

THE NATURAL WAY TO EVOLVE HARDWARE

Adrian Thompson

Inman Harvey

Philip Husbands

School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, UK.

adrianth, inmanh, philh@cogs.susx.ac.uk

ABSTRACT

Artificial evolution can automatically derive the configuration of a reconfigurable hardware system such that it performs a given task. Individuals of the evolving population are evaluated when instantiated as real circuits, so if constraints inherent to human design (but not to evolution) are dropped, then the natural physical dynamics of the hardware can be exploited in new ways. The notion of an artificially evolving ‘species’ (SAGA) allows the open-ended incremental evolution of complex circuits. Theoretical arguments are given, as well as the real-world example of an evolved hardware robot controller.

1. INTRODUCTION

The use of artificial evolution to design electronic circuits automatically is attractive when conventional design fails. This is in situations where the desired circuit behaviour is difficult to specify, or where the nature of the circuit — e.g. asynchronous, continuous-time, analogue, high-complexity, low power, low area, fault-tolerant — does not yield to traditional approaches. This paper¹ discusses the new issues which must be faced in tackling such applications. Firstly, the evolution of complex structures is facilitated by the incremental open-ended evolution of a relatively converged ‘species.’ This is in contrast to the global search in a fixed parameter space performed by Genetic Algorithms (GAs) [3] used for engineering optimisation. Secondly we note that evolution, because it involves no modelling, abstraction or analysis, does not impose the same constraints on the circuit as conventional design. After relaxing redundant constraints (some of which are implicit in our preconceptions of what circuits should be like), evolution can explore the full space of possible designs: the resulting circuits can have a richer spatial and temporal organisation and can exploit the physical resources available far more than circuits produced by traditional methods.

By ‘hardware evolution’ we refer to the use of an evolutionary algorithm to determine the configuration of a many-times reconfigurable hardware device, such as a Field-Programmable Gate Array (FPGA). Configurations (*phenotypes*) are somehow encoded as a strings of symbols — typically bits — and a population of these genotypes is maintained. Fitnesses are assigned to individuals in this population by expressing each genotype as a real physical circuit configuration² and evaluating the circuit’s performance at the required task. This evaluation procedure can be automated to avoid any need for human intervention.

¹ A much shortened adaptation of [2].

² This is sometimes called *intrinsic* hardware evolution.

The individual genotypes are then inter-bred under a selection pressure, such that fitter individuals are more likely to survive and/or have offspring. Breeding is usually a stochastic process involving a *crossover* or *recombination* operator, which combines segments of the genotypes of two different parent genotype strings to produce children; and a *mutation* operator, which introduces random variations into the genotypes. Over the course of time, all being well, real physical circuits (typically as configurations of an FPGA) are obtained that are well adapted to the task at hand and the hardware resources available.

The remainder of this paper is structured in three parts. In the next section we describe the type of evolutionary process that is suitable for the evolution of complex hardware. Then the nature of evolution as a design process and of the evolved circuits themselves is investigated. Finally, the concepts are illustrated with a simple piece of hardware evolved as a real physical circuit to control a real autonomous mobile robot. This circuit (which was the first evolved hardware robot controller [4]) displays surprising abilities from a miniscule amount of hardware as a result of placing temporal constraints under evolutionary control and allowing exploitation of the physical characteristics of the real hardware.

2. EVOLUTION FOR COMPLEX SYSTEMS

The majority of GA work, both applications and theoretical analysis, refers to optimisation problems which can be seen as search problems in some high-dimensional search space of known (often enormous) size. The components to be optimised could be parameters which need to be set at appropriate real values; or they could be discrete values which need to be chosen. The well-defined finite dimensionality of the search space of these optimisation problems allows a choice of genotype coding such that a genotype of fixed length can encode any possible solution. For hardware design, this approach is appropriate when the optimal attributes of a predetermined number of components are to be found; also where the ordering of a given set of components needs to be determined.

GA theory has in general assumed such a fixed-dimensional search space. Typically the GA starts with a population of random points effectively spanning and coarsely sampling the whole search space. Successive rounds of selection, reproduction and mutation are intended to focus the population of sample points towards fitter regions of the space, homing in on an optimum or near-optimal region. Theorems such as the Schema Theorem, intended to show the circumstances under which GAs can be expected to produce the desired results, rely on these assumptions. One consequence of this approach has been

the primary reliance on recombination as the genetic operator, which combines information from different samples in order to move towards regions of expected higher fitness; mutation is typically treated as a background operator.

Standard GA theory only applies when there is a predetermined number of components to be used. However, there are at least two possible scenarios in which this is not the case. The first is incremental evolution: a sequence of increasingly complex tasks is posed, requiring the evolution in succession of ever more complex circuits. This is necessary when aiming to solve a problem that is too difficult for evolution to gain a foot-hold on if faced with it all at once. The second scenario is evolution for parsimony, where the number of components used is to be kept to a minimum.

If there is no predefined number of components in a structure to be designed, then an appropriate coding scheme used in a GA will, in general, code for different structures with genotypes of different lengths — which gives a search space of open-ended dimensionality. Evolutionary search can operate in domains of varying dimensionality — indeed evolution in the natural world has done just that. Relatively complex species, with lengthy genotypes, have evolved from simpler ancestors with smaller genotypes. GAs when applied to search spaces of varying dimensionality need a different framework from those used for standard optimisation problems. Species Adaptation Genetic Algorithms (SAGA) were developed as this framework.

2.1. Species Evolution

The conceptual framework of SAGA was introduced by Harvey in 1991 in order to try to understand the dynamics of a GA when genotype lengths are allowed to increase [5]. It was shown that progress through such a genotype space will only be feasible through relatively gradual increases in information in the genotype (typically, in genotype length). This is associated with the evolution of a genetically-converged *species* rather than global search. Such evolutionary search in the space of hardware designs would be from initially simple designs for simple tasks, towards more complex designs for more complex tasks; although in natural evolution there is no externally provided sense of direction, in artificial evolution this can be provided.

Throughout such artificial evolution, a species will be relatively fit, in the sense that most members of the population will be fitter than most of their neighbours in the fitness landscape. Evolutionary search can be thought of as searching around the current focus of a species (the current local optimum or hill in the fitness landscape) for neighbouring regions which are fitter (or in the case of neutral drift, not less fit) whilst being careful not to lose gains that were made in achieving the current *status quo*. Any higher hills in the landscape will tend to be found along ridges leading away from the current hill, and evolution should concentrate exploration along these ridges.

2.2. SAGA and Mutation Rates

GAs can be considered as a balance between exploration (of new untested regions) and exploitation (building on what has already been found to be good), here analysed in terms of mutation effects. Too few mutations means too little exploration away from the current hill. Too much mutation means that previously achieved gains are lost. In the context of molecular quasi-species [6, 7], it is shown that there is an ‘error catastrophe’: there is a maximum rate of mutation which allows a quasi-species of molecules to stay localised around its current optimum. Hence if mutation

rates can be adjusted, it would be a good idea to use a rate close to, but less than, this critical rate.

For an infinite asexual population, in a particular simplified fitness landscape representing molecular evolution, Eigen and Schuster show [6] that these forces just balance for a mutation rate $m = \ln(\sigma)/l$; here l is the genotype length and σ is the *superiority* parameter of the *master sequence* (the fittest member of the population) — the factor by which selection of this sequence exceeds the average selection of the rest of the local fitness landscape, and hence the rest of the population. The SAGA framework takes into account more complex fitness landscapes, including the element of ‘junk DNA’, which leads to a recommended rate of mutation of between 1 and 5 mutations per genotype. When applying such mutation rates, it is essential that the probability of mutation is applied independently at each position on the genotype.

2.3. Selection and Recombination

Selective forces need to be maintained at the same level throughout an evolutionary run, so as to balance mutational forces and maintain a similar degree of genetic convergence throughout. Basing selection directly on absolute fitness values does not achieve this, and some system based on ranking of the population must be used. This implies that the fittest member of the population has the same expected number of offspring whether it is far better than the rest, or only slightly better. One way to achieve an effect comparable to linear ranking in a steady state GA is through tournament selection. Rather than replacing the whole population by a similar number of offspring at each generation, only one new offspring at a time replaces a fatality in an otherwise unchanged population. Two parents for the offspring can each be chosen by picking the fittest of a randomly picked pair (the tournament), and the fatality chosen at random from the whole population; alternatively, the parents can be picked at random from the whole population, and the fatality selected as the loser of a tournament.

With a genetically converged population, sections of genotype that are swapped in recombination are likely to be fairly similar. With species evolution, recombination does not have the prime significance it has in standard GAs — asexual evolution is indeed feasible — but nevertheless it is a useful genetic operator.

3. THE NATURE OF EVOLVED CIRCUITS

In our formulation of evolvable hardware, individuals of the evolving population receive fitness scores according to their performance when instantiated as real physical pieces of electronics. Evolution proceeds by taking note of the overall behavioural effect of variations made to the real circuits; this is very different from conventional design techniques, which proceed by manipulating abstract models.

The use of abstract models simplifies design by allowing some aspects of reality to be ignored, but the properties of the real hardware that have been ‘abstracted away’ must be suppressed: they must not be allowed to influence the final behaviour of the designed circuit. For example, a designer engaged in digital design does not need to think about the analogue behaviour of the transistors, but considers them as ON/OFF switches. To allow circuits designed at this level of abstraction to work in reality, the transistors must always be kept in either the ON state or the OFF state except during short transient periods while they are switching between them. Steps must be taken to ensure that these transients do not influence the overall behaviour of

the system as predicted by the designer’s digital model. In synchronous design, this is done by compartmentalising the system into modules which only communicate on the ticking of a clock: the transient dynamics are localised within each module and die away before the module is allowed to influence the rest of the system. Thus, both the spatial organisation and the dynamical behaviour of the circuit are constrained in order to support the designer’s abstract model. This does not only apply to digital design, but to all design methodologies: none can proceed far with detailed physical descriptions of the dynamical behaviour of the elementary components (e.g. transistors) before abstraction is necessary.

Hardware evolution, by observing the consequences of variations made to the real hardware, avoids the need for design abstractions and the accompanying constraints. Our notion of the nature of electronic systems is heavily biased by our design methodologies and the constraints applied to facilitate their abstractions, so evolvable hardware demands a radical rethink of what electronic circuits can be. Both the spatial structure (modularity) and the temporal structure (synchronisation and the rôle of phase in general) need to be considered.

3.1. Spatial Structure

As well as to support abstract models, modularity arises in designs according to the way in which the problem was decomposed. Humans typically use some sort of “divide and conquer” strategy; whether the decomposition is a functional one or a behavioural one [8], the final structure arrived at usually has modules echoing that decomposition. The *evolutionary* process could also benefit from a kind of modularity, such that different phenotypic characteristics can be improved semi-independently by genetic mutations [9]. However, these ‘modules’ are not necessarily reflected in the phenotype circuit’s spatial or topological structure: for example, they could correspond to *basins of attraction* in state-space. These are currently open questions, and are intimately tied to the study of *morphogenesis*: the expression of genotype by a growth process to give rise to the phenotype.³ The important observation here is that any kind of modularity that *is* appropriate to evolution is very different to that arising in design by humans. Conventional notions of modularity should not be imposed upon an evolving circuit.

3.2. Temporal Structure

Real physical electronic circuits are continuous-time dynamical systems. They can display a broad range of dynamical behaviour, of which discrete-time systems, digital systems and even computational systems are but subsets. These subsets are much more amenable to design techniques than dynamical electronic systems in general, because the restrictions to the dynamics that each subset brings support design abstractions, as described above. Evolution does not require abstract models, so there is no need to constrain artificially the dynamics of the reconfigurable hardware being used.

In particular, there no longer needs to be an *enforced* method of controlling the phase (temporal co-ordination) in reconfigurable hardware originally intended to implement digital designs. The phase of the system does not have to be advanced in lock-step by a global clock, nor even the local phase-controlling mechanisms of asynchronous digital design methodologies imposed. The success of pulse-stream

³See [2] for more details.

neural networks [10, 11], where *analogue* operations are performed using *binary* pulse-density signals, gives a clue that allowing the system’s phase to unfold in real-time in a way useful to the problem at hand can add a powerful new dimension to electronic systems: time. Mead’s highly successful analogue neural VLSI devices (e.g. the ‘silicon retina’) [12], exploiting the continuous-time dynamics of networks of analogue components (with the transistors mostly operating in the sub-threshold region), show how profitable an excursion into the space of general dynamical electronic systems can be.

In some applications, dynamics on more than a single timescale are needed in an evolved circuit. For example, a real-time control system needs to behave on a timescale suited to the actuators (typically in the range milliseconds to seconds), while the underlying dynamics of the controller’s electronic components might be measured in nanoseconds. Being able to have different parts of a circuit behaving at different timescales can also be significant in other ways; indeed, *learning* can be thought of as a dynamic on a slower timescale than individual task-achieving behaviours. In a simulation experiment reported in [2], a 4kHz oscillator (slow) was evolved as a network of 68 logic gates with propagation delays between 1.0 and 5.0 nanoseconds (fast): evolution manipulated the timescales of the circuit’s overall behaviour without this having to be imposed from outside. This is in contrast to the usual techniques of using a clock, or large time-constant components — though it can still be sensible to provide these as resources that evolution can choose to utilise if useful. In the next section, these ideas are put into practice.

4. AN EVOLVED HARDWARE ROBOT CONTROL SYSTEM

This experiment takes a standard electronic architecture, removes some of the dynamical constraints used to make conventional design tractable, and subjects the resulting *dynamical* electronic system to hardware evolution.

The circuit to be evolved was the on-board controller for a two-wheeled autonomous mobile robot (height 63cm, diameter 46cm). The circuit’s task was to induce simple wall-avoidance behaviour into the robot when placed in a rectangular arena. For this scenario, the d.c. motors were not allowed to run in reverse and the robot’s only sensors were a pair of time-of-flight sonars rigidly mounted on the robot, one pointing left and the other right. The sonars fire simultaneously five times a second; when a sonar fires, its output changes from logic 0 to logic 1 and stays there until the first echo is sensed at its transducer, at which time its output returns to 0.

Consider the RAM based implementation of a finite-state machine (FSM) [13]. The RAM holds a look-up table for the next-state and output functions, and a clocked register holds the current state. The dynamics are highly constrained to deterministic state transitions which are independent of the precise characteristics of the hardware. This allows the FSM designer to operate at a high level of abstraction. Suppose now that as well as placing the contents of the RAM under evolutionary control, we allow it to be genetically determined whether each signal is latched according to the clock or whether it is allowed to vary asynchronously, in continuous time. Further, let the clock frequency be under genetic control. We then arrive at a very different type of system: call it a Dynamic State Machine (DSM). The DSM is endowed with a new rich range of possible dynamical behaviour, so that it can be directly coupled

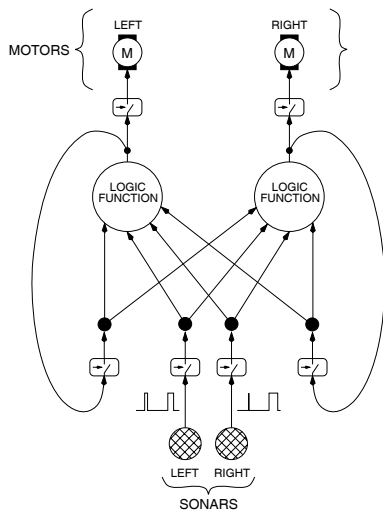



Figure 1. The evolvable Dynamic State Machine (real hardware). Each  is a ‘Genetic Latch’.



Figure 2. Behaviour of the evolved circuit.

to the pulsing sonars and the motors (which require pulse-modulating). The arrangement is shown in Figure 1, where a ‘Genetic Latch’ can either latch the signal according to the clock of evolved frequency, or can let the signal through asynchronously. Here, the 16 words \times 2 bits RAM is illustrated as implementing a pair of logic functions of four variables.

A designer would not be able to work with the DSM (which is no longer finite-state, but a continuous-time dynamical system). Metastability and glitches will be rife, and the detailed characteristics of the hardware (such as propagation delays and metastability constants) can be a crucial part of the way the system behaves. Nevertheless, evolution was able to exploit all of this to produce a robot controller exhibiting the excellent behaviour shown in the long exposure photograph of Figure 2. This is because all fitness evaluations happened with the individuals instantiated as real configurations of a hardware implementation of the DSM, driving the real motors. The reader is referred to [2] for full experimental details: a fairly conventional GA was used, but in accordance with SAGA theory.

One of the evolved DSMs was analysed, and was found to be going from sonar echo signals to motor pulses using only 32 bits of RAM and 3 flip-flops (excluding clock generation): highly efficient use of hardware resources, made possible by the absence of design constraints. The circuit

had subtle dynamics, consisting of a stochastic interplay between continuous-time and clocked components, as well as a tight coupling to the sensorimotor environment. The behaviour included a strategy to avoid being misled by specular sonar reflections, which were common.

5. CONCLUSION

Species Adaptation Genetic Algorithms (SAGA) and a respect for the natural physical dynamics of the hardware can allow complex, efficient circuits to be evolved. These circuits look very different to those produced by conventional design.

ACKNOWLEDGEMENTS

Special thanks to Dave Cliff. The research is funded by a D.Phil. scholarship from the School of Cognitive & Computing Sciences and by EPSRC.

REFERENCES

- [1] For an overview of the literature, see A.J. Hirst: Notes on the evolution of adaptive hardware. *To appear in: Proc. of 2nd Int. Conf. on Adaptive Computing in Engineering Design and Control (ACEDC96)*, Univ. of Plymouth UK, 26–28th March 1996. <http://kmi.open.ac.uk/~monty/evoladaphpaper.html>
- [2] Adrian Thompson, Inman Harvey, and Philip Husbands. Unconstrained evolution and hard consequences. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware*. Forthcoming: Springer-Verlag Lecture Notes in Computer Science, 1996.
- [3] David E. Goldberg. *Genetic Algorithms in Search, Optimisation & Machine Learning*. Addison Wesley, 1989.
- [4] Adrian Thompson. Evolving electronic robot controllers that exploit hardware resources. In F. Morán et al., editors, *Advances in Artificial Life: Proc. of the 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp640–656. LNAI 929, Springer-Verlag, 1995.
- [5] Inman Harvey. Species adaptation genetic algorithms: The basis for a continuing SAGA. In Varela and Bourgine, eds, *Toward a Practice of Autonomous Systems: Proc. of the 1st Eur. Conf. on Artificial Life*, pp346–354. MIT Press/Bradford Books, 1992.
- [6] M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self-Organization*. Springer-Verlag, 1979.
- [7] M. Eigen, J. McCaskill, and P. Schuster. Molecular quasi-species. *J. Phy. Chem.*, 92:6881–6891, 1988.
- [8] Rodney A. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–159, 1991.
- [9] Günter P. Wagner. Adaptation and the modular design of organisms. In F Morán et al., editors, *Advances in Artificial Life: Proc. of the 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp317–328. Springer-Verlag, 1995.
- [10] A. F. Murray et al. Pulsed silicon neural networks - following the biological leader. In Ramacher and Rückert, editors, *VLSI Design of Neural Networks*, pp103–123. Kluwer Academic Publishers, 1991.
- [11] Alan F. Murray. Analogue neural VLSI: Issues, trends and pulses. *Artificial Neural Networks*, (2):35–43, 1992.
- [12] Carver A. Mead. *Analog VLSI and Neural Systems*. Addison Wesley, 1989.
- [13] David J. Comer. *Digital Logic & State Machine Design*. Holt, Rinehart and Winston, 1984.