMOS transistors. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[7] D. W. Bradley and A. M. Tyrrell (2001): The architecture for a hardware immune system. *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*. EH 2001. 12–14 July 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[8] R. O. Canham and A. M. Tyrrell (2002): Evolved Fault Tolerance in Evolvable Hardware. *2002 World Congress on Computational Intelligence*, Honolulu, HI, U.S.A., IEEE, Piscataway, NJ, USA.

[9] E., Damiani and V. Liberali, et al. (2000): Dynamic Optimisation of Non-linear Feed-Forward Circuits. *3rd International Conference on Evolvable Systems*, Edinburgh, U.K.

[10] R. Dawkins, (1989): The evolution of evolvability. *Proceedings of Artificial Life*: *The Quest for a New Creation*, Santa Fe, U.S.A., Addison-Wesley.

[11] J. F. M., do Amaral, J. L. M. do Amaral, et al. (2002): Towards Evolvable Analog Fuzzy Logic Controllers. *2002 NASA/DoD Conference on Evolvable Hardware*, Alexandria, VA, U.S.A., IEEE Press.

[12] M. Dorigo, and G. Di Caro (1999): The Ant Colony Optimization Meta-heuristic. *New Ideas in Optimization*. D. Corne, M. Dorigo and F. Glover. London, UK, McGraw-Hill**:** 11–32.

[13] I., Ferguson, A. Stoica, et al. (2002): An Evolvable Hardware Platform based on DSP and FPTA. *2002 Genetic and Evolutionary Computation Conference*, Memlo Park, CA, U.S.A., AAAI Press.

[14] S. J. Flockton and K. Sheehan (1999): A system for intrinsic evolution of linear and non-linear filters. *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*. 19–21 July 1999, Pasadena, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[15] A. Fukunaga and A. Stechert (1998): Evolving Nonlinear Predictive Models for Lossless Image Compression with Genetic Programming. *Third Annual Genetic Programming Conference*, Madison, WI, U.S.A.

[16] J. C. Gallagher, (2003): The once and future analog alternative: evolvable hardware and analog computation. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[17] M. Garvie and A. Thompson (2003): Evolution of Self-diagnosing Hardware. *5th International Conference on Evolvable Systems*, Trondheim, Norway, Springer-Verlag.

[18] N., Göckel, R. Drechsler, et al. (1997): A Multi-Layer Detailed Routing Approach based on Evolutionary Algorithms. *Proceedings of the IEEE International Conference on Evolutionary Computation*, Indianapolis, IN, U.S.A.

[19] D. E. Goldberg (1989): Genetic algorithms in search, optimization, and machine learning. Reading, Mass.; Harlow, Addison-Wesley.

[20] T. G. W. Gordon (2003): Exploring Models of Development for Evolutionary Circuit Design. *2003 Congress on Evolutionary Computation*, Canberra, Australia.

[21] T. G. W. Gordon and P. J. Bentley (2002): Towards Development in Evolvable Hardware. *2002 NASA/DoD Conference on Evolvable Hardware*, Washington D.C., U..S.A.

[22] G. W. Greenwood and X. Song (2002): How to Evolve Safe Control Strategies. *2002 NASA/DoD Conference on Evolvable Hardware*, Alexandria, VA, U.S.A., IEEE Press.

[23] J. B. Grimbleby (2000): Automatic Analogue Circuit Synthesis Using Genetic Algorithms. *IEE Proceedings on Circuits Devices and Systems 147*(6): 319–323.

[24] F. Gruau (1994): Neural Network Synthesis Using Cellular Encoding and the Genetic Algorithm. *Laboratoire de l'Informatique du Parallilisme*. Lyon, Ecole Normale Supirieure de Lyon: 151.

[25] X., Guo, A. Stoica, et al. (2003): Development of consistent equivalent models by mixed-mode search. *IASTED International Conference on Modeling and Simulation*, Palm Springs, California, U.S.A.

[26] D. A. Gwaltney and M. I. Ferguson (2003): Intrinsic hardware evolution for the design and reconfiguration of analog speed controllers for a DC Motor. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[27] P. C., Haddow, G. Tufte, et al. (2001): Shrinking the Genotype: L-systems for EHW? *The 4th International Conference on Evolvable Systems*: From Biology to Hardware, Tokyo, Japan.

[28] P. C. Haddow and P. van-Remortel (2001): From here to there: future robust EHW technologies for large digital designs. *Proceedings Third*

NASA/DoD Workshop on Evolvable Hardware, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[29] A., Hamilton, K. Papathanasiou, et al. (1998): Palmo: Field Programmable Analogue and Mixed-signal VLSI for Evolvable Hardware. *2nd International Conference on Evolvable Systems*, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[30] M., Hartmann, P. Haddow, et al. (2002): Evolving robust digital designs. *2002 NASA/DoD Conference on Evolvable Hardware*. 15–18 July 2002, Alexandria, VA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[31] I. Harvey, (1991): Species Adaptation Genetic Algorithms: The basis for a continuing SAGA. *1st European Conference on Artificial Life*, Paris, France.

[32] I. Harvey and A. Thompson (1997): Through the labyrinth, evolution finds a way: A silicon ridge. *1st International Conference on Evolvable Systems*, Tsukuba, Japan, Springer-Verlag, Berlin, Germany.

[33] T., Higuchi, H. Iba, et al. (1994): Evolvable Hardware. *Massively Parallel Artifical Intelligence*. Cambridge, MA, U.S.A., MIT Press: 398-421.

[34] T., Higuchi, M. Iwata, et al. (1996): Evolvable hardware and its application to pattern recognition and fault-tolerant systems. *Proceedings of Towards Evolvable Hardware*: An International Workshop. 2–3 Oct. 1995, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[35] G., Hollingworth, S. Smith, et al. (2000): The Intrinsic Evolution of Virtex Devices Through Internet Reconfigurable Logic. *Proceedings of the Third International Conference on Evolvable Systems*, Edinburgh, U.K.

[36] G. Hornby (2003): Generative Representations for Evolutionary Design Automation. Department of Computer Science. Waltham, MA, U.S.A., Brandeis University.

[37] B. I. Hounsell and T. Arslan (2000): A novel genetic algorithm for the automated design of performance driven digital circuits. *2000 Congress on Evolutionary Computation*, La Jolla, CA, USA, IEEE, Piscataway, NJ, USA.

[38] B. L. Hounsell and T. Arslan (2001): Evolutionary design and adaptation of digital filters within an embedded fault tolerant hardware platform. *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*. EH 2001. 12–14 July 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[39] M. A., Huynen, P. F. Stadler, et al. (1996): Smoothness within ruggedness: The role of neutrality in adaptation. *Proceedings of the National Academy of Science* 93.

[40] K., Imamura, J. A. Foster, et al. (2000): The test vector problem and limitations to evolving digital circuits. *2nd NASA/DoD Workshop on Evolvable Hardware*, Palo Alto, CA, U.S.A., IEEE Comput. Soc., Los Alamitos, CA, USA.

[41] A. Inc. (2003): AN120E04 FPAA Data Sheet, http://www.anadigm.com. 2004.

[42] M., Iwata, I. Kajitani, et al. (1996): A pattern recognition system using evolvable hardware. *4th International Conference on Parallel Problem Solving from Nature PPSN IV*, Berlin, Germany, Springer-Verlag, Berlin, Germany.

[43] D., Job, V. Shankararaman, et al. (1999): Hybrid AI Techniques for Software Design. *Proceedings of the 11th International Conference on Software Engineering and Knowledge Engineering*, Kaiserslautern, Germany.

[44] I., Kajitani, T. Hoshino, et al. (1996): Variable length chromosome GA for evolvable hardware. *3rd IEEE International Conference on Evolutionary Computation*, Nagoya, Japan, IEEE, New York, NY, USA.

[45] I., Kajitani, T. Hoshino, et al. (1999): An Evolvable Hardware Chip and Its Application as a Multi-Function Prosthetic Hand Controller. *16th National Conference on Artificial Intelligence*, Orlando, FL, U.S.A., AAAI Press.

[46] T., Kalganova, J. F. Miller, et al. (1998): Some aspects of an evolvable hardware approach for multiple-valued combinational circuit design. *2nd International Conference on Evolvable Systems, Lausanne*, Switzerland, Springer-Verlag, Berlin, Germany.

[47] S. Kauffman and S. Levin (1987): Towards a General Theory of Adaptive Walks on Rugged Landscapes. *Journal of Theoretical Biology*. *128*: 11-45.

[48] S., Kazadi, Y. Qi, et al. (2001): Insufficiency of piecewise evolution. *3rd NASA/DoD Workshop on Evolvable Hardware*, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[49] M. Kirschner and J. Gerhart (1998): Evolvability. *Proceedings of the National Acadamy of Science 95*(8): 420–8427.

[50] J., Koza, F. H. I. Bennett, et al. (1999): *Genetic Programming III*. San Francisco, California, U.S.A., Morgan-Kauffmann.

[51] J. R., Koza, M. A. Keane, et al. (2000): Automatic creation of human-competitive programs and controllers by means of genetic programming. *Genetic Programming and Evolvable Machines 1*(1-2): 121–64.

[52] W. B. Langdon, (1997): Scheduling Maintenance of Electrical Power Transmission. *Artificial Intelligence Techniques in Power Systems*. K. Warwick and A. O. Ekwue. London, IEE Press: 220-237.

[53] J., Langeheine, J. Becker, et al. (2001): A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits. *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*. EH 2001. 12–14 July 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[54] P. Layzell (1998): A new research tool for intrinsic hardware evolution. *2nd International Conference on Evolvable Systems*, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[55] P. Layzell and A. Thompson (2000): Understanding Inherent Qualities of Evolvaed Circuits: Evolutionary History as a Predictor of Fault Tolerance. *3rd International Conference on Evolvable Systems*, Edinburgh, U.K., Springer-Verlag.

[56] D. Levi (2000): HereBoy: a fast evolutionary algorithm. *The Second NASA/DoD Workshop on Evolvable Hardware*., Palo Alto, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[57] D. Levi and S. A. Guccione (1999): GeneticFPGA: evolving stable circuits on mainstream FPGA devices. *1st NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA, U.S.A., IEEE Comput. Soc., Los Alamitos, CA, USA.

[58] E. B. Lewis (1992): Clusters of master control genes regulate the development of higher organisms. *Journal of the American Medical Association 267*: 1524–1531.

[59] J. H. Li and M. H. Lim (2003): Evolvable Fuzzy System for ATM Cell Scheduing. *5th International Conference on Evolvable Systems*, Trondheim, Norway, Springer-Verlag.

[60] D. S. Linden (2001): A system for evolving antennas in-situ. *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*. EH 2001. 12–14 July 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[61] D. S. Linden (2002): An evolvable antenna system for optimizing signal strength in-situ. *IEEE Antennas and Propagation Society International Symposium*, vol.1, 16–21 June 2002, San Antonio, TX, USA, IEEE, Piscataway, NJ, USA.

[62] D. S. Linden (2002): Optimizing signal strength in-situ using an evolvable antenna system. *2002 NASA/DoD Conference on Evolvable Hardware*. 15–18 July 2002, Alexandria, VA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[63] D. S. Linden and E. E. Altshuler (1999): Evolving wire antennas using genetic algorithms: a review. *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*. 19–21 July 1999, Pasadena, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[64] A. Lindenmayer (1968): Mathematical models for cellular interactions in development I Filaments with one-sided inputs. *Journal of Theoretical Biology 18*: 280–289.

[65] W., Liu, M. Murakawa, et al. (1997): ATM cell scheduling by function level evolvable hardware. *1st International Conference on Evolvable Systems*, Tsukuba, Japan, Springer-Verlag, Berlin, Germany.

[66] J., Lohn, G. Larchev, et al. (2003): A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs. *5th International Conference on Evolvable Systems, Trondheim*, Norway, Springer-Verlag.

[67] J. D. Lohn and S. P. Colombano (1998): Automated analog circuit synthesis using a linear representation. *2nd International Conference on Evolvable Systems*, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[68] J. D., Lohn, G. L. Haith, et al. (1999): A comparison of dynamic fitness schedules for evolutionary design of amplifiers. *1st NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[69] J. D., Lohn, D. S. Linden, et al. (2003): Evolutionary Design of an X-Band Antenna for NASA's Space Technology 5 Mission. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL.

[70] S. J. Louis (2003): Learning for evolutionary design. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[71] M., Lukac, M. A. Perkowski, et al. (2003): Evolutionary Approach to Quantum and Reversible Circuits Synthesis. *Artificial Intelligence Review 20*(3-4): 361–417.

[72] N. J. Macias and L. J. K. Durbeck (2002): Self-assembling circuits with autonomous fault handling. *2002 NASA/DoD Conference on Evolvable Hardware*. 15–18 July 2002, Alexandria, VA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[73] J., Masner, J. Cavalieri, et al. (1999): Representation and robustness for evolved sorting networks. *1st NASA/DoD Workshop on Evolvable Hardware*, Pasadena, CA, U.S.A., IEEE Comput. Soc., Los Alamitos, CA, USA.

[74] P. Mazumder and E. M. Rudnick (1999): *Genetic Algorithms for VLSI Design, Layout and Test Automation*. Upper Saddle River, NJ, U.S.A., Prentice-Hall.

[75] J. F. Miller and K. Downing (2002): Evolution in materio: looking beyond the silicon box. *2002 NASA/DoD Conference on Evolvable Hardware*. 15–18 July 2002, Alexandria, VA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[76] J. F., Miller, D. Job, et al. (2000): Principles in the Evolutionary Design of Digital Circuits -Part I. *Genetic Programming and Evolvable Machines 1*(1/2): 7–35.

[77] J. F., Miller, D. Job, et al. (2000): Principles in the Evolutionary Design of Digital Circuits -Part II. *Genetic Programming and Evolvable Machines* 1(3): 259–288.

[78] J. F., Miller, T. Kalganova, et al. (1999): The Genetic Algorithm as a Discovery Engine: Strange Circuits and New Principles. *Proceedings of the AISB Symposium on Creative Evolutionary Systems*, Edinburgh, U.K.

[79] J. F. Miller and P. Thomson (1998): Aspects of Digital Evolution: Evolvability and Architecture. *5th International Conference on Parallel Problem Solving from Nature*, Amsterdam, The Netherlands, Springer-Verlag.

[80] J. F. Miller and P. Thomson (1998): Aspects of digital evolution: geometry and learning. *Proceedings of Second International Conference on Evolvable Systems*: From Biology to Hardware. (ICES 98). 23–25 Sept. 1998, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[81] J. F. Miller and P. Thomson (1998): Evolving Digital Electronic Circuits for Real-Valued Function Generation using a Genetic Algorithm. *3rd Annual Conference on Genetic Programming*, San Francisco, CA, U.S.A,.

[82] J. F. Miller and P. Thomson (2003): A Developmental Method for Growing Graphs and Circuits. *5th International Conference on Evolvable Systems*, Trondheim, Norway, Springer-Verlag.

[83] J. F., Miller, P. Thomson, et al. (1997): Designing electronic circuits using evolutionary algorithms. Arithmetic circuits: a case study. Applications of Computer Systems. *Proceedings of the Fourth International Conference*. 13–14 Nov. 1997, Szczecin, Poland, Wydwnictwo i Drukarnia Inst. Inf. Polytech. Szczecinskiej, Szezecin, Poland.

[84] T. M. Mitchell (1997): *Machine Learning*. London, McGraw-Hill.

[85] G. E. Moore (1965): Cramming More Components Onto Integrated Circuits. *Electronics 38*(8): 114–117.

[86] J. M., Moreno, J. Madrenas, et al. (1998): Feasible, evolutionary and self-repairing hardware by means of the dynamic reconfiguration capabilities of

the FIPSOC devices. *2nd International Conference on Evolvable Systems*, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[87] M., Murakawa, T. Adachi, et al. (2002): An AI-calibrated IF filter: a yield enhancement method with area and power dissipation reductions. *2002 IEEE Custom Integrated Circuits Conference*, Singapore.

[88] M. Murakawa, S. Yoshizawa, et al. (1998): Analogue EHW chip for intermediate frequency filters. *Proceedings of Second International Conference on Evolvable Systems*: *From Biology to Hardware*. (ICES 98). 23–25 Sept. 1998, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[89] M. Murakawa, S. Yoshizawa, et al. (1996): Hardware evolution at function level. *5th Conference on Parallel Problem Solving from Nature*, Berlin, Germany, Springer-Verlag, Berlin, Germany.

[90] M. Murakawa, S. Yoshizawa, et al. (1999): The GRD chip: Genetic reconfiguration of DSPs for neural network processing. *IEEE Transactions on Computers 48*(6): 628–639.

[91] J. Plante, H. Shaw, et al. (2003): Overview of Field Programmable Analog Arrays as Enabling Technology for Evolvable Hardware for High Reliability Systems. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, U.S.A., IEEE Press.

[92] E. Ramsden (2001): The ispPAC family of reconfigurable analog circuits. *3rd NASA/DoD Workshop on Evolvable Hardware*, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[93] D. E. Rumelhart, B. Widrow, et al. (1994): The Basic Ideas in Neural Networks. *Communications of the ACM 37*(3): 87–92.

[94] H. Sakanashi, M. Iwata, et al. (2001): A Lossless Compression Method for Halftone Images using Evolvable Hardware. *4th International Conference on Evolvable Systems*, Tokyo, Japan, Springer-Verlag.

[95] M. Salami, M. Murakawa, et al. (1996): Data compression based on evolvable hardware. *1st International Conference on Evolvable Systems from Biology to Hardware*, Tsukuba, Japan, Springer-Verlag, Berlin, Germany.

[96] C. C. Santini, R. Zebulum, et al. (2001): PAMA-programmable analog multiplexer array. *3rd NASA/DoD Workshop on Evolvable Hardware*, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[97] Sechen (1988): *VLSI Placement and Global Routing Using Simulated Annealing*. Boston, MA, U.S.A, Kluwer Academic Publishers.

[98] L. A. Segel and I. Cohen, Eds. (2001): Design Principles for the Immune System and Other Distributed Autonomous Systems. *Santa Fe Institute Studies in the Sciences of Complexity*. New York, Oxford University Press.

[99] L. Sekanina (2002): Evolution of digital circuits operating as image filters in dynamically changing environment. *8th International Conference on Soft Computing*, Brno, CZ.

[100] L. Sekanina (2003): Easily Testable Image Operators: The Class of Circuits Where Evolution Beats Engineers. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, U.S.A., IEEE Press.

[101] L. Sekanina (2003): Towards Evolvable IP Cores for FPGAs. *2003 NASA/Dod Conference on Evolvable Systems*, Chicago, IL, U.S.A., IEEE Press.

[102] H. T. Sinohara, M. A. C. Pacheco, et al. (2001): Repair of analog circuits: extrinsic and intrinsic evolutionary techniques. *Proceedings Third NASA/DoD Workshop on Evolvable Hardware*. EH 2001. 12–14 July 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[103] J. M. W. Slack (1991): *From Egg to Embryo*. Cambridge, Cambridge University Press.

[104] A. Stoica, A. Fukunaga, et al. (1998): Evolvable hardware for space applications. *Second International Conference on Evolvable Systems*: *From Biology to Hardware*. (ICES 98). 23–25 Sept. 1998, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.

[105] A. Stoica, D. Keymeulen, et al. (1999): Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital CMOS circuits. *Proceedings of the First NASA/DoD Workshop on Evolvable Hardware*. 19–21 July 1999, Pasadena, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[106] A. Stoica, D. Keymeulen, et al. (2001): Evolvable hardware solutions for extreme temperature electronics. *3rd NASA/DoD Workshop on Evolvable Hardware*., Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[107] A. Stoica, R. Zebulum, et al. (2000): Mixtrinsic Evolution. *3rd International Conference on Evolvable Systems*, Edinburgh, U.K.

[108] A. Stoica, R. Zebulum, et al. (2002): On polymorphic circuits and their design using evolutionary algorithms. *20th IASTED International Multiconference on Applied Informatics*, Innsbruck, Austria, ACTA Press, Anaheim, CA, USA.

[109] A. Stoica, R. S. Zebulum, et al. (2003): Silicon validation of evolution-designed circuits. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[110] S. Sundaralingam and K. C. Sharman (1998): Evolving Complex Adaptive IIR Structures. *9th European Signal Processing Conference*, Rhodes, Greece.

[111] A. J. Surkan and A. Khuskivadze (2002): Evolution of quantum computer algorithms from reversible operators. *2002 NASA/DoD Conference on Evolvable Hardware*. Alexandria, VA, U.S.A., IEEE Comput. Soc., Los Alamitos, CA, USA.

[112] E. Takahashi, Y. Kasai, et al. (2003): A Post-Silicon Clock Timing Adjustment Using Genetic Algorithms. *2003 Symposium on VLSI circuits*, IEEE Press.

[113] G. Tempesti, D. Mange, et al. (2002): The BioWall: an electronic tissue for prototyping bio-inspired systems. *2002 NASA/DoD Conference on Evolvable Hardware*, Alexandria, VA, U.S.A., IEEE Comput. Soc., Los Alamitos, CA, USA.

[114] A. Thompson (1995): Evolving electronic robot controllers that exploit hardware resources. Advances in Artificial Life. *Third European Conference on Artificial Life*. *Proceedings. 4–6 June 1995*, Granada, Spain, Springer-Verlag, Berlin, Germany.

[115] A. Thompson (1996): An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics. *1st International Conference on Evolvable Systems*, Springer-Verlag.

[116] A. Thompson (1996): Hardware Evolution. Brighton, U.K., University of Sussex.
[117] A. Thompson (1996): Silicon Evolution. *Proceedings of the 1st Annual Conference on Genetic Programming*, Stanford, CA, U.S.A.
[118] A. Thompson (1998): On the automatic design of robust electronics through artificial evolution. *Proceedings of Second International Conference on Evolvable Systems*: From Biology to Hardware. (ICES 98). 23–25 Sept. 1998, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.
[119] A. Thompson (2002): Notes on design through artificial evolution: Opportunities and algorithms. *Adaptive computing in design and manufacture 5*(1): 17–26.
[120] A. Thompson, I. Harvey, et al. (1996): Unconstrained Evolution and Hard Consequences. *Towards Evolvable Hardware: The Evolutionary Engineering Approach*. E. Sanchez and M. Tomassini. Berlin, Germany, Springer-Verlag. *1062*: 136–165.
[121] A. Thompson and P. Layzell (2000): Evolution of Robustness in an Electronics Design. *Proceedings of the 3rd International Conference on Evolvable Systems*: *From Biology to Hardware*, Edinburgh, U.K.
[122] A. Thompson and C. Wasshuber (2000): Evolutionary design of single electron systems. *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*. 13–15 July 2000, Palo Alto, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.
[123] R. Thomson and T. Arslan (2003): The evolutionary design and synthesis of non-linear digital VLSI systems. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.
[124] J. Torresen (2000): Possibilities and limitations of applying evolvable hardware to real-world applications. *Proceedings of FPL 2000. 10th International Conference on Field Programmable Logic and Applications*. 27–30 Aug. 2000, Villach, Austria, Springer-Verlag, Berlin, Germany.
[125] G. Tufte and P. C. Haddow (2000): Evolving an adaptive digital filter. *Proceedings. The Second NASA/DoD Workshop on Evolvable Hardware*. 13–15 July 2000, Palo Alto, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.
[126] A. Tyrrell, E. Sanchez, et al. (2003): POEtic Tissue: An Integrated Architecture for Bio-inspired Hardware. *5th International Conference on Evolvable Systems*, Trondheim, Norway.
[127] A. M. Tyrrell, G. Hollingworth, et al. (2001): Evolutionary strategies and intrinsic fault tolerance. *3rd NASA/DoD Workshop on Evolvable Hardware*. EH 2001, Long Beach, CA, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.
[128] V. Vassilev and J. F. Miller (2000): The Advantages of Landscape Neutrality in Digital Circuit Evolution. *Proceedings of the 3rd International Conference on Evolvable Systems*: *From Biology to Hardware*, Edinburgh, U.K.
[129] V. Vassilev and J. F. Miller (2000): Embedding Landscape Neutrality To Build a Bridge from the Conventional to a More Efficient Three-bit

Multiplier Circuit. *Genetic and Evolutionary Computation Conference*, Las Vegas, NV, U.S.A.

[130] S. Vigander (2001): Evolutionary Fault Repair of Electronics in Space Applications. Trondheim, Norway, Norwegian University Sci. Tech.

[131] K. A. Vinger and J. Torresen (2003): Implementing evolution of FIR-filters efficiently in an FPGA. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[132] X. Yao and T. Higuchi (1997): Promises and challenges of evolvable hardware. *1st International Conference on Evolvable Systems*: *From Biology to Hardware*, Tsukuba, Japan, Springer-Verlag, Berlin, Germany.

[133] J. S. Yih and P. Mazumder (1990): A Neural Network Design for Circuit Partitioning. *IEEE Transactions on Computer Aided Design 9*(10): 1265–1271.

[134] R. S. Zebulum, M. Aurélio Pacheo, et al. (1997): Increasing Length Genotypes in Evolutionary Electronics. *7th International Conference on Genetic Algorithms*, East Lansing, MI, U.S.A.

[135] R. S. Zebulum, D. Keymeulen, et al. (2003): Experimental results in evolutionary fault-recovery for field programmable analog devices. *2003 NASA/DoD Conference on Evolvable Hardware*, Chicago, IL, USA, IEEE Comput. Soc., Los Alamitos, CA, USA.

[136] R. S. Zebulum, M. A. Pacheco, et al. (1998): Analog circuits evolution in extrinsic and intrinsic modes. *2nd International Conference on Evolvable Systems*, Lausanne, Switzerland, Springer-Verlag, Berlin, Germany.