Evolving Electronic Robot Controllers that Exploit Hardware Resources *
CSRP 368

Adrian Thompson

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

Abstract. Artificial evolution can operate upon reconfigurable electronic circuits to produce efficient and powerful control systems for autonomous mobile robots. Evolving physical hardware instead of control systems simulated in software results in more than just a raw speed increase: it is possible to exploit the physical properties of the implementation (such as the semiconductor physics of integrated circuits) to obtain control circuits of unprecedented power. The space of these evolvable circuits is far larger than the space of solutions in which a human designer works, because to make design tractable, a more abstract view than that of detailed physics must be adopted. To allow circuits to be designed at this abstract level, constraints are applied to the design that limit how the natural dynamical behaviour of the components is reflected in the overall behaviour of the system. This paper reasons that these constraints can be removed when using artificial evolution, releasing huge potential even from small circuits. Experimental evidence is given for this argument, including the first reported evolution of a real hardware control system for a real robot.

Keywords: Evolvable Hardware, Evolutionary Robotics, Physics of Computation.

1 Introduction — What is Evolvable Hardware?

Evolvable hardware [4, 10, 8, 9, 20, 24] is a reconfigurable electronic circuit, which can be changed by an adaptive process such as a genetic algorithm. This paper considers the evolution of hardware to control an autonomous mobile robot; initially by examining exactly what evolvable hardware is, and what its advantages over software systems are, and then by concentrating on one of these benefits: the exploitation of the physics of the implementation, and how this may be maximised. Finally, early results are presented for the first ever evolution of real hardware to control a real robot, which benefits from the change of perspective that I claim evolvable hardware justifies.

A type of commercially available VLSI chip called a Field Programmable Gate Array (FPGA) [25] will provide a good illustration of how hardware may be subject to adaptation, although many other evolvable hardware architectures (both analogue and digital) are possible. A typical FPGA consists of an array of hundreds of reconfigurable blocks that can perform a variety of digital logic functions, and a set of wires to which the inputs and outputs of the blocks can be connected (Figure 1). What logic functions are performed by the blocks, and how the wires are connected to the blocks and to each-other, can be thought of as being controlled by electronic switches (represented as dots in the diagram). The settings of these switches are determined by the contents of digital memory cells. For example, if a block could perform any one of the 2^4 boolean functions of two inputs, then four bits of this "configuration memory" would be needed to determine its behaviour. The blocks around the periphery of the array have special configuration switches to control how they are interfaced to the external connections of the chip (its pins).

The configuration memory of an FPGA can be conceptualised as its genotype, which determines what the blocks do and how they are wired together. If an FPGA is the control system of a

* ©Adrian Thompson, 1995. All rights reserved. To appear in The Proceedings of The 3rd European Conference on Artificial Life (ECAL95), Springer Verlag 1995.

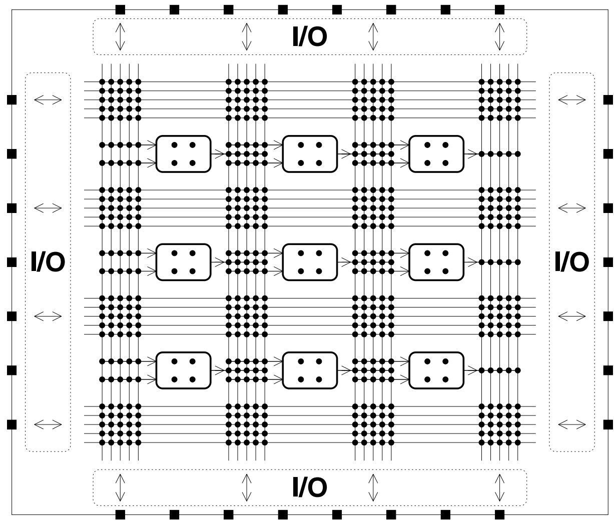


Fig. 1. Part of a simplified typical FPGA architecture.

robot, then artificial evolution can manipulate an encoding of the configuration memory (genotype) according to the robot behaviour induced by the corresponding circuit implemented on the FPGA: the hardware is evolvable. Commercial FPGAs have more features than I have mentioned, but the principles are the same.

There is often no clear distinction between software and hardware, however. The contents of an FPGA's configuration memory can be thought of as causing it to instantiate a particular circuit by means of setting its configuration switches, but also the contents of the configuration memory might be viewed as the "program" of a parallel computer that happens to be called an FPGA. Which of these viewpoints is adopted is a matter of deciding on the most appropriate style of description for the particular system in question: a conventional personal computer could be envisaged as evolvable hardware, but it is usually more useful to think of it as a fixed piece of electronics that operates upon data and instructions held in a memory. An intimately linked consideration is whether the evolving system is viewed as a computational one (performing calculations or manipulating symbols), or whether it is seen from the much wider perspective of dynamical systems theory. Later, I shall advocate robot control systems that are appropriately described as non-computational dynamical systems evolving in hardware.

There are two ways in which artificial evolution may be applied to a reconfigurable hardware system. In the first (sometimes known as "*extrinsic* evolvable hardware"), an evolutionary algorithm produces a configuration based on the performance of a software simulation of the reconfigurable hardware. The final configuration is then downloaded onto the real hardware in a separate implementation step: a useful approach if the hardware is only capable of being reconfigured a small number of times. In the second ("intrinsic"[2]) method, each time the evolutionary algorithm generates a new variant configuration, it is used to configure the real hardware, which is then evaluated at its task. Consequently, an implemented system is evolved directly; the constraints imposed by the hardware are satisfied automatically, and all of its detailed (difficult to simulate) characteristics can be brought to bear on the control problem. This second method is much more powerful than the first and is what I mean by "evolvable hardware" in this paper.

I shall use the term "evolution" to mean any artificial evolution-like process, and the genetic algorithm in particular [11]. Much of the discussion also applies to a broader class of adaptive processes, including learning techniques, which may usefully be used in conjunction with artificial evolution, or even instead of it.

Next, Section 2 examines the motivations behind the evolution of hardware systems. One of the possible objectives — the maximal exploitation of hardware resources — is singled out as being particularly interesting; to facilitate this, a new approach is proposed. Section 3 consolidates some

[2] The terms *extrinsic* and *intrinsic* evolvable hardware are due to Hugo de Garis.

of the ideas put forward, by means of a simulation study using a highly abstract model of an FPGA. Finally, Section 4 describes a new evolvable hardware architecture, which is easy to build, but yet takes many of the important ideas on-board. The benefits of this architecture, and the underlying approach, are demonstrated by the real-world performance of this machine in controlling a robot.

2 Why Evolve in Hardware?

Under what conditions is the use of evolvable hardware beneficial, when compared with systems evolved in software simulation? This section identifies three areas: the first two are related to the raw speed of hardware, while the third is more profound and suggests powerful new kinds of system that have not been seen before.

2.1 Speed of Operation

Currently, much of the research on evolved control systems for autonomous agents is centred around simulations of neuro-mimetic networks performed on a general purpose computer. As expertise in the evolution of complex systems advances, so the size and complexity of these networks will increase, with a corresponding decrease in the speed with which they can be simulated in this way. If the speed increase in general purpose computers does not keep pace with improvements in evolutionary techniques, then general purpose computers will have to be abandoned, and more specialised parallel machines used to perform the simulations. Initially, coarse-grain parallel multi-processor computers will be sufficient, but if the complexity of the networks increases still more, progressively finer grained parallelism will have to be utilised, until eventually there is one special purpose processor for each neural node. We would then have arrived at a reconfigurable hardware implementation for artificial neural networks — evolvable hardware.

The assumption that our ability to evolve control systems will outstrip the speed of simulating computers is not necessarily true. Nevertheless, it is interesting to observe that if our aspirations for building the most complex technically-possible brain-like systems are fulfilled, then the result will be an evolvable hardware implementation of the system. In that case, would it not be possible to evolve hardware *in its own right* (as an electronic circuit, and not as an implementation of anything else) and obtain an even more powerful system, better suited to silicon than other kinds of architectures like neural networks? I shall return to this question soon.

2.2 Accelerated Evolution

When evolving a robot control system to achieve some task, the time taken before a satisfactory controller is obtained is usually dominated by how long it takes to evaluate each variant controller's efficacy at that task (fitness evaluation). The genetic operations take a small amount of time in comparison.

The obvious way to evaluate a control system is to connect it to the robot and see how well it performs in the real world, in real time. Even for tasks that take a short length of time to perform (a minute or two), the large number of fitness evaluations normally required can make this highly time-consuming. Consequently, there is a case for interfacing the evolving hardware control system to a high-speed simulation of the robot and its environment, in order to accelerate the entire evolutionary process.

It is suggested by de Garis[4] that the environment simulation could be implemented in special purpose electronics situated next to the evolving hardware control system on a VLSI chip. The implementation of the simulator in hardware is made feasible by modern automatic synthesis techniques, which can derive a circuit from a textual description resembling a computer program. Implementing the environmental simulator in hardware rather than software makes it faster, but does not solve the problem that it is extremely difficult adequately to simulate the interactions between a control system and its environment, such that a control system evolved in the simulated world behaves in a satisfactory way in the real world. This is especially the case when vision is involved [2]. Nevertheless, it is possible that environment simulation in special purpose hardware will be an important tool as new techniques are developed [13, 23].

When a circuit that has been rapidly evolved for behaviour in a high-speed simulated world is ready for use in the *real* world, all of its dynamics that influence the robot's behaviour must be slowed down by the same factor by which the real world is slower than the simulation. (Imagine a controller that was evolved for a high-speed simulated world and was then let loose in the real world without being slowed down. Everything in the environment would then be happening slower than it "expected," and the motor signals produced would tend to be too fast for the robot's actuators and the world. It would probably no longer perform the task.) This means that the acceleration of evolution through the use of a high-speed simulated environment is at the cost of the efficiency of the control circuit produced. The final circuit cannot be making maximal use of the available hardware when it is operating in the real world, because it is capable of producing the same behaviour in a world that is running faster: the resources needed to allow for this could be being used for real-world performance. This may, however, be a sensible use of some of the high speed available from electronic hardware.

The fact that it must be possible to adjust the speed of all of the control circuit's dynamics that affect the behaviour of the robot restricts the set of control circuits that could be produced by evolution in a high-speed world. Either the time-scales of all of the semiconductor physics must be adjustable by large amounts (this is not practical), or the aspects of the control circuit's dynamics that make a difference to the robot must be restricted such that they are more easy to control. The latter can be arranged by restricting evolution to use pre-defined indivisible modules that have adjustable time-constants, or by applying the restriction of discrete-time dynamics through the use of an adjustable clock. Each of these possibilities diminishes the degree to which evolution can exploit the resources available from the reconfigurable hardware. Again, this may often be a sensible sacrifice to make for accelerated evolution, but the remainder of this paper deals with an interesting alternative.

2.3 Exploitation of Semiconductor Physics

Work with the "gantry robot" at the University of Sussex [7] suggests that it may be feasible to carry out artificial evolution in a real-world environment with fitness evaluations taking place in real time. If, in addition, evolution is given control over the reconfigurable hardware at the lowest possible level (for example the "configuration bits" of the FPGA mentioned in Section 1) to produce an electronic circuit as a type of control system in its own right (and not as an implementation of anything else), then circuits of unprecedented power and efficiency will be produced. These systems can exploit every facet of the characteristics of the reconfigurable hardware on which they are developed, because all of the natural real-time behaviours of its components (their physics) are allowed potentially to affect the robot's behaviour, and detailed control is provided to tune their collective action to achieve the task.

In nature, control systems are always adapted to the manner in which they are implemented, because their evolutionary success is determined by their effectiveness as real implemented systems. If engineering success is the objective, then implementations of neural networks (that evolved for an implementation allowing slow, highly unconstrained connections between slow units) may be a bad use of integrated circuits, which are very fast, but have severe constraints on interconnections (because of their planar nature). Perhaps it will be possible to evolve an architecture that is better suited to silicon than neural networks, but yet induces intelligent ("brain-like") behaviour into a robot. The fine-grained control over the hardware that is required implies a vast search-space for a system of any complexity, so techniques need to be developed cope with this. One possibility is the use of developmental genetic encoding schemes for genetic algorithms [12], which allow the evolution of re-usable building blocks (analogous to neurons?), permitting fine-grained tuning of the building blocks, but yet reducing the search space to systems built out of them.

The space of circuits of the type I propose is very much larger than the space of solutions available to a human designer. The designer works with high-level models of how components or higher-level building blocks behave and interact. Design constraints are adopted to prevent the imperfections of these models from affecting overall behaviour. One such constraint is the modularisation of the design into parts with simple, well defined interactions between them. Another is the use of a clock to prevent the natural dynamics of the components from affecting overall

behaviour: the clock is used so that the components are given time to reach a steady state before their condition is allowed to influence the rest of the system. I suggest (with the aid of the empirical evidence presented in the following sections) that such constraints should be abolished whenever they are a limitation on the potentially useful behaviour of an evolving hardware system. A designer carefully avoids "glitches," "cross-talk," "transients" and "meta-stability," but all of these things could be *put to use* by artificial evolution.

Evolution could also put to use properties of the hardware that the designer could *never* know about. For instance, a circuit may evolve to rely on some internal time-delays of an integrated circuit that are not externally observable. Even if there is a silicon defect, the system could evolve to use whatever function the "faulty" part happened to perform. This raises a fundamental problem: a circuit that is evolved for a particular evolvable hardware system (a certain FPGA chip, for example) may not work on a different system that is nominally identical — no two silicon chips are the same. There are several ways in which this could be avoided. The circuits could be evolved to be robust to perturbations in some properties that vary from chip to chip (by altering the chip's temperature or power supply during evolution, for example). Evolution could be forced to produce building blocks that are repeatedly used in the circuit, and would therefore be insensitive to characteristics that varied across one chip. Evolution could evaluate a configuration on more than one piece of reconfigurable hardware when judging its quality (this can also be done using a single reconfigurable chip that can instantiate the same circuit in several different ways, e.g. by using an FPGA's rotational symmetry). Finally, it could be accepted that further adaptation will have to take place each time a configuration is transferred from one reconfigurable device to another [14, 15, 16, 17].

The next two sections of this paper will provide experimental evidence for the ideas I have put forward here. Firstly, I present a simulation of the evolution of an FPGA configuration, which demonstrates that evolution can produce circuits optimised for a particular implementation, and in the absence of modularisation and clocking constraints. Then I describe a real evolved hardware control system that controls a real robot, and was produced according to the above rationale, demonstrating its benefits.

3 A Millisecond Oscillator from Nanosecond Logic Gates

Abandoning the external clock can reap even more rewards than were mentioned above. A clocked digital system is a finite-state machine, whereas an unclocked (asynchronous) digital system is not. To describe the state of an unclocked circuit, the temporal relationships between its parts must be included. These are continuously variable analogue quantities, so the machine is not finite-state. This theoretical point gives a clue to a practical advantage: in an unclocked digital system, it is possible to perform analogue operations using the time dimension, even when the logic gates assume only binary values (see for example, the *pulse stream* technique [21, 22]).

In the previous section, I argued that when producing circuits by evolution rather than design, the use of a clock is often an *unnecessary* limitation on the way in which the natural dynamics of the components can be used to mediate robot behaviour. This is not always the case — electronic components usually operate on time-scales much smaller than would be useful to a robot; unless the system can evolve such that the overall behaviour of the components (when integrated into the sensorimotor feedback loop of the robot) is much slower than the behaviour of individual components, then a clock (perhaps of evolvable frequency) will be required to give control over the time-scales. (Sometimes, the use of a clock can *expand* the useful dynamics possible from the evolving circuit.)

A simulation has been performed to investigate whether a genetic algorithm can evolve a recurrent asynchronous network of high speed logic gates to produce behaviour on a time-scale that would be useful to a robot. 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 (the nodes were analogous to the reconfigurable logic blocks of an FPGA), and how the nodes were connected (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 [2]. 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 [19], 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 0.5×10^{-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 nth transition occurring at time t_n seconds, then the average error in the time spent at each level was calculated as :

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

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 (but took very much more wall-clock time to simulate). 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 [5], but used 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 6.0×10^{-4} per bit.

The experiment succeeded. Figure 2 shows that the output after 40 generations 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. The final circuit (Figure 3) *was* exploiting the characteristics of its "implementation" — if the propagation delays were changed, it reverted to behaviour similar to that at the first generation. A spike-train, rather than the desired square-wave was produced, allowing the phenomenon of spike trains of slightly different frequencies beating together to produce a much lower frequency (but it is difficult to gain the massive reduction in frequency required and yet produce a regular output). The entire network contributes to the behaviour, and meaningful sub-networks could not be identified.

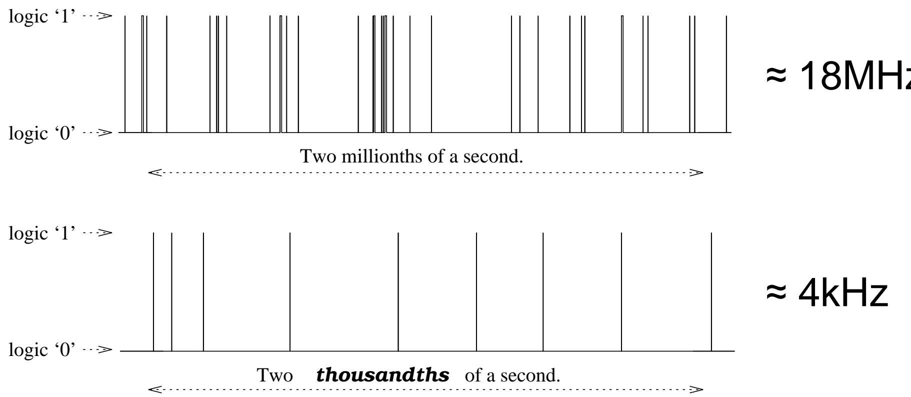


Fig. 2. Output of the evolving oscillator. (Top) Best of the initial random population of 30 individuals, (Bottom) best of generation 40. Note the different time axes. A visible line is drawn for every output spike, and in the lower picture each line represents a single spike.

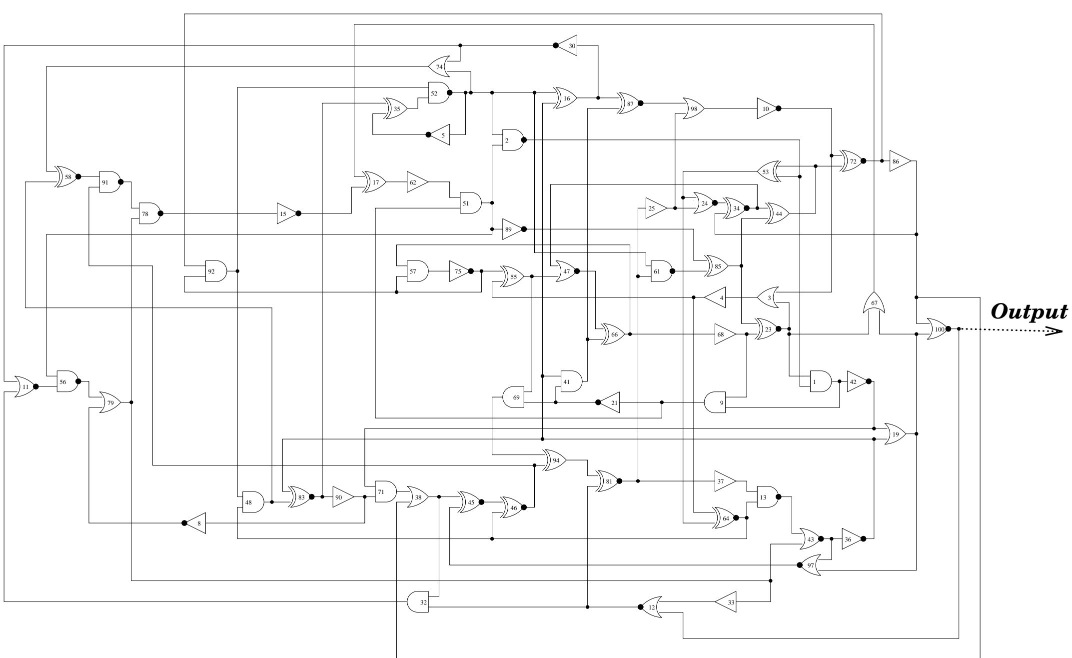


Fig. 3. The evolved 4kHz oscillator (unconnected gates removed, leaving the 68 shown).

This simulation, although quite an unrealistic model of the evolution of a real FPGA configuration, has shown how evolution can assemble high speed components to produce behaviour on a time-scale that approaches that useful to a robot. It exploits the characteristics of the implementation, and does not require the imposition of spatial or temporal constraints such as modularisation or clocking. The style of solution adopted (the beating of spike trains) is an analogue operation over the time axis, and would have been more difficult in a discrete time system.

4 A Real Evolved Hardware Robot Controller

In this experiment, a real hardware robot control system was evolved for wall-avoidance behaviour in an empty 2.9m×4.2m rectangular arena, using sonar time-of-flight sensing. The two-wheeled robot ("Mr Chips," Figure 4(a)) has a diameter of 46cm, and a height of 63cm. For this scenario, its only sensors were a pair of fixed sonar heads pointing left and right.

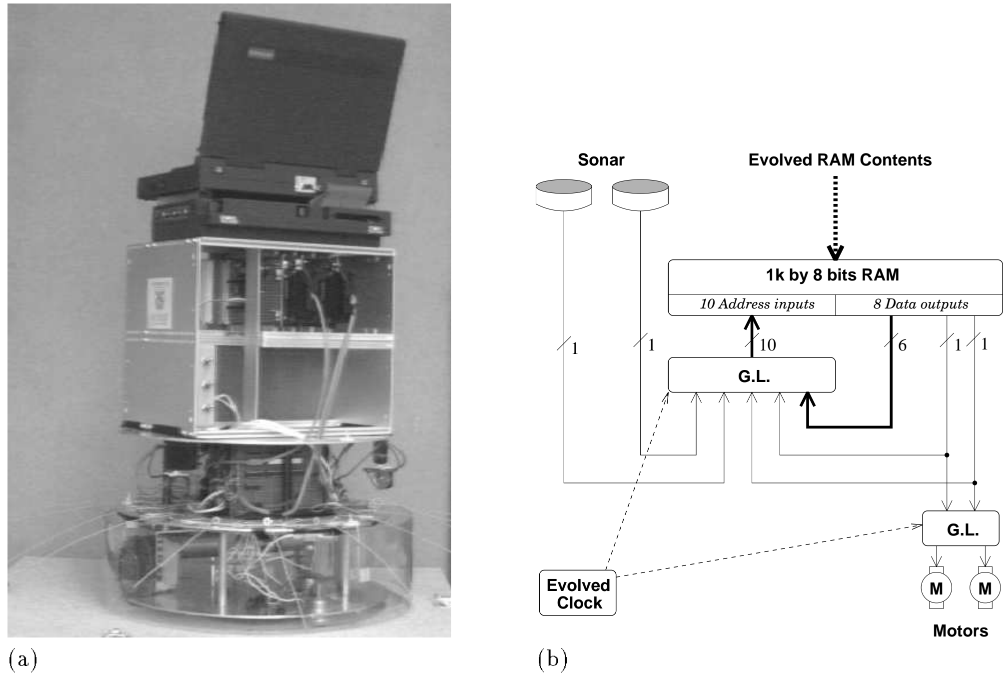


Fig. 4. (a) The "Mr Chips" Robot. (b) The evolvable Dynamic State Machine. "G.L." stands for "Genetic Latch": which of the bits are latched according to the clock, and which are passed straight through is under genetic control.

Consider how this control problem might traditionally have been solved using finite-state machines (FSMs). Firstly, a pair of FSMs would be used to measure the time of flight of the sonar "pings." When the sonar echoes returned, each timer would deliver a binary code representing the time of flight to a central "control" FSM, which would compute a motor response on the basis of its current internal state and these inputs. The control FSM would then deliver a binary code representing desired motor speed to each of a pair of pulse-width-modulation FSMs, which would drive the d.c. motors with the appropriate waveforms. Notice that there is a very strict sensory/control/motor functional decomposition inherent in this architecture, and the system is a computational one, in which the control FSM deals in binary codes and is totally divorced from the dynamics of the sensorimotor systems and the environment. It would be possible to evolve the control FSM in hardware, but there would be no benefit: *exactly* the same behaviour would be obtained from an implementation of the FSM in software. A hardware implementation would use a clocked register to hold the current state, and a random-access memory (RAM) chip with evolvable

contents would hold the next-state and output variables corresponding to each present-state and input combination (this is the well known "direct-addressed ROM" implementation of an FSM [3]).

The hardware control system used in this experiment is superficially similar to an FSM, but fundamentally different. Call it a Dynamic State Machine (DSM). As its input, it directly takes the echo signals from the sonars. There is one wire from each sonar, on which pulses arrive: the lengths of these pulses are equal to the time of flight for that sonar. (The sonars fire, and the pulses begin, simultaneously, with a pulse repetition rate of 5Hz. The sonar firing cycle is completely asynchronous to the DSM.) The output of the DSM goes directly to the motor drivers; if the motors are to go at an intermediate speed, then the DSM must pulse them itself. Like the FSM, the DSM implementation is centred around an RAM chip with evolvable contents, but which of the input, state and output variables are clocked, and which are free running (asynchronous) is genetically determined. For those that *are* clocked, the clock frequency is also genetically determined. The full arrangement is shown in Figure 4(b).

The sensory/control/motor functional decomposition has now been removed, and the control system is intimately linked to the dynamics of the sensorimotor signals and the environment, with time now able to play an important role throughout the system. The possibility of mixing asynchronous state variables with state variables being clocked at an evolved frequency endows the system with a rich range of possible dynamical behaviour: its actions can immediately be influenced by the input signals, but at the same time it is able to keep a trace of previous stimuli and actions over a time-scale that is under evolutionary control. It is able to exploit 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.

The presence of asynchronous state variables means that this is not a finite-state machine (their continuous-valued temporal relationships need to be included in a description of the machine's state). It would not be possible to simulate this machine 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 more predictable ones.

For the simple wall-avoidance behaviour, only the two state variables that also go to the motors were used — the others were disabled, and can be introduced incrementally as the difficulty of the task is increased. The genetic algorithm was the same as that described in the previous section, with the contents of the RAM (only 32 bits required for the machine with two state variables), the period of the clock (16 bits, giving a clock frequency from around 2Hz to several kHz) and the clocked/unclocked condition of each variable 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 string. (It can be shown that this small DSM is statistically likely to visit all of its possible states: DSMs with more state variables are likely to visit a smaller fraction of their possible states, and the mutation rate needs to be higher, because many mutations will have no immediate phenotypic effect.) 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:

$$\text{fitness} = \frac{1}{T}\int_0^T \left(e^{-k_x c_x(t)^2} + e^{-k_y c_y(t)^2} - s(t)\right) dt \qquad \text{where } s(t) = \begin{cases} 1 \text{ when stationary} \\ 0 \text{ otherwise} \end{cases} \tag{2}$$

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 $s(t)$ 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 being taken as its fitness [6]. 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.

Figure 5 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 Figure 5 were deliberately chosen to illustrate this.

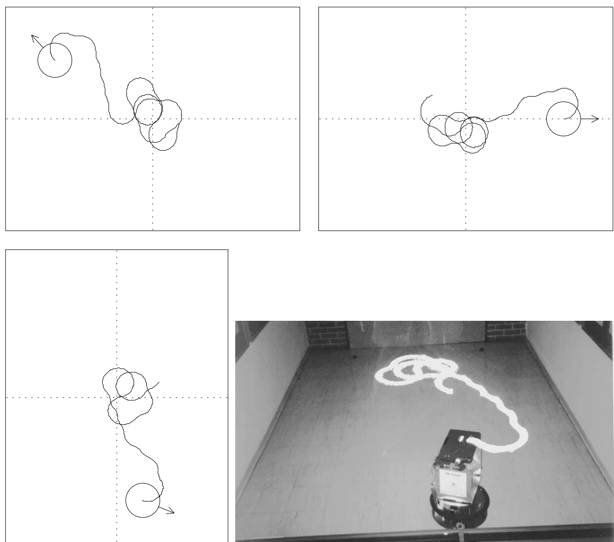
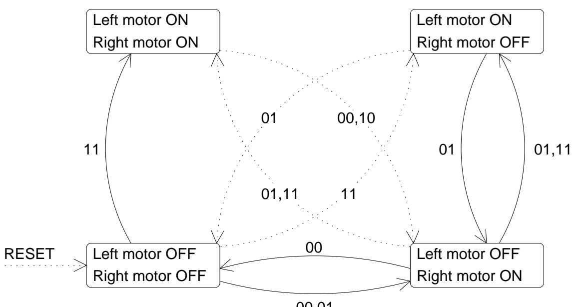


Fig. 5. Wall avoidance in virtual reality and (bottom right) in the real world, after 35 generations. The top pictures are of 90 seconds of behaviour, the bottom ones of 60.



Asynchronous transitions are shown *dotted.*

Synchronous transitions are shown *solid.*

Arcs are labelled with *(left, right)* sonar input combinations. Inputs causing no change of state are not shown.

For this controller, the values of the state variables shown are *latched by the clock* before becoming the actual motor outputs.

Fig. 6. A representation of one of the wall-avoiding DSMs. There is more to its 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.

When it is remembered that the DSM receives the raw echo signals from the sonars and directly drives the motors (one of which happens to be more powerful than the other), with only two internal state variables, then this performance in 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" [1]), 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 Figure 5. 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.

Figure 6 illustrates the state-transition behaviour of one of the wall avoiders. This particular individual used a clock frequency of 9Hz (about twice the sonar pulse repetition rate). Both sonar inputs were asynchronous, and both motor outputs were clocked, but the internal state variable that was clocked 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 probability of the machine being in a certain state at that same instant (remember that one of the state variables is free-running).

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. Only time will tell whether the DSM architecture will be capable of more sophisticated behaviour, using more state variables. The success of this particular architecture in the long term is less important than the powerful demonstration of the principles which inspired it, given here.

5 Conclusion

This paper has been a manifesto for an approach to evolving hardware for robot control. This approach aims fully to exploit the potential power of reconfigurable electronic hardware by relaxing the constraints on the spatial and temporal organisation of the system that are necessary for a human designer. The resulting systems may seem extraordinary to engineers at first, but experimental evidence has been presented to corroborate the claim that this may be a route to electronic control systems of unprecedented power and efficiency. A particular architecture — the Dynamic State Machine — has been proposed as a reconfigurable system that is easier to build than the more sophisticated alternatives based around Field Programmable Gate Array (FPGA) technology, but yet contains the essential ingredients. Its successful use in the first reported evolved hardware control system for a real robot demonstrates the *viability* of the new framework, but the long term goal must be to evolve systems as hardware that cannot be made in any other way — it could be that this is possible even with currently available FPGAs.

6 Acknowledgements

This research is funded by a D.Phil. scholarship from the School of Cognitive and Computing Sciences, for which I am very grateful. Special thanks are also due to Phil Husbands, Dave Cliff and Inman Harvey for their kind, expert and tireless help.

References

1. Valentino Braitenberg. *Vehicles : Experiments in Synthetic Psychology.* MIT Press, 1984.
2. Dave Cliff, Inman Harvey, and Phil Husbands. Explorations in evolutionary robotics. *Adaptive Behaviour*, 2(1):73–110, 1993.
3. David J. Comer. *Digital Logic & State Machine Design.* Holt, Rinehart and Winston, 1984.
4. Hugo de Garis. Evolvable hardware: Genetic programming of a Darwin Machine. In C.R. Reeves R.F. Albrecht and N.C. Steele, editors, *Artificial Neural Nets and Genetic Algorithms - Proceedings of the International Conference in Innsbruck, Austria*, pages 441–449. Springer-Verlag, 1993.
5. David E. Goldberg. *Genetic Algorithms in Search, Optimisation & Machine Learning.* Addison Wesley, 1989.
6. I. Harvey, P. Husbands, and D. Cliff. Genetic convergence in a species of evolved robot control architectures. CSRP 267, School of Cognitive and Computing Sciences, University of Sussex, 1993.
7. Inman Harvey, Phil Husbands, and Dave Cliff. Seeing the light : Artificial evolution, real vision. In Dave Cliff, Philip Husbands, Jean-Arcady Meyer, and Stewart W. Wilson, editors, *From animals to animats 3: Proceedings of the third international conference on simulation of adaptive behaviour*, pages 392–401. MIT Press, 1994.
8. Hitoshi Hemmi, Jun'ichi Mizoguchi, and Katsunori Shimohara. Development and evolution of hardware behaviours. In Rodney Brooks and Pattie Maes, editors, *Artificial Life IV*, pages 317–376. MIT Press, 1994.
9. Tetsuya Higuchi, Hitoshi Iba, and Bernard Manderick. *Massively Parallel Artificial Intelligence*, chapter "Evolvable Hardware", pages 195–217. MIT Press, 1994. Edited by Hiroaki Kitano.
10. Tetsuya Higuchi, Tatsuya Niwa, Toshio Tanaka, Hitoshi Iba, Hugo de Garis, and Tatsumi Furuya. Evolving hardware with genetic learning: A first step towards building a Darwin Machine. In *Proceedings of the 2nd Int. Conf. on the Simulation of Adaptive Behaviour (SAB92).* MIT Press, 1993.
11. J. H. Holland. *Adaptation in Natural and Artificial Systems.* Ann Arbor: University of Michigan Press, 1975.
12. Philip Husbands, Inman Harvey, Dave Cliff, and Geoffrey Miller. The use of genetic algorithms for development of sensorimotor control systems. In P. Gaussier and J-D. Nicoud, editors, *From Perception to Action Conference*, pages 110–121. IEEE Computer Society Press, 1994.
13. Nick Jakobi, Phil Husbands, and Inman Harvey. Noise and the reality gap: The use of simulation in evolutionary robotics. In *To appear: Proceedings of the 3rd European Conference on Artificial Life (ECAL95)*, Granada, June 4-6 1995. Springer-Verlag.
14. D. Mange. Wetware as a bridge between computer engineering and biology. In *Proceedings of the 2nd European Conference on Artificial Life (ECAL93)*, pages 658–667, Brussels, May 24-26 1993.
15. Daniel Mange and André Stauffer. *Artificial Life and Virtual Reality*, chapter "Introduction to Embryonics: Towards new self-repairing and self-reproducing hardware based on biological-like properties", pages 61–72. John Wiley, Chichester, England, 1994.
16. P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand. Achieving von Neumann's dream: Artificial life on silicon. In *Proc. of the IEEE International Conference on Neural Networks, icNN'94*, volume IV, pages 2321–2326, 1994.
17. P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand. Embryological development on silicon. In Rodney Brooks and Pattie Maes, editors, *Artificial Life IV*, pages 365–366. MIT Press, 1994.
18. Carver A. Mead. *Analog VLSI and Neural Systems.* Addison Wesley, 1989.
19. Alexander Miczo. *Digital Logic Testing and Simulation.* Wiley New York, 1987.
20. Jun'ichi Mizoguchi, Hitoshi Hemmi, and Katsunori Shimohara. Production genetic algorithms for automated hardware design through an evolutionary process. In *IEEE Conference on Evolutionary Computation*, 1994.
21. A. F. Murray et al. Pulsed silicon neural networks - following the biological leader. In Ramacher and Rückert, editors, *VLSI Design of Neural Networks*, pages 103–123. Kluwer Academic Publishers, 1991.
22. Alan F. Murray. Analogue neural VLSI: Issues, trends and pulses. *Artificial Neural Networks*, (2):35–43, 1992.
23. Stefano Nolfi, Orazio Miglino, and Domenico Parisi. Phenotypic plasticity in evolving neural networks. In P. Gaussier and J-D. Nicoud, editors, *From Perception to Action Conference*, pages 146–157. IEEE Computer Society Press, 1994.
24. David P.M. Northmore and John G. Elias. Evolving synaptic connections for a silicon neuromorph. In *Proc of the 1st IEEE Conference on Evolutionary Computation, IEEE World Congress on Computational Intelligence*, volume 2, pages 753–758. IEEE, New York, 1994.
25. Trevor A. York. Survey of field programmable logic devices. *Microprocessors and Microsystems*, 17(7):371–381, 1993.