

# Unconstrained Evolution and Hard Consequences

CSRP 397

Adrian Thompson, Inman Harvey and  
Philip Husbands

December 1995

To appear in: E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware*, Springer-Verlag Lecture Notes in Computer Science, 1996. Presented at the workshop of the same name, EPFL, Lausanne, 2-3<sup>rd</sup> October 1995.

# Unconstrained Evolution and Hard Consequences

Adrian Thompson, Inman Harvey and Philip Husbands

School of Cognitive and Computing Sciences,  
University of Sussex, Brighton BN1 9QH, UK.  
E-mail: adrianth, inmanh, philh@cogs.susx.ac.uk

**Abstract.** Artificial evolution as a design methodology for hardware frees many of the simplifying constraints normally imposed to make design by humans tractable. However, this freedom comes at some cost, and a whole fresh set of issues must be considered. Standard genetic algorithms are not generally appropriate for hardware evolution when the number of components need not be predetermined. The use of simulations is problematic, and robustness in the presence of noise or hardware faults is important. We present theoretical arguments, and illustrate with a physical piece of hardware evolved in the real-world ('intrinsically evolved' hardware). A simple asynchronous digital circuit controls a real robot, using a minimal sensorimotor control system of 32 bits of RAM and a few flip-flops to co-ordinate sonar pulses and motor pulses with no further processing. This circuit is tolerant to single-stuck-at faults in the RAM. The methodology is applicable to many types of hardware, including Field-Programmable Gate Arrays (FPGA's).

## 1 Introduction

An evolutionary approach to hardware design makes possible the relaxation of several constraints which other more orthodox techniques require. Human designers conventionally need a prior rigorous analysis of the problem, and a decomposition of a complex system into separate parts of manageable size; using artificial evolution this is not necessary. Simplifying design constraints are often applied to hardware so as to make it behave in an easily analysable fashion — for instance, strict synchronisation to a global clock. This is no longer necessary with evolution, and such constraints can be relaxed.

However, this freedom comes at some cost; there are a whole new set of issues relating to evolution that must be considered, and many of these will be unfamiliar to those schooled in conventional design methods. Evolution of hardware systems often cannot be fitted into the constrained optimisation framework which standard genetic algorithms assume. This means that these algorithms need some crucial changes.

The main cost of an evolutionary approach is the large number of trials that are required. Adequate simulations may take time comparable to doing the trials for real, or may not be feasible; e.g. when vision in complex environments, or the modelling of detailed semiconductor physics is involved, as will be shown later.

Under many circumstances robustness in the presence of noise or hardware faults is a crucial factor, which can increase the number of trials needed. Noise is not always a problem, indeed it may have advantageous evolutionary effects.

We discuss the constraints that can be relaxed and the hard consequences that must be recognised, initially at a theoretical level. Then a real example of evolved hardware will be presented in the light of these discussions. A simple asynchronous digital circuit directly takes echo pulses from a pair of left/right sonars, and drives the two motors of a real robot, so that it exhibits a wall-avoidance behaviour in the real world. The complete sensorimotor control system (no pre- or post-processing) consists of just 32 bits of RAM and a few flip-flops, and is even tolerant to single-stuck-at faults in the RAM. The remarkable efficiency of this circuit can be attributed to the facts that it was evolved as a physical piece of hardware in the real world, and that many of the constraints on its dynamics were under evolutionary control. The rationale behind this experiment applies to many other kinds of system, including Field-Programmable Gate Arrays (FPGA's) [2].

The paper proceeds thus: Sections 2–11 discuss various aspects of artificial evolution. Sections 12–14 cover issues of noise, the relationship between simulation and reality, and fault tolerance. Sections 15–17 discuss the theory of Intrinsic Hardware Evolution. Sections 18 and 19 give a case study of a physical piece of hardware, intrinsically evolved in the real world as a robot controller. A final section summarises the discussion.

## 2 Evolution not Design

Human beings find it difficult to design complex systems, and the main heuristic used to make design easier is “Divide and Conquer”. The complex whole is decomposed into smaller semi-independent parts or modules, which can be tackled one at a time. For this to work the modules must have minimal interactions between them, to allow any one to be tackled independently of the others. However, there are many complex systems which either do not have any such decomposition, or do not obviously show how they should be carved up.

If, however, some objective fitness function can be derived for any given complex system which is intended to carry out some specific task, there is the possibility of automatic evolution of the system without explicit design. Natural evolution is the existence proof for the viability of this approach, given appropriate resources. Genetic Algorithms (GAs) [3] use ideas borrowed from evolution in order to solve problems in highly complex search spaces, and by suitably extending GAs they can be used to search through design space whilst evading the problems of decomposition faced by human designers.

The artificial evolution approach maintains a population of viable genotypes (chromosomes), coding for designs or architectures, which are inter-bred and mutated according to a selection pressure. This pressure is controlled by a task-oriented evaluation function: the better the system performs its task the more offspring it has in succeeding generations. Offspring inherit genetic material from

their parents. Rather than attempting to hand-design a system to perform a particular task or range of tasks well, the evolutionary approach should allow their gradual emergence.

### 3 GAs for Optimisation

Here we will discuss the framework within which GAs are typically used, before going on to suggest that for many hardware design problems a significantly different framework is needed.

The majority of published 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 (usually 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. In the former case the real values needed are usually coded to some desired degree of precision, so that they can be specified in binary form with a defined number of bits (although some evolutionary algorithms work directly with real values).

What such optimisation problems share is the well-defined finite dimensionality of the search space. This allows the choice of some genotype coding, such that a genotype (often binary) of fixed length can encode any potential solution within the space of possibilities. In the context of hardware design, this approach is appropriate where the optimal attributes for a predetermined number of components is to be found; also where the ordering of any given set of components needs to be determined.

GA theory has in general assumed such a fixed-dimensional search space. The optimisation problem has typically been seen as starting 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.

GAs can be considered a compromise between a *weak* and a *strong* form of search. While far stronger than random search, which typically is infeasible in large search spaces, GAs are not strong enough for one to be able to demonstrate conclusively that the global optimum has been reached. Once recombination has brought the population, over a number of generations, into a focused region of search space — i.e. it is genetically converged — then with only background rates of mutation further exploration cannot be expected, and search can be terminated. If this convergence happens too rapidly, it implies that only a sketchy sampling of the search space has been done, and any local optimum reached may

be far from the global optimum. Much GA analysis is directed towards setting up the GA parameters so as to avoid this premature convergence.

However, some problems — including perhaps most hardware design problems — do not fall into this convenient picture of a fixed-dimensional search space. If there is no predetermined number of components to be used in the design then standard GA theory will not apply.

## 4 GAs for Structure Evolution

There are at least two possible scenarios in which one might be uncertain initially how many components might be needed. The first case is incremental evolution: where a sequence of increasingly complex tasks is posed, requiring the evolution in succession of ever more complex hardware systems. The second is evolution for parsimony: where the number of components is to be reduced to a minimum.

If there is no predefined number of components in a structure to be designed by GA, then any encoding of potential solutions onto a genotype will use an amount of information which varies from case to case, given a fixed interpretation process. In other words, genotypes will have to be variable in length. But if the genotype is potentially unbounded in length, then the nature of the search space is such that it is impossible for an initial random population of finite size to effectively sample from all parts of it. Any finite population inevitably spans only a constricted region of the whole. Hence any GA search in an unbounded space must work with a relatively converged finite population from the very start. Possible ways in which systems of variable size can be encoded on a genotype will be discussed below, but this convergence property holds regardless of how the encoding is done.

Something very similar holds true if there is an upper bound to the number of components. Suppose that a maximum of 20 components is allowed for some hardware system, and an initial population contains sampling points to be evaluated with varying numbers of components between 0 and this maximum of 20. Since the subspace of designs with 10 or less components is of a radically different nature from that with 20 components, samples from the former subspace have no useful correlation in fitness with corresponding points from the latter (corresponding in the sense that the extra components have been added without altering the existing ones). But the underlying theory of standard GAs relies on there being some such correlation.

Evolutionary search can, however, 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, but a present-day animal should not be considered as a solution to a problem posed 4 billion years ago, with a search space of fixed dimensionality.

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.

## 5 Species Evolution

In artificial evolution problems of varying dimensionality one should expect to have a genetically converged population, in effect a *species*, at all times. This contrasts with optimisation problems, where convergence in a GA signals the end of the search process. Using the example given above, where a hardware design being evolved could potentially have any number of components up to 20, during any single generation one should expect all the members of the population to have the same or a very similar number of components, for instance 10. What counts as *similarity* will be qualified later.

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 [4]. It was shown, using concepts of epistasis and fitness landscapes drawn from theoretical biology [5], that progress through such a genotype space will only be feasible through relatively gradual increases in information in the genotype (typically, in genotype length). A general trend towards increase in length is associated with the evolution of a *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 for neighbouring regions which are fitter (or in the case of neutral drift, not less fit) while being careful not to lose gains that were made in achieving the current *status quo*. In the absence of any mutation (or change-length) genetic operator, selection will concentrate the population at the current best. The smallest amount of mutation will hill-climb this current best to a local optimum. As mutation rates increase, the population will spread out around this local optimum, searching the neighbourhood; but if mutation rates become too high then the population will disperse completely, losing the hill-top, and the search will become random. If an ideal balance is achieved between selective forces and those of mutation (as modified by recombination), then some elements of the population can crawl down the hill far enough to reach a ridge of relatively high selective values. As discussed in [6], this results in a significant proportion of the population working their way along this ridge under selection, and making possible the reaching of outliers ever further in Hamming-distance in that particular direction from the current fittest. The term ‘ridge’ is used here to fit in with intuitive notions of fitness landscapes; in fact in high-dimensional search spaces such ridges may form complex neutral networks, percolating long distances through genotype space.

If any such outliers reach a second hill that climbs away from the ridge, then parts of the population can climb this hill. Depending on the difference in fitness and the spread of the population, it will either move *en masse* to the new hill as a better local optimum, or share itself across both of them.

So in a SAGA setup of evolution of a converged species, we want to encourage through the genetic operators such exploration along ridges to new hills, subject to the constraint that we do not want to lose track of the current hill. Eigen and co-workers use the concept of a *quasi-species* to refer to a similar genetically converged population in the study of early RNA evolution. To quote from [6]:

In conventional natural selection theory, advantageous mutations drove the evolutionary process. The neutral theory introduced selectively neutral mutants, in addition to the advantageous ones, which contribute to evolution through random drift. The concept of quasi-species shows that much weight is attributed to those slightly deleterious mutants that are situated along high ridges in the value landscape. They guide populations toward the peaks of high selective values.

## 6 SAGA and Mutation Rates

Although progress of a species through a fitness landscape is not discussed in the standard GA literature, in theoretical biology there is relevant work in the related field of molecular quasi-species [7, 6]. In particular, analysis of the ‘error catastrophe’ shows that, subject to certain conditions, there is a maximum rate of mutation which allows a quasi-species of molecules to stay localised around its current optimum. Selection and mutation are opposing forces, the former tending to increase numbers of the fittest members of the population, while the latter tends to drag offspring down in fitness away from any local optimum. A zero mutation rate allows for no further local search beyond the current species, and other things being equal increased mutation rates will increase the rate of evolution. Hence if mutation rates can be adjusted, it would be a good idea to use a rate close to but less than any critical rate which causes the species to fall apart. A further possibility, in the spirit of simulated annealing, is to temporarily allow the rate to go *slightly above* the critical rate — to allow exploration — and then cut it back again to consolidate any gains thus made.

For an infinite asexual population, in the particular context of molecular evolution, Eigen and Schuster show [7] that these forces just balance for a mutation rate

$$m = \frac{\ln \sigma}{l}$$

where  $l$  is the genotype length and  $\sigma$  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 diagrams they show for the very sharp cutoff at the critical rate refer to a fitness landscape with a single ‘needle’ peak for the master sequence, taking all the rest of the population to be equally (un-)fit; where the hill slopes more gently from the master sequence, the cutoff is less abrupt. For typical values of  $\sigma$  between 2 and 20, the upper limit of mutation before a quasi-species ‘loses its grip’ on the current hill would be between  $0.7/l$  and  $3/l$ . For finite population size, there is some reduction

in this critical mutation rate (the ‘error threshold’) [8], but for genotypes of length order 100, and populations of size order 100, the error threshold will be extremely close to that for an infinite population.

Since it is the natural logarithm of  $\sigma$  which enters into the equation for  $m$ , variations in  $\sigma$  of an order of magnitude do not affect  $\ln(\sigma)$  (and hence the error threshold) as significantly as variations in genotype length. In conventional GAs, choice of mutation rates tends to be a low figure, typically 0.01 or 0.001 per bit as a background operator, decided upon without regard to the genotype length. The SAGA framework means that mutation rates of the order of 1 per genotype are required when using linear rank selection or tournament selection as discussed below, subject to some qualifications concerning recombination and ‘junk DNA’. These qualifications tend to increase the recommended rate to somewhere in the range 1 to 5 mutations per genotype, the idiosyncratic nature of fitness landscapes for different problems making it difficult to be more specific.

When applying such mutation rates in a GA, it is essential that the probability of mutation is applied independently at each locus on the genotype. This gives a binomial distribution (approximating a Poisson distribution for long genotypes) for the number of mutations per string, so that genotypes with an expected  $m$  mutations have this as the average value with a wide variance (including the possibility of zero mutations).

## 7 Neutral Sequences and Drift

Mutations in a genotype encoding a fit phenotype are often deleterious, and occasionally advantageous. There is a third possibility, that a mutation is neutral and leaves the fitness unchanged.

Neutral mutations can in turn be subdivided into two kinds, with a rather grey area between them. They can be in parts of ‘junk DNA’, such that the decoding of the genotype ignores the values in that part. In this case it is only the functional part of the genotype (the part which is capable of causing some difference in the fitness) that counts towards effective genotype length when deciding upon mutation rates. For example, if a genotype of length 1000 is 90% junk, then a mutation rate set at the rate of one per effective genotype length should be implemented at the rate of 1/100 per locus, rather than 1/1000. It is often difficult to estimate what proportion of a genotype is junk, however, as this shades into the second class of neutral mutation.

This second type of mutation may leave the phenotype unchanged, yet open the possibility of a further mutation making some difference. As a simple example, a binary genotype with two loci, whose fitness is given by the logical AND of the alleles at each locus, retains a fitness of 0 during mutation from **00** to **01**; yet this opens up the possibility of a further single point mutation reaching **11**, with a fitness of 1 which was not achievable from the starting point. Such neutral mutations can in a high-dimensional space allow extended neutral paths which can percolate through vast areas of sequence space. Neutral drift of a population through such pathways means that it is much more difficult to get stuck on a

local optimum than one's intuition based on 3-D landscapes might lead one to think. In addition, the percolation of such paths through sequence space tends to mean that it does not matter too much where in sequence space a converged population starts; under many circumstances it is possible to reach all possible fit regions from most starting points.

The SAGA selection and mutation rates encourage just such exploration through neutral drift in sequence space.

## 8 Selection

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. Truncation selection (reproducing only from a top slice of the population) is one way of achieving this, but generally is too severe in restricting exploration by the less fit mutants. Less severe methods are recommended such as linear ranking; for instance giving the top ranker twice the average expected number of offspring, and reducing this amount linearly as one goes down the ranks, towards zero.

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.

Elitism is often advocated in GAs when used for optimisation. This is the requirement that the current fittest member of the population is never deleted to make way for another that is less fit. In real world applications such as hardware design, however, evaluations are likely to be noisy. Since in this case one can never be certain which is the fittest, elitism cannot be relied upon.

## 9 Recombination

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.

There are two rôles recombination has which are opposite sides of the same coin. On the one hand, it allows two fortunate mutations which happen to have

occurred independently in two different lineages within the population to be combined into one which has both: something not possible with asexual reproduction. On the other hand, it allows parents with a detrimental mutation to produce an offspring which does not have it: also impossible asexually, in the absence of highly improbable back-mutations. This latter effect in general allows higher mutation rates to be used with recombination than were suggested above for asexual populations, thus promoting exploration without risking loss of a currently achieved local optimum.

Recombination is particularly powerful when combined with a distributed GA. Here each member of the population is allocated a different position in some notional geographical space, often a two-dimensional toroidal grid. Recombination between individuals is only allowed for pairs within a certain distance of each other on this grid, which thus comprises a number of overlapping neighbourhoods. This combines the virtues of small and large populations; small interrelated local populations allow through random drift more extended search through genotype space, but the overlapping nature of such localities means that any improvement found percolates through the whole population.

Recombination can run into problems with genotypes of differing lengths; it may not be clear, given a crossover or recombination point in one parent, where a corresponding crossover should be made in the other parent. Whatever system is used should ensure that homologous segments of the genotype are swapped. Often this may need domain-specific GA program code; a general algorithm for long binary strings is given in [9].

## 10 Change in Genotype Length

If systems of different complexity are encoded by genotypes of different length, then the genetic operators must allow changes in genotype length to take place in the process of going from parent to offspring. How this is done will necessarily depend to a large extent on the form of genotype encoding. One lesson spelt out in [4] is that any genetic operator which allows a change in genotype length should be restricted to small changes only — to be precise, to changes which in general can be expected to produce a small phenotypic change. This last qualification allows for genetic operators such as gene duplication where the encoding used means that such duplications are neutral. Neutral gene duplication, followed by mutations in one of the copies, is a potentially powerful method for creating variants on useful substructures.

## 11 Morphogenesis

The preceding discussion has been deliberately general insofar as very few constraints have been laid down as to how the genotype should encode for the system to be evolved. To some extent such an encoding is always domain-specific, but nevertheless there are some general rules that come into play as a system grows more complex.

With simple designs where there are no symmetries or repetitions to be expected, then the straightforward method is for the genotype to encode in sequence the type and parameters of each component part, together with the interconnections between such parts. However, as systems become more complex, then symmetries and repetitions can be expected to play a significant rôle in many circumstances. For instance, when designing a control system for a robot with bilateral symmetry, then there may well be good reason to expect this bilateral symmetry to be reflected to some extent in the controller. If designing an artificial retina, then repetitions of similar local structure across a 2-D array can be expected.

If the genotype constitutes in effect a linear description of each component of the system in turn, with for instance the left and right halves separately so described, then bilateral symmetry can only be achieved by separate independent evolution of each half of the genotype towards the same target. In a stochastic process such as evolution this is improbable and difficult; however it could be achieved relatively simply if the genotype described just one half, together with a routine which ‘called’ the description twice with appropriate parameters, in the sense that a program can call a sub-routine. In the case of multiple repetitions as in a retinal array, such a process becomes even more efficient.

Earlier it was suggested that evolution was a method of avoiding the constraints of human design, which seems to require the decomposition of a system into semi-independent modules. The repetitions and symmetries now being discussed differ from such human decomposition in two critical ways. Firstly, any such decomposition can emerge from evolutionary choice, rather than being pre-determined by fallible human prejudice; and secondly, such repeated modules can be intimately connected with each other, as with neighbourhood relationships on a retina, and need not be semi-independent.

In the natural world the genotype does not constitute a description of the organism. Instead it acts as a constraint on the way in which a multi-cellular organism develops from a single cell, in such a fashion that symmetries and repetitions can naturally emerge. Attempts to replicate such emergence in artificial evolution are currently *ad hoc* and domain-specific.

## 12 The Use of Noise

Stochastic noise can be actually be advantageous to the evolutionary process. Such noise alters the behaviour of the hardware, as compared to its behaviour in some idealised non-noisy world; hence it alters the fitness of this hardware at the task on which it is being evaluated. These alterations are in general such as to blur the fitness landscape, to decrease the difference in behaviour (and fitness) between two pieces of hardware which are neighbours in genotype space. Rugged regions in the fitness landscape tend to get blurred into more rolling hills, on which the evolutionary process finds progression towards higher fitness much more easy.

This effect is related distantly to that of stochastic resonance. The arguments are similar to those for the ‘Baldwin effect’ [10]. Practical examples of this phenomenon can be seen in [11, 12]. The rôle of noise in smoothing the transition from simulations to reality is under investigation [12].

### 13 The Use of Simulation

Artificial evolution requires the evaluation of members of a population over the course of many generations. Virtually all the time and expense is in performing these evaluations rather than in the genetic operations; optimising these latter operations has little practical use. The number of evaluations may be reduced by setting up the GA appropriately. Each individual evaluation should also not be unnecessarily long.

Simulations are often seen as attractive, offering the possibility of performing evaluations in faster than real time. Such simulations would need to be validated by testing evolved architectures in realised form at regular intervals. However there are a number of possible problems, depending on what precisely is to be simulated. Consider in turn the cases where simulations are of just the hardware, of both hardware and environment, and of just the environment.

Simulation of the hardware alone might be attempted if appropriate reconfigurable hardware was not available — simulations are likely to run more slowly than the real hardware. This is ‘extrinsic’ evolution, as opposed to ‘intrinsic’ evolution to be discussed later. Often detailed simulations of physical properties of hardware are too computationally expensive to be practical; evolution can then only be effective with the real hardware.

When both the hardware and the environment are to be simulated, it is frequently the latter that is much more complex. Often such simulations are simply not practical. For instance, when evolving control systems for visually guided robots [11], the visual world of a robot takes so much computing power to adequately simulate that testing in the real world is simpler and faster.

Consider finally the use of real hardware in a simulated environment. It has been suggested by de Garis [13] that where a hardware device such as a robot controller must be evaluated within some environment, then an environment simulation could be implemented in electronics situated next to the evolving hardware control system on a VLSI chip. It is suggested that this will allow the speeding up of each evaluation by perhaps many orders of magnitude. There are two serious doubts to be raised about this proposal.

The first is scepticism about the complexity of the simulation. For many real tasks (and that of a robot controller in a human environment is one example) the complexity of those crucial aspects of the environment which must be adequately simulated makes it infeasible; e.g., when the environment includes other entities of a complexity and speed comparable to that of the hardware itself.

A second doubt arises when an adequate evaluator circuit running faster than real-time *can* be built (perhaps possible for simpler situations than robotics). The same hardware that has been evolved within a high-speed environment must then

adjust to cope with normal timescales in the real world. This could be done with clocked hardware, by simply adjusting the clock-rate. However, the imposition of adjustable clocking on a piece of hardware is a severe constraint, limiting the range of dynamics available. Later sections will show the benefit of *removing* such constraints. More subtle ways of controlling timescales may be possible, but only those aspects of the dynamics which *are* controlled may be exploited during accelerated evolution. Thus, for this technique to be useful, any benefits from the increased rate of evolution must outweigh the costs of constraints. Note that *real-time* simulations are not prone to this difficulty.

## 14 SAGA and Fault Tolerance

When considering the mutation rate for SAGA (Section 6), we saw that there is an ‘error catastrophe’ mutation rate above which the species disintegrates and loses its local fitness peak. For practical SAGA mutation rates, the population will never completely converge upon the single locally-most-fit sequence, but will converge *around* it, in an equilibrium of being driven away by mutation and pulled back by selection. Most of the individuals will not have the locally-most-fit genotype, but will be a small number of mutations away in genotype space. This has the effect that selection is not able to hold the population at an isolated ‘needle in the haystack’ peak as well as it can hold it on a ‘smoother’ hillside, where fitness falls away gradually with increasing Hamming distance from the peak. In fact, Thompson [14] adapts one of Eigen & Schuster’s experiments [15] to SAGA, illustrating that an initially completely converged population can abandon an isolated peak in favour of a *less fit* smoother one.

The result is that, under certain conditions, the evolving species tends to move to a high fitness region of the landscape at which mutations have only a gradual effect on fitness on average (if such a region exists). In particular, most *single* mutations will tend not to decrease fitness dramatically, although a few critical ones might. What does this have to do with fault tolerance? Take an example: say that part of the genotype directly encodes the connectivity between some components of the phenotype circuit, with a **1** indicating a connection and a **0** no connection. The preceding argument predicts that single mutations will tend to have only small effects on fitness, implying that the performance of the circuit will tend not to be drastically degraded by the creation or removal of connections. Therefore the evolved circuits will tend to be tolerant to hardware faults that cause connections to be broken or spuriously created.

This general phenomenon applies whenever the genetic encoding is such that a genetic mutation has the same effect on the phenotype as would a certain type of fault; there will be a tendency for the evolved systems to be less sensitive to that type of fault than equivalent systems produced by non-evolutionary means. Current research aims to characterise the effect using the NK model of fitness landscapes [16] over the full range of possible SAGA parameters and selection schemes: for some settings, evolved individuals have been seen to be 10% less sensitive to single mutations than equivalent individuals found through

exhaustive search, on average. Thompson [14] deals with the evolution of fault tolerance in greater detail than here, also suggesting that the use of a co-evolving antagonistic population of emulated faults could force tolerance to a large set of faults in cases where the above technique is inapplicable or insufficient.

## 15 The Relationship Between Intrinsic Hardware Evolution and Conventional Design Techniques

*Intrinsic*<sup>1</sup> evolution of hardware means that the 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; this is manifest in the use of a global clock in synchronous design, and the more local co-ordination mechanisms of asynchronous design methodologies. Hence, the use of an abstract model at the design stage imposes constraints on what circuits can be produced, in order to ensure that the model is valid.

Intrinsic 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.

### 15.1 Unconstrained Spatial Structure

We saw above that with digital design, care must be taken to prevent switching transients (a feature absent from the designer’s model) from affecting the system’s overall behaviour. This is done by making sure that one part of the system does not influence the rest until any transient dynamics have died down

---

<sup>1</sup> This term is due to Hugo de Garis.

and a steady state is reached (all transistors stable in the ON or OFF states). Usually, this means that the circuit is broken into modules, the internal transient dynamics of which are hidden from each-other. (Here, the word ‘module’ is used in a general sense to mean a cohesive sub-assembly within a larger system.) The spatial or topological structure of the circuit has thus been constrained to facilitate a design abstraction.

Even if reconfigurable hardware intended for use by digital designers is used (eg. most current FPGAs) then design abstractions, such as the digital model, are not required for intrinsic hardware evolution. Is modularity, then, a designer’s constraint that can be abandoned, or is it necessary or useful for all complex systems whether designed or evolved? Are the kinds of modules appropriate for an evolving circuit different from those used by a designer?

These questions are currently difficult to answer fully. Certainly, we have seen that evolution does not need modular structures to support abstract designer’s models, because intrinsic evolvable hardware (hereafter ‘intrinsic EHW’) does not use such models. However, the modularity of a system can also be caused by the nature of the problem-solving or adaptive process that derived it. Humans typically use some sort of “divide and conquer” strategy, by which the problem is successively decomposed. Whether the decomposition is a functional one or a behavioural one [17], the final structure arrived at usually has modules corresponding to that decomposition. Wagner [18] argues that the *evolutionary* process also requires a kind of modularity: that there should be an “independent genetic representation of functionally distinct character complexes.” The idea is that such a genotype-phenotype mapping prevents small mutational variations applied at one point from having large-scale ramifications throughout the whole phenotype, so that parts of it can be improved semi-independently. However, it is not clear to what extent this consideration necessarily implies modules in the structure of an EHW circuit, because the “distinct character complexes” of the phenotype are components of the *behaviour* of the circuit, not of its physical organisation<sup>2</sup>. A related rôle for modules in an evolved circuit was given above (Section 11) when considering the part which repeated structures and symmetries can play in the morphogenesis and evolution of complex systems.

It seems that if modular circuits are desirable in EHW, it is because modularity may aid the evolution of complex systems, rather than because modularity is essential to the operation of a complex electronic circuit. For this reason, the *kinds* of modules appropriate to EHW will be tuned to the characteristics of the evolutionary process (particularly the genetic encoding and morphogenetic development) and the way in which this interacts with the detailed properties of the particular type of reconfigurable hardware being used. It remains to be seen what such modules may look like, but the important message is that they may be radically different from what is seen in circuits produced by traditional design methods.

---

<sup>2</sup> For example, a character complex might inhere in a particular *basin of attraction* of the system, which could be a property of the *whole* circuit.

## 15.2 Unconstrained 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. Intrinsic EHW 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 neural networks [19, 20], 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 (eg. the 'silicon retina') [21], 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 EHW 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.

There are several ways in which high-speed electronic components can give rise to much slower behaviour:

- The phase can be governed by one or more external signals, a digital clock being the prime example.
- Large time-constant resources can be provided. Large capacitors or inductors cannot be made in VLSI, so either external components can be made available, or special techniques can be used to make the most of smaller on-chip components [22].
- The high-speed components can somehow be assembled to give rise to slower dynamics, without explicitly providing large time-constant resources or slow-speed clocks.

Is the last of these possibilities feasible in an EHW framework? To find out, a simulation experiment was performed to see if a network of high-speed logic gates could be evolved to oscillate at a much slower timescale. The number of

logic nodes available was fixed at 100, and the genotype determined which of the boolean functions of Table 1(a) was instantiated by each node, and how the nodes were connected. The nodes were analogous to the reconfigurable logic blocks of an FPGA, but an input could be connected to the output of any node without restriction. The linear bit-string genotype consisted of 101 segments (numbered 0..100 from left to right), each of which directly coded for the function of a node and the sources of its inputs, as shown in Table 1(b). (Node 0 was a special ‘ground’ node, the output of which was always clamped at logic zero.) This encoding is based on that used in [23]. The source of each input was specified by counting forwards/backwards along the genotype (according to the ‘Direction’ bit) a certain number of segments (given by the ‘Length’ field), either starting from one end of the string, or starting from the current segment (dictated by the ‘Addressing Mode’ bit). When counting along the genotype, if one end was reached, then counting continued from the other.

Name	Symbol
BUFFER	
NOT	
AND	
OR	
XOR	
NAND	
NOR	
NOT-XOR	

(a)

BITS	MEANING
0-4	Junk
5-7	Node Function
	POINTER TO FIRST INPUT
8	Direction
9	Addressing Mode
10-15	Length
	POINTER TO SECOND INPUT
16	Direction
17	Addressing Mode
18-23	Length

(b)

**Table 1.** (a) Node functions, (b) Genotype segment for one node.

At the start of the experiment, each node was assigned a real-valued propagation delay, selected uniformly randomly from the range 1.0 to 5.0 nanoseconds, and held to double precision accuracy. These delays were to be the input-output delays of the nodes during the entire experiment, no matter which functions the nodes performed. There were no delays on the interconnections. To commence a simulation of a network’s behaviour, all of the outputs were set to logic zero. From that moment onwards, a standard asynchronous event-based logic simulation was performed [24], with real-valued time being held to double precision accuracy. An equivalent time-slicing simulation would have had a time-slice of  $10^{-24}$  seconds, so the underlying synchrony of the simulating computer was only manifest at a time-scale 15 orders of magnitude smaller than the node delays, allowing the *asynchronous* dynamics of the network to be seen in the simulation. A low-pass filter mechanism meant that pulses shorter than 0.5ns never happened anywhere in the network.

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

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

For the purpose of this equation, transitions were also assumed to occur at the very beginning and end of the trial, which lasted for 10ms of simulated time. The fitness was simply the reciprocal of the average error. Networks that oscillated far too quickly or far too slowly (or not at all) had their evaluations aborted after less time than this, as soon as a good estimate of their fitness had been formed. The genetic algorithm used was a conventional generational one [3], with elitism and linear rank-based selection. At each breeding cycle, the 5 least fit of the 30 individuals were killed off, and the 25 remaining individuals were ranked according to fitness, the fittest receiving a fecundity rating of 20.0, and the least fit a fecundity of 1.0. The linear function of rank defined by these end points determined the fecundity of those in-between. The fittest individual was copied once without mutation into the next generation, which was then filled by selecting individuals with probability proportional to their fecundity, with single-point crossover probability 0.7 and mutation rate  $6.0 \times 10^{-4}$  per bit.<sup>3</sup>

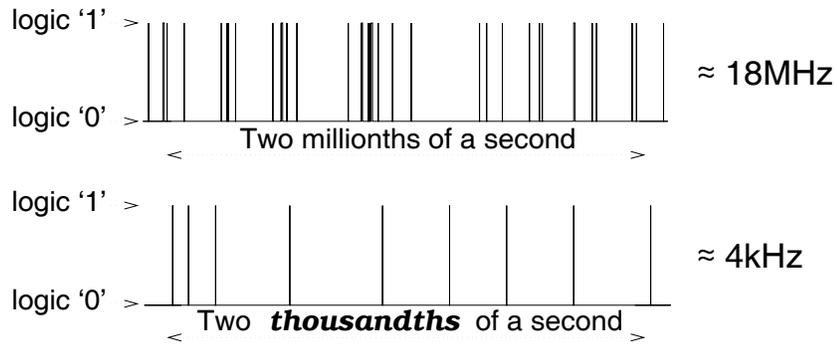
Fig. 1 shows that the output of the best individual in the 40<sup>th</sup> generation (Fig. 2) was approximately  $4\frac{1}{2}$  thousand times slower than the best of the random initial population, and was six orders of magnitude slower than the propagation delays of the nodes. In fact, fitness was still rising at generation 40 when the experiment was stopped because of the excessive processor time needed to simulate this kind of network. This result suggests that it *is* possible for evolution to arrange for a network of high-speed components to generate much slower behaviour, without having to provide explicit ‘slowing-down’ resources with large time constants, and without the need for a clock (though these could still be useful).

The evolved oscillators produced spike trains rather than the desired square wave. Probing internal nodes indicated that this was because *beating* between spike trains of slightly different frequencies was being used to generate a much lower frequency; beating only works for spikes, not for square waves.<sup>4</sup> This does not mean that the task was easy: it is difficult for beats to reduce the frequency by the massive factor required and yet produce an output as regular as that seen in Fig. 1.

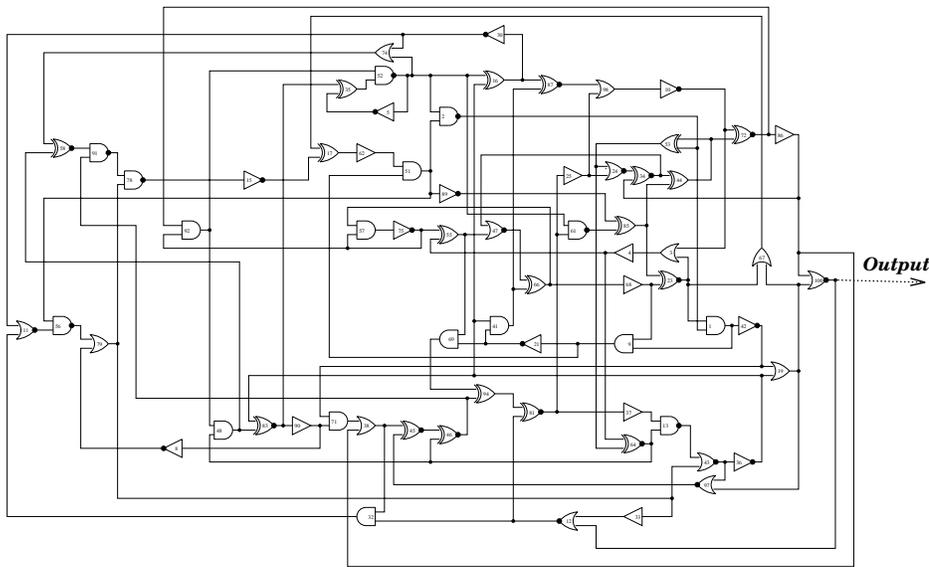
---

<sup>3</sup> This per-bit mutation rate was crudely derived by trial and error, but turns out to be on the order of one significant mutation per genotype for the final evolved circuit, in line with SAGA theory.

<sup>4</sup> Of course, the output spike train could be converted into a square wave by passing it through a toggling flip-flop, though this did not evolve here.



**Fig. 1.** Output of the evolving oscillator. (Top) Best of the initial random population of 30 individuals, (Bottom) best of generation 40. Note the different time axes. A visible line is drawn for every output spike, and in the lower picture each line represents a single spike.



**Fig. 2.** The evolved 4kHz oscillator (unconnected gates removed, leaving the 68 shown).

The simulation was quite an unrealistic model of the evolution of the configuration of a real FPGA. No analogue effects were modelled apart from time, but they would be a big part of the way a real chip would behave. Nevertheless, the result lends strength to the concept of evolving unconstrained dynamical systems, because beating evolved: a highly effective solution which is essentially a continuous-time phenomenon. Beating does not just occur at one node but

throughout the network, which does not have any significant modules<sup>5</sup>.

## 16 Exploiting the Hardware versus Tolerance to Device Variations

In the evolved oscillator experiment of the previous section, each node had a slightly different input  $\Rightarrow$  output time delay, crudely modelling the propagation delays of the reconfigurable blocks of an FPGA. The delays were initially randomly chosen but then held fixed during the experiment. If the delays of the final evolved circuit were *re-randomised* then the behaviour was totally destroyed, and the circuit was no better than a randomly generated one: it relied on the particular time delays present during its evolution. Extrapolating to real intrinsic hardware evolution, it can be expected that all of the detailed physics of the hardware will be brought to bear on the problem at hand: time delays, parasitic capacitances, cross-talk, meta-stability constants and other low-level characteristics might all be used in generating the evolved behaviour.

The exploitation of all of the hardware's physical properties must be traded against sensitivity to variations in them. Some tolerance is essential because of the inevitable changes over time due to fluctuations in the temperature or power supply, for example, or to the various on-chip ageing phenomena. In addition, if an evolved design is to be implemented on more than one device (as in a commercial application), then it must not be affected by properties that vary from chip to chip. The required tolerance can be evolved by making sure that the properties in question actually do vary while the fitnesses are being evaluated. This might involve purposely varying the temperature or power supply during evaluations and/or carrying out multiple evaluations on different (nominally identical) reconfigurable devices. An alternative to evaluating over several separate devices is to use the same reconfigurable hardware to instantiate the circuit in different ways: translating or rotating it on an FPGA for example. Extending that idea, a genetic encoding could be used which forces the use of repeated structures, so that each structure must cope with the characteristics of all of the places in which it occurs. Another alternative would be to use further adaptation each time the evolved circuit is transferred from one reconfigurable device to another.

The introduction of a *fault* can be seen as an extreme form of variation in the device's properties: Section 19 will describe a real application of the 'further adaptation' approach to coping with a fault. It may also be possible to use the evolutionary fault tolerance mechanisms described in Section 14 to give tolerance to normal device variations.

Even when forced to produce a circuit with tolerance to some range of variations, there is still room for intrinsic EHW to exploit detailed hardware characteristics much more than conventional design does. Traditional design methods

---

<sup>5</sup> A heuristic graph partitioning algorithm was used to search for non-trivial cohesive sub-assemblies, but none were found.

cannot proceed far with precise descriptions of the physics of the individual components (eg. transistors) before abstraction and modularisation need to be invoked to make the problem tractable, as discussed above. The ‘try it and see’ opportunistic nature of intrinsic EHW is not subject to this difficulty, so the detailed behaviours of the components can be integrated usefully to give rise to the required overall performance. This paper largely concentrates on the use of digital reconfigurable devices because these are currently available off-the-shelf, but we can now see that the advantages of intrinsic EHW over conventional design are even greater for analogue systems. Analogue FPGAs are being developed [25] and will be a fruitful avenue for EHW research in the future.

## 17 The Danger of Inheriting Inappropriate Constraints from Natural Evolution

Evolvable Hardware is a combination of electronics and evolution. We have discussed the error of adhering too closely to the conventional principles of electronics, but there are also potential pitfalls in blindly applying ideas from natural evolution.

Consider biological neural networks. Compared to electronics, the neuron response and signal propagation times are extremely slow. On the other hand, there is very high connectivity in three dimensions, contrasting with the highly restricted planar wiring in VLSI. The two media — biological cell based and silicon VLSI based — provide very different resources. A structure evolved to exploit the former may not efficiently utilise the latter. It may be possible to evolve parallel distributed architectures better tailored to the opportunities provided by VLSI than models of biological neural networks are. Such an architecture might use the high speed of VLSI to compensate for limited connectivity in a more sophisticated way than the multiplexing schemes commonly seen in VLSI implementations of neural nets. Hence it would be unwise to rigidly limit EHW to a neuro-mimetic structure when intrinsic EHW can offer a more unconstrained exploitation of hardware resources. For engineering purposes, ‘VLSI-plausible’ architectures are required, not ‘biologically-plausible’ ones.

In the same way that the architecture of natural nervous systems evolved to be suited to the restrictions and opportunities of biology, so did the process of natural evolution itself adapt to the resources available (“the evolution of evolvability”). The large timescale, highly parallel, distributed co-evolution found in nature is somewhat different that possible in present-day implementations of artificial evolution. It is thus justifiable to use biologically-implausible mechanisms where these are effective, for example in the setting of the mutation rate or in the morphogenesis process. The aim is to arrive at an implementation of artificial evolution that is inspired by nature, but suited to the facilities available.

We have now proposed a synthesis of genetic algorithms, natural evolution and electronics that adapts each in the formation of a new field: Intrinsic Evolvable Hardware. The next section begins to put some of the ideas into practice.

## 18 Case Study: An evolved hardware sensorimotor control structure

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 intrinsic hardware evolution. The result was the first evolved hardware control system for a real robot, reported in [26].

The EHW circuit was the on-board controller for the robot shown in Fig. 3. This two-wheeled autonomous mobile robot has a diameter of 46cm, a height of 63cm, and was required to display simple wall-avoidance behaviour in an empty 2.9m×4.2m 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**.

Conventional electronic design would tackle the control problem along the following lines: For each sonar, a timer would measure the length of its output pulses — and thus the time of flight of the sound — giving an indication of the range to the nearest object on that side of the robot. These timers would provide binary-coded representations of the two times of flight to a central controller. The central controller would be a hardware implementation of a finite-state machine (FSM), with the next-state and output functions designed so that it computes a binary representation of the appropriate motor speed for each wheel. For each wheel, a pulse-width modulator would take the binary representation of motor speed from the central controller and vary the mark:space ratio of pulses sent to the motor accordingly.

It would be possible to evolve the central controller FSM as intrinsic EHW by implementing the next-state and output functions as look-up tables held in an off-the-shelf random access memory (RAM) chip.<sup>6</sup> The FSM would then be specified by the bits held in the RAM, which could be reconfigured under the control of each individual’s genotype in turn. There would be no benefit in evolving this architecture as hardware, however, because the electronics is constrained to behave in accordance with the FSM design abstraction: all of the signals are synchronised to a global clock to give clean, deterministic state-transition behaviour as predicted by the model. Consequently, the hardware would behave identically to a software implementation of the same FSM.

What if the constraint of synchronisation of all signals is relaxed and placed under evolutionary control? Although superficially similar to the FSM implementation, the result (shown in Fig. 4), is a machine of a fundamentally different nature. Not only is the global clock frequency placed under genetic control, but the choice of whether each signal is synchronised (latched) by the clock or whether it is asynchronous is also genetically determined. These relaxations of

---

<sup>6</sup> This is the well known ‘Direct Addressed ROM’ implementation of an FSM [27].

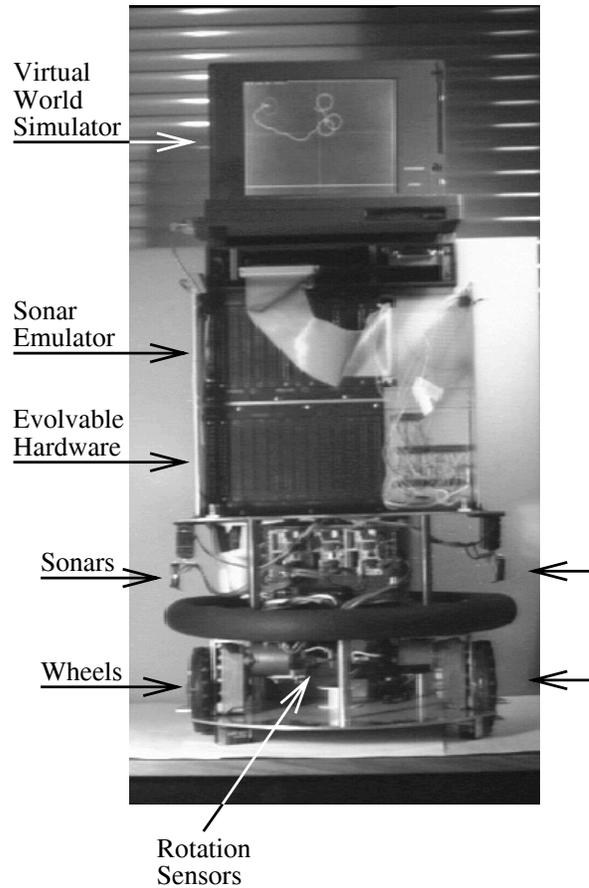
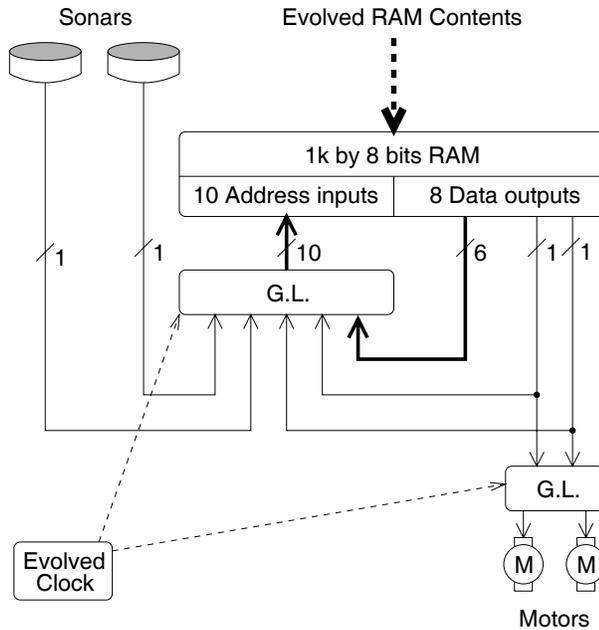


Fig. 3. The robot known as “Mr Chips.”

temporal constraints — constraints necessary for a designer’s abstraction but not for intrinsic EHW — endow the system with a rich range of potential dynamical behaviour, to the extent that the sonar echo pulses can be fed directly in, and the motors driven directly by the outputs, without any pre- or post-processing: no timers or pulse-width modulators. (The sonar firing cycle is asynchronous to the evolved clock).

Let this new architecture be called a *Dynamic State Machine* (DSM). It is not a finite-state machine because a description of its state must include the temporal relationship between the asynchronous signals, which is a real-valued analogue quantity. In the conventionally designed control system there was a clear sensory/control/motor decomposition (timers/controller/pulse-width-modulators), communicating in atemporal binary representations which hid the real-time dynamics of the sensorimotor systems, and the environment linking them, from the central controller. Now, the evolving DSM is intimately coupled to the real-time

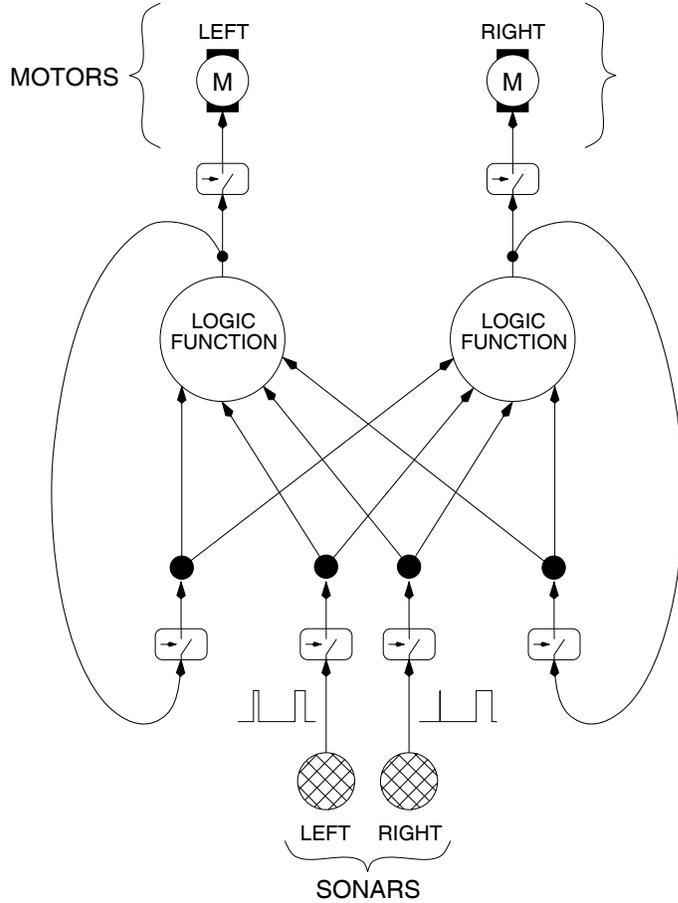


**Fig. 4.** The hardware implementation of the evolvable DSM. ‘G.L.’ stands for a bank of genetic latches: it is under genetic control whether each signal is passed straight through asynchronously, or whether it is latched according to the global clock of evolved frequency.

dynamics of its sensorimotor environment, so that real-valued time can play an important rôle throughout the system. The evolving DSM can explore special-purpose tight sensorimotor couplings because the temporal signals can quickly flow through the system being influenced by, and in turn perturbing, the DSM on their way.

For the simple wall-avoidance behaviour, only two of the possible eight feedback paths seen in Fig. 4 were enabled. The resulting DSM can be viewed as the fully connected, recurrent, mixed synchronous/asynchronous logic network shown in Fig. 5, where the bits stored in the RAM give a look-up table implementing any pair of logic functions of four inputs. This continuous-time dynamical system cannot be simulated in software, because the effects of the asynchronous variables and their interaction with the clocked ones depend upon the characteristics of the hardware: meta-stability and glitches will be rife, and the behaviour will depend upon physical properties of the implementation, such as propagation delays and meta-stability constants. Similarly, a designer would only be able to work within a small subset of the possible DSM configurations — the ones that are easier to analyse.

The genetic algorithm was the same as that described in Section 15.2, with the contents of the RAM (only 32 bits required for the machine with two feedback



**Fig. 5.** An alternative representation of the evolvable Dynamic State Machine, as used in the experiment. Each  is a ‘Genetic Latch’ (see previous figure).

paths), the period of the clock (16 bits, giving a clock frequency from around 2Hz to several kHz) and the clocked/unclocked condition of each signal all being directly encoded onto the linear bit-string genotype. The population size was 30, probability of crossover 0.7, and the mutation rate was set to be approximately 1 bit per genotype. If the distance of the robot from the centre of the room in the  $x$  and  $y$  directions at time  $t$  was  $c_x(t)$  and  $c_y(t)$ , then after an evaluation for  $T$  seconds, the robot’s fitness was a discrete approximation to the integral:

$$\text{fitness} = \frac{1}{T} \int_0^T \left( e^{-k_x c_x(t)^2} + e^{-k_y c_y(t)^2} - s(t) \right) dt \quad (2)$$

$k_x$  and  $k_y$  were chosen such that their respective Gaussian terms fell from their maximum values of 1.0 (when the robot was at the centre of the room) to a

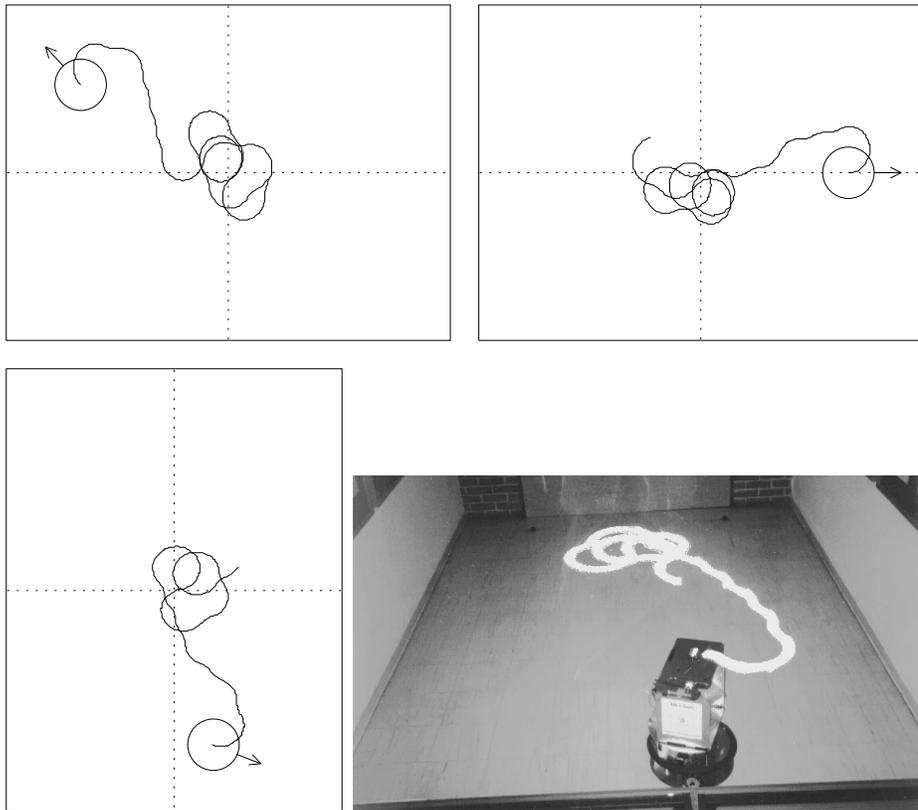
minimum of 0.1 when the robot was actually touching a wall in their respective directions. The function  $s(t)$  has the value 1 when the robot is stationary, otherwise it is 0: this term is to encourage the robot always to keep moving. Each individual was evaluated for four trials of 30 seconds each, starting with different positions and orientations. The worst of the four scores was taken as the fitness [28]. For the final few generations, the evaluations were extended to 90 seconds, to find controllers that were not only good at moving away from walls, but also *staying* away from them.

For convenience, evolution took place with the robot in a kind of ‘virtual reality.’ The real evolving hardware controlled the real motors, but the wheels were just spinning in the air. The wheels’ angular velocities were measured, and used by a real time simulation of the motor characteristics and robot dynamics to calculate how the robot would move. The sonar echo signals were then artificially synthesised and supplied in real time to the hardware DSM. Realistic levels of noise were included in the sensor and motor models, both of which were constructed by fitting curves to experimental measurements, including a probabilistic model for specular sonar reflections. The photograph of Fig. 3 was taken during an evolutionary run of this kind.

Fig. 6 shows the excellent performance which was attained after 35 generations, with a good transfer from the virtual environment to the real world. The robot is drawn to scale at its starting position, with its initial heading indicated by the arrow; thereafter only the trajectory of the centre of the robot is drawn. The bottom-right picture is a photograph of behaviour in the real world, taken by double-exposing a picture of the robot at its starting position, with a long exposure of a light fixed on top of the robot, moving in the darkened arena. If started repeatedly from the same position in the real world, the robot follows a different trajectory each time (occasionally *very* different), because of real-world noise. The robot displays the same qualitative range of behaviours in the virtual world, and the bottom pictures of Fig. 6 were deliberately chosen to illustrate this.

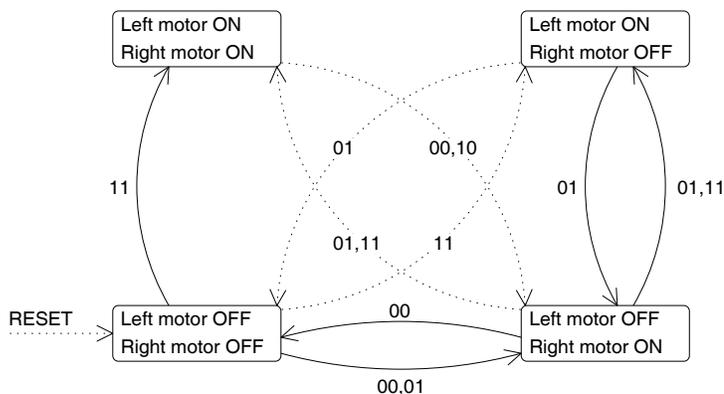
When it is remembered that this miniscule electronic circuit receives the raw echo signals from the sonars and directly drives the motors (one of which happens to be more powerful than the other), then this performance is surprisingly good. It is not possible for the DSM directly to drive the motors from the sonar inputs (in the manner of Braitenberg’s ‘Vehicle 2’ [29]), because the sonar pulses are too short to provide enough torque. Additionally, such naïve strategies would fail in the symmetrical situations seen at the top of Fig. 6. One of the evolved wall-avoiding DSMs was analysed (see below), 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.

Fig. 7 illustrates the state-transition behaviour of one of the wall avoiders. This particular individual used an evolved clock frequency of 9Hz (about twice the sonar pulse repetition rate). Both sonar inputs evolved to be asynchronous, and both motor outputs clocked, but the internal state variable that was clocked



**Fig. 6.** Wall avoidance in virtual reality and (bottom right) in the real world, after 35 generations. The top pictures are of 90 seconds of behaviour, the bottom ones of 60.

to become the left motor output was free-running (asynchronous), whereas that which became the right output was clocked. In the diagram, the dotted state transitions occur as soon as their input combination is present, but the solid transitions only happen when their input combinations are present at the same time as a rising clock edge. Since both motor outputs are synchronous, the state can be thought of as being sampled by the clock to become the motor outputs. This state-transition representation is misleadingly simple in appearance, because when this DSM is coupled to the input waveforms from the sonars and its environment, its dynamics are subtle, and the strategy being used is not at all obvious. It is possible to convince oneself that the diagram is consistent with the behaviour, but it would have been very difficult to predict the behaviour from the diagram, because of the rich feedback through the environment and sensorimotor systems on which this machine seems to rely. The behaviour even involves a stochastic component, arising from the probabilities of the asynchronous echo inputs being present in certain combinations at the clocking instant, and the



**Fig. 7.** A representation of one of the wall-avoiding DSMs. Asynchronous transitions are shown *dotted*, and synchronous transitions *solid*. The transitions are labelled with (*left, right*) sonar input combinations, and those causing no change of state are not shown. There is more to the behaviour than is seen immediately in this state-transition diagram, because it is not entirely a discrete-time system, and its dynamics are tightly coupled to those of the sonars and the rest of the environment.

probability of the machine being in a certain state at that same instant (remember that one of the feedback loops is unlocked).

Even this small system is non-trivial, and performs a difficult task with minimal resources, by means of its rich dynamics and exploitation of the real hardware. After relaxing the temporal constraints necessary to support the designer's finite-state model, a tiny amount of hardware has been able to display rather surprising abilities<sup>7</sup>. FPGAs are undoubtedly very much more powerful than the DSM. It is left to the reader to speculate on the potential capabilities of an evolvable chip containing many hundreds of logic gates, bearing in mind the power released by unconstrained evolution from the equivalent of only two gates in the DSM described above.

## 19 The Fault Tolerance of The Evolved Hardware Control System

To end the paper, we briefly use the evolved DSM robot controller to consolidate two of the points made earlier pertaining to fault tolerance.

Firstly, notice that the contents of the RAM were directly encoded bit-for-bit onto the genotype. A genetic mutation in the RAM coding region causes one of

<sup>7</sup> If all of the genetic latches were constrained to be synchronous, so that we have a FSM, then three control experiments failed to produce successful systems. If all of the latches were constrained to be asynchronous, then three control experiments each produced a recognisable wall-avoider, but much inferior to those evolved when the genetic latches were under evolutionary control.

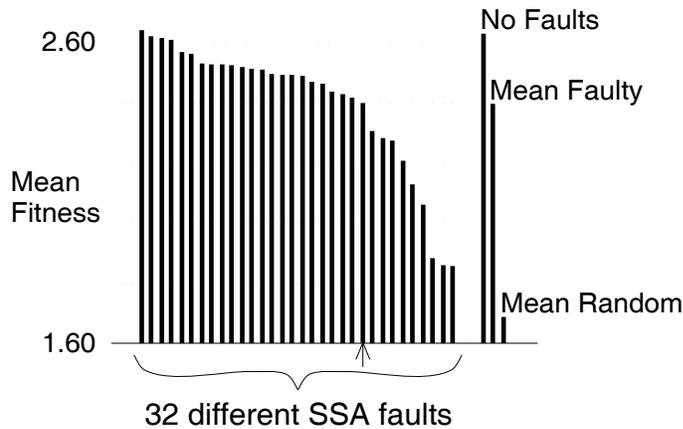


Fig. 8. Sensitivity to adverse SSA faults.

the bits in the RAM to be inverted: the same effect as a single-stuck-at (SSA) fault in the RAM’s memory array. By the argument of Section 14, there should therefore be a tendency for the evolved controller to be tolerant to such SSA faults. Fig. 8 shows that the evolved wall-avoider DSM is indeed quite robust to adverse SSA faults — observation of the robot’s qualitative behaviour bears this out — but it is not known how much is due to the action of evolution, and how much is simply a property of the DSM architecture. The 32 possible adverse SSA faults were each emulated in turn by writing the opposite value to that specified by the genotype to the RAM bit in question. For each fault, the DSM was then used to control the robot (in the virtual environment) for sixteen 90-second runs from the same starting position, and the average fitness was measured to give the data in the figure. The results are in accord with the theory, but gathering sufficient data from the real robot actually to verify the theory would be prohibitively time-consuming: hence the ongoing study using NK landscapes mentioned in Section 14.

The second experiment was to introduce the SSA fault marked with an arrow in Fig. 8 as a permanent feature in the DSM, and then to allow the already-evolved population to evolve further. At first, the fitness of the population was significantly lowered, with none of the individuals performing as well as the best of the population used to, but after 10 generations the mean and best fitnesses of the population had recovered to their previous values. This approach to fault-tolerance (proposed in Section 16 above) would be useful when transferring the evolved controller from one piece of hardware to another, or to cope with long-lasting faults within one.

## 20 Conclusion

We have formulated Evolvable Hardware — and in particular *intrinsic* EHW — as a synthesis of genetic algorithms, inspiration from natural evolution and electronics. In forming this synthesis, none of the three components has been left unaltered. For the evolution of complex structures, genetic algorithms need to be extended to incorporate the notions of converged species evolution and incrementally increasing complexity: the SAGA framework. When drawing inspiration from nature, it is necessary to recognise the significantly different constraints and opportunities associated with the artificial evolution of VLSI circuits. Finally, the whole concept of *what electronics can be* needs to be re-thought, because until now it has been governed by what is amenable to design techniques. Stepping into the wider space of dynamical electronic systems, the first intrinsically evolved hardware robot controller has been presented, showing remarkable levels of efficiency and robustness. There is every reason to expect that this new field will go far, but extravagant projections are not appropriate at this early stage.

## 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 EHW 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)*, University of Plymouth UK, 26–28<sup>th</sup> March 1996. <http://kmi.open.ac.uk/~monty/evoladaphwpaper.html>
2. Trevor A. York. Survey of field programmable logic devices. *Microprocessors and Microsystems*, 17(7):371–381, 1993.
3. David E. Goldberg. *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, Reading MA, 1989.
4. Inman Harvey. Species adaptation genetic algorithms: The basis for a continuing SAGA. In F. J. Varela and P. Bourguine, editors, *Toward a Practice of Autonomous Systems: Proc. of 1st Eur. Conf. on Artificial Life*, pp346–354. MIT Press/Bradford Books, Cambridge, MA, 1992.
5. Stuart Kauffman. Adaptation on rugged fitness landscapes. In Daniel L. Stein, editor, *Lectures in the Sciences of Complexity*, pp527–618. Addison Wesley: Santa Fe Institute Studies in the Sciences of Complexity, 1989.
6. M. Eigen, J. McCaskill, and P. Schuster. Molecular quasi-species. *Journal of Physical Chemistry*, 92:6881–6891, 1988.
7. M. Eigen and P. Schuster. *The Hypercycle: A Principle of Natural Self-Organization*. Springer-Verlag, 1979.
8. M. Nowak and P. Schuster. Error thresholds of replication in finite populations, mutation frequencies and the onset of Muller’s ratchet. *Journal of Theoretical Biology*, 137:375–395, 1989.
9. Inman Harvey. The SAGA cross: the mechanics of crossover for variable-length genetic algorithms. In R. Männer and B. Manderick, editors, *Parallel Problem Solving from Nature 2*, pp269–278. North-Holland, 1992.
10. G.E. Hinton and S.J. Nowlan. How learning can guide evolution. *Complex Systems*, 1:495–502, 1987.

11. I. Harvey, P. Husbands, and D. T. Cliff. Seeing the light: Artificial evolution, real vision. In D. Cliff, P. Husbands, J.-A. Meyer, and S. Wilson, editors, *From Animals to Animats 3: Proc. of 3rd Int. Conf. on Simulation of Adaptive Behaviour (SAB94)*, pp392–401. MIT Press/Bradford Books, Cambridge MA, 1994.
12. N. Jakobi, P. Husbands, and I. Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In F. Morán et al., editors, *Advances in Artificial Life: Proc. of 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp704–720. LNAI 929, Springer-Verlag, 1995.
13. Hugo de Garis. Evolvable hardware: Genetic programming of a Darwin Machine. In C.R. Reeves et al., editors, *Artificial Neural Nets and Genetic Algorithms - Proc. of the Int. Conf. in Innsbruck, Austria*, pp441–449. Springer-Verlag, 1993.
14. Adrian Thompson. Evolving fault tolerant systems. In *Proc. of 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems (GALESIA95), IEE Conference Publication No. 414*, pp524–529, 1995.
15. M. Eigen. New concepts for dealing with the evolution of nucleic acids. In *Cold Spring Harbor Symposia on Quantitative Biology*, volume LII, 1987.
16. Stuart A. Kauffman. *The Origins of Order*. Oxford University Press, 1993.
17. R. Brooks. Intelligence without representation. *Artificial Intelligence*, (47):139–159, 1991.
18. Günter P. Wagner. Adaptation and the modular design of organisms. In F Morán et al., editors, *Advances in Artificial Life: Proc. of 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp317–328. LNAI 929, Springer-Verlag, 1995.
19. 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.
20. Alan F. Murray. Analogue neural VLSI: Issues, trends and pulses. *Artificial Neural Networks*, (2):35–43, 1992.
21. Carver A. Mead. *Analog VLSI and Neural Systems*. Addison Wesley, 1989.
22. P. Kinget, M. Steyaert, and J. van der Spiegel. Full analog CMOS integration of very large time constants for synaptic transfer in neural networks. *Analog Integrated Circuits and Signal Processing*, 2(4):281–295, 1992.
23. Dave Cliff, Inman Harvey, and Phil Husbands. Explorations in evolutionary robotics. *Adaptive Behaviour*, 2(1):73–110, 1993.
24. Alexander Miczo. *Digital Logic Testing and Simulation*. Wiley New York, 1987.
25. IMP, Inc. *IMP50E10 EPAC Electronically Programmable Analog Circuit: Preliminary product information sheet*, November 1994.
26. Adrian Thompson. Evolving electronic robot controllers that exploit hardware resources. In F. Morán et al., eds., *Advances in Artificial Life: Proc. of 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp640–656. LNAI 929, Springer-Verlag, 1995.
27. D.J. Comer. *Digital Logic & State Machine Design*. Holt, Rinehart & Winston, 1984.
28. I. Harvey, P. Husbands, and D. Cliff. Genetic convergence in a species of evolved robot control architectures. In S. Forrest, editor, *Proc. of 5th Int. Conf. on Genetic Algorithms*, p636. Morgan Kaufmann, 1993.
29. V. Braitenberg. *Vehicles : Experiments in Synthetic Psychology*. MIT Press, 1984.
30. D.P.M. Northmore and J.G. Elias. Evolving synaptic connections for a silicon neuromorph. In *Proc of 1st IEEE Conf. on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 2, pp753–758. IEEE, New York, 1994.