Silicon Evolution

Adrian Thompson
School of Cognitive and Computing Sciences, University of Sussex, Brighton BN1 9QH, UK.
adrianth@cogs.susx.ac.uk, http://www.cogs.susx.ac.uk/users/adrianth/

Abstract

The advent of new families of reconfigurable integrated circuits makes it possible for artificial evolution to manipulate a real physical substrate to produce electronic circuits evaluated in the real world. This raises new issues about the potential nature of electronic circuits, because evolution uses no modelling, abstraction or analysis; only physical behaviour. The simplifying constraints of conventional design methodologies can be dropped, allowing evolution to exploit the full range of physical dynamics available from the silicon medium. This claim is investigated theoretically and in simulation, before presenting the first reported direct evolution of the configuration of a Field Programmable Gate Array (FPGA). Evolution is seen to harness its natural dynamics and exploit them in achieving a real-world task.

1 Introduction

There is a type of Very-Large Scale Integrated circuit (a VLSI chip) known as a Field-Programmable Gate Array (FPGA). These chips do not have a predetermined function, but provide uncommitted electronic resources which the user can configure *from software.* One can imagine the attributes of the components, and the way in which they are interconnected, to be under the control of a set of transistor switches. The states of these switches are determined by the contents of digital memory cells (similar to normal computer memory) situated next to them on the silicon. By writing to this "configuration store," the user can physically create new electronic circuits as easily as writing to computer memory.

This paper discusses the use of artificial evolution to generate these circuit configurations automatically. I argue that once evolution is instantiating individuals as real physical electronic circuits, and fitness evaluation is the process of observing the laws of physics unfolding within them in real time, a new perspective is called for. Standard views of how electronic circuits should be structured are no longer appropriate. The reconfigurable hardware is a continuous-time, analogue dynamical electronic system: a primordial soup of resources to be exploited. However, once the simplifying design constraints used by human designers are dropped, can artificial evolution effectively control the behaviour of a totally unconstrained, recurrent, arrangement of high-speed electronic components? The main body of the paper addresses this question, firstly through a simulation study and then using a real FPGA (the first reported direct evolution of an FPGA configuration). We shall see that evolution can indeed sculpt the dynamics of the system.

The next section gives a little more detail on what reconfigurable hardware is, how it can be evolved, and why. The implications of removing inappropriate design constraints are then considered, and the experiments shown to corroborate the new perspective.

2 Evolvable Hardware

The notion of *evolvable hardware,* i.e. reconfigurable hardware to which artificial evolution can be applied, has been debated to an increasing extent over the last six years (de Garis, 1991; Higuchi et al., 1993; de Garis, 1993; Mange, 1993; Hemmi et al., 1994; Marchal et al., 1994; Mizoguchi et al., 1994; Thompson, 1995a, 1995b; Hirst, 1996; Thompson et al., 1996), but only now are new FPGAs making it feasible. Previously (York, 1993), commercial off-the-shelf chips have been unsuitable because of a limit on the number of times they could be reconfigured, the presence of a layer of secret proprietary software between the user and the chip, or the possibility of easily damaging the device with an invalid configuration.

It is possible to build custom reconfigurable hardware systems out of separate components (Thompson, 1995a), and small analogue FPGAs are now available (IMP, 1994). However, it is FPGAs intended for the implementation of digital logic systems which hold the most promise in the near future. Figure 1 shows a simplified typical architecture: there is a large two-dimensional array of reconfigurable logic blocks, each with a collection of wires running past it. Reconfigurable switches (shown as dots in the figure) determine how the inputs and out-

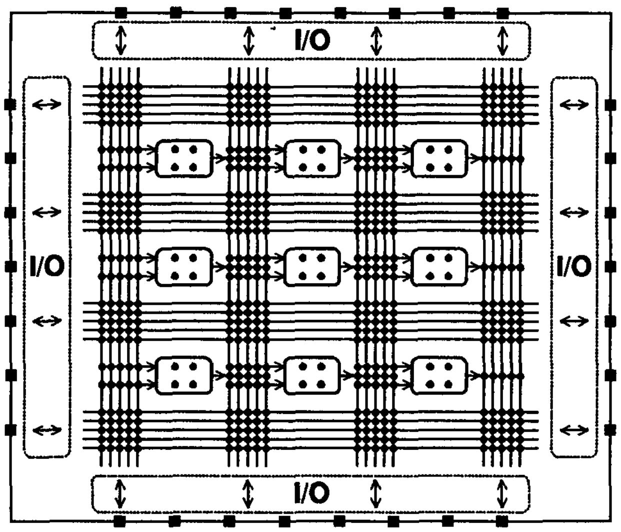


Figure 1: Part of a simplified typical FPGA architecture.

puts of the blocks are connected to the wires. Wires can also connect to other wires at some places. There are additional configuration switches within each block: in the figure, the four dots in each block represent the four bits of information required to specify which out of the set of all boolean functions of two inputs that block is to perform. There are special blocks around the periphery of the array, which interface it to the pins of the chip. Note that real commercial products are much more elaborate than this example.

The settings of the switches are stored in a configuration memory, which can be written to from software. By coding the contents of the configuration memory onto a genotype, an evolutionary algorithm can be used to automatically derive a circuit implemented on the FPGA to achieve a given task. To evaluate an individual's fitness, its genotype is expressed to produce a setting of the configuration memory; this is then applied to the chip, which instantiates the corresponding circuit. The behaviour of this real circuit is then evaluated to give a fitness score. This is sometimes called 'intrinsic' hardware evolution, as compared to the 'extrinsic' case where evolution is carried out using a simulation model, with only the final configuration being downloaded onto the chip. This paper addresses the issues associated with *intrinsic* hardware evolution.

Perhaps the most promising applications of evolved hardware are for high-speed pattern recognition and signal processing. These are domains where a hardware implementation is required because software is too slow, but yet conventional hardware design techniques are not well suited in some cases. Fitness evaluations could be extremely short, resulting in a rapid rate of evolution. Other possible applications in which traditional methods flounder are fault-tolerant design (Thompson, 1995b), design under low power/area constraints, and ill-defined tasks such as autonomous mobile robotics (Thompson, 1995a).

3 Embodiment versus Abstraction

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

The use of abstract models simplifies design by allowing some aspects of reality to be ignored, but the properties of the real hardware that have been 'abstracted away' must be suppressed: they must not be allowed to influence the final behaviour of the designed circuit. For example, a designer engaged in digital design does not need to think about the analogue behaviour of the transistors, but considers them as ON/OFF switches. To allow circuits designed at this level of abstraction to work in reality, the transistors must always be kept in either the ON state or the OFF state except during short transient periods while they are switching between them. Steps must be taken to ensure that these transients do not influence the overall behaviour of the system as predicted by the designer's digital model. In synchronous design, this is done by compartmentalising the system into modules which only communicate on the ticking of a clock: the transient dynamics are localised within each module and die away before the module is allowed to influence the rest of the system. Thus, both the spatial organisation and the dynamical behaviour of the circuit are constrained in order to support the designer's abstract model. This not only applies to digital design, but to all design methodologies: none can proceed far with detailed physical descriptions of the dynamical behaviour of the elementary components (e.g. transistors) before abstraction is necessary.

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

3.1 Spatial Structure

As well as to support abstract models, modularity arises in designs according to the way in which the problem was decomposed. Humans typically use some sort of "divide and conquer" strategy; whether the decomposition is a functional one or a behavioural one (Brooks, 1991), the final structure arrived at usually has modules echoing that decomposition. The *evolutionary* process could also benefit from a kind of modularity, such that different phenotypic characteristics can be improved semi-independently by genetic mutations (Wagner, 1995). However, these 'modules' are not necessarily reflected in the phenotype circuit's spatial or topological structure: for example, they could correspond to *basins of attraction* in state-space. These are currently open questions, and are intimately tied to the study of *morphogenesis*: the expression of genotype by a growth process. (Thompson et al. (1996) considers these issues in more detail.) The important observation here is that any kind of modularity that *is* appropriate to evolution could be very different to that arising in design by humans. Conventional notions of modularity should not be imposed upon an evolving circuit; although human and evolutionary decomposition may sometimes coincide, evolution should ideally be allowed to explore the full range of possibilities without inheriting limitations from humans.

3.2 Temporal Structure

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

In particular, there no longer needs to be an *enforced* method of controlling the phase (temporal co-ordination) in reconfigurable hardware originally intended to implement digital designs. The phase of the system does not have to be advanced in lock-step by a global clock, nor even the local phase-controlling mechanisms of asynchronous digital design methodologies imposed. The success of pulse-stream neural networks (Murray et al., 1991; Murray, 1992), where *analogue* operations are performed using *binary* pulse-density signals, gives a clue that allowing the system's phase to unfold in real-time in a way useful to the problem at hand can add a powerful new dimension to electronic systems: time. Mead's highly successful analogue neural VLSI devices (e.g. the 'silicon retina'), 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 (Mead, 1989).

Once unessential dynamical constraints have been removed, we are relying on evolution to arrange the dynamics of the system so that the overall behaviour is appropriate. Without evidence, it would be too much a leap of faith to expect this to work. For example, surely a complex continuous-time recurrent network of digital logic gates is doomed to fall into useless uncontrollable high-frequency oscillation? The remainder of the paper addresses exactly this issue. If evolution is able to control the dynamics of such seemingly unmanageable networks, then there is great potential for it to exploit usefully those very dynamics it has harnessed — dynamics normally precluded from contributing to the way an electronic system operates.

4 Experiment 1: Simulation

The task was to evolve a network of high-speed logic gates, in a simulation loosely based on the structure of an FPGA, to oscillate regularly at a much slower timescale than the gate delays. The task was simple, but success would demonstrate evolution's ability to manipulate dynamical time-scales.

The number of logic nodes available was fixed at 100, and the genotype determined which of the boolean functions of Table 1 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 2. (Node 0 was a special 'ground' node, the output of which was always clamped at logic zero.) This encoding is based on that used by Cliff, Harvey, and Husbands (1993). 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.

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 (Miczo, 1987), with real-valued time being held to double precision accuracy. An equiv-

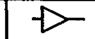
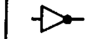
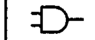
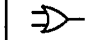
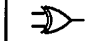
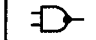
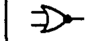
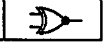
Name	Symbol
BUFFER	
NOT	
AND	
OR	
XOR	
NAND	
NOR	
NOT-XOR	

Table 1: Node functions.

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

Table 2: Genotype segment for one node.

alent 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 n^{th} transition occurring at time t_n seconds, then the average error in the time spent at each level was calculated as :

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

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 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 (Goldberg, 1989) with elitism, truncation of the five least-fit individuals, and linear rank-based selection. Population size was 30, single-point crossover probability 0.7 and mutation rate 6.0×10^{-4} per bit.

Figure 2 shows that the output of the best individual in the 40^{th} generation (Figure 3) 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 have constraints applied to the dynamics.

The evolved oscillators produced spike trains rather than the desired square wave. (A square wave could have been produced by the addition of a toggling flip-flop to the output, but this did not arise within the 40 generations.) Probing internal nodes indicated that *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. 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 Figure 2. Beating does not just occur at a few nodes, but is distributed throughout the network: I was unable to attribute functions to particular subnetworks of components. Indeed, a heuristic graph-partitioning algorithm revealed that there are no significant 'modules' (cohesive subnetworks) in the topology of the circuit. The use of beats is essentially a continuous-time phenomenon: a highly effective solution that would not have been available via conventional methods. It is an example of evolution putting to use the complex dynamics it has harnessed.

5 Experiment 2: Reality

In the simulation experiment, we saw that evolution could control the dynamics of a noise-free purely digital network. However, real recurrent continuous-time logic networks are far from noise-free and digital. Even though the gates are nominally digital, they are essentially very high gain analogue amplifiers, and this can become a more appropriate description of their behaviour. Here, we repeat the previous experiment, but with all fitness evaluations taking place on a *real* FPGA. There is no simulation of the circuit, just real semiconductor physics.

The apparatus is shown in Figure 4. The FPGA is from the forthcoming XC6200 family of Xilinx, Inc. (Xilinx, 1995). The details of this device are confidential at

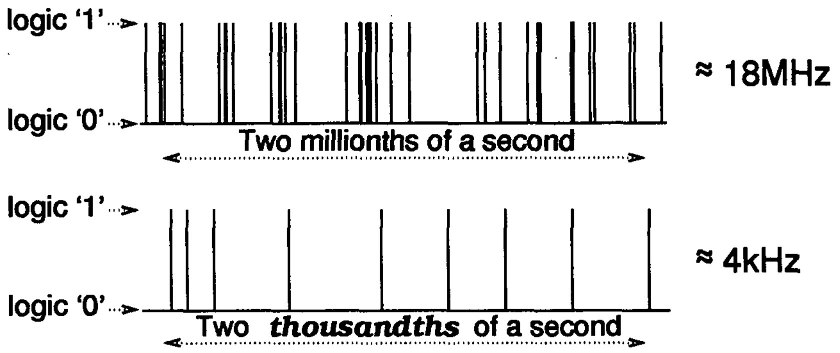


Figure 2: Output of the oscillator evolved in simulation. (Top) Best of the initial random population of 30 individuals, (Bottom) best of generation 40. Note the different time axes. A visible line is drawn for every output spike, and in the lower picture each line represents a single spike.

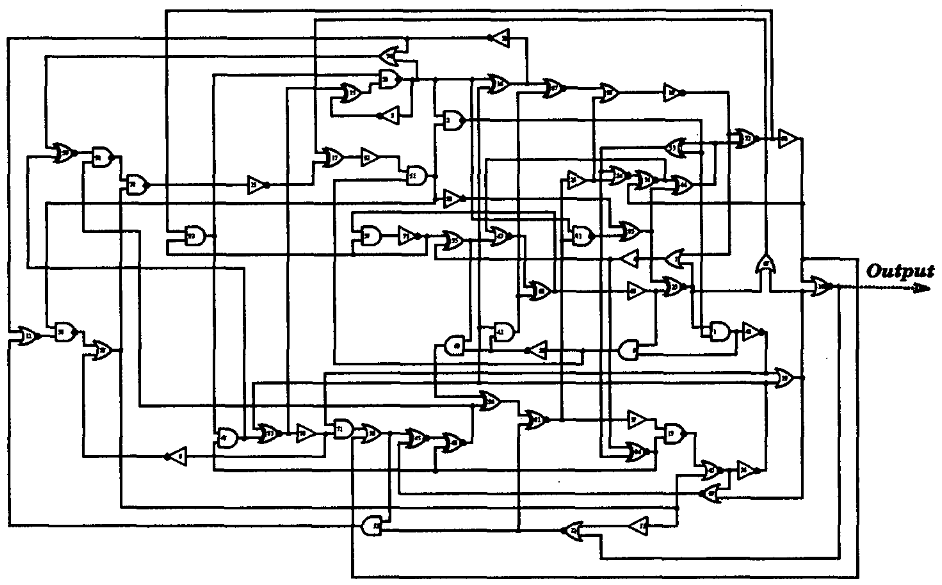


Figure 3: The 4kHz oscillator evolved in simulation (unconnected gates removed, leaving the 68 shown).

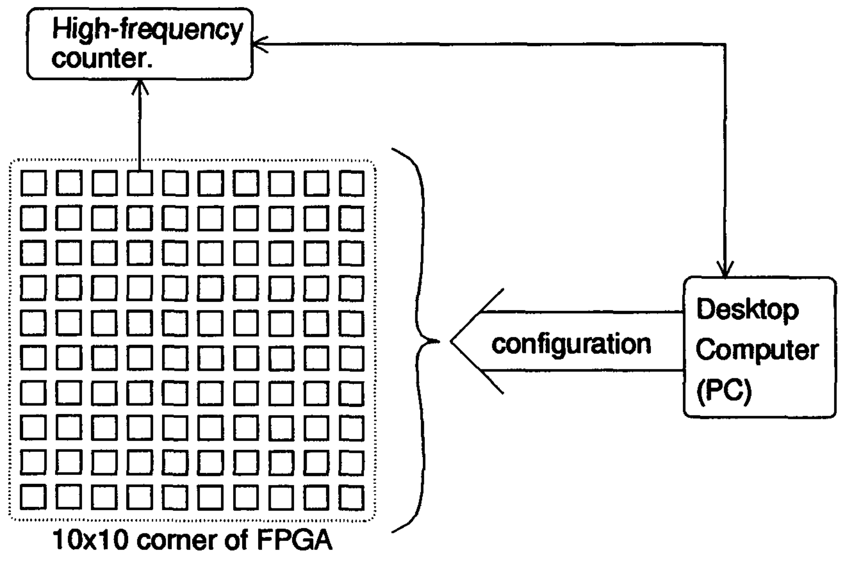


Figure 4: The experimental arrangement for oscillator evolution with the real FPGA.

the time of writing, but it is sufficient to say that a subset of its functionality was used such that it appeared quite similar to the simplified example given in Section 2. Only a 10×10 corner of a much larger chip was used, and the necessary configuration bits were directly encoded onto the linear bit-string genotype, of length 1800 bits. The GA was the same as above, except that the population size was 50 and there was no truncation in the rank selection. The bitwise mutation rate was set so as to give an expected mutation rate of 1.45 per genotype (see Thompson et al. (1996) for the theory behind this).

Each individual, once downloaded as a configuration onto the real hardware, was given a single one-second evaluation. A particular block at the edge of the array was designated to be the output (as shown), and monitored by a high-frequency counter. This counter (an HC11 micro-controller) is capable of counting positive-going logic transitions at a rate of up to 1MHz. Its count at the end of the one-second evaluation was taken as a direct measurement of frequency. Taking the fitness to be the inverse error as before, we have fitness $= 1/|f - n|$, where f is the desired frequency and n is the number of positive transitions counted over one second. If no transitions at all were counted, then the fitness was set to zero, and if $f - n$ was zero then the fitness was declared to be a million.

The initial population was hand-seeded by randomly generating a large number of circuits, inspecting their output by eye using an oscilloscope, and simply selecting those for which the output was not always constant. Starting from this same initial population, the GA was run for 40 generations for each of the target frequencies f = 10Hz, 1kHz, 100kHz, 1MHz. The measured frequency of oscillation of each individual over the course of each run is shown in Figure 5. As the diagram clearly shows, the population quickly converges on the desired frequency. The population is *not* just converging upon an individual which was already in the initial population: the maximum fitness in the population, as well as the mean, increases over time.

Although using the counter to measure frequency seems crude, there is evidence that the evolved individuals do indeed display the desired behaviour: if they are allowed to run for *two* seconds instead of one, then the number of counts measured during the second second is very close to that measured during the first. This implies a certain degree of regularity in the oscillation. Repeated evaluations over extended periods of time produce very similar readings.

How is the behaviour being achieved? Visual inspection of the output on an oscilloscope shows that *all* of the evolved solutions produce very high frequency waveforms of some kind, but that this high frequency component is not crossing the digital logic threshold, so is not being registered by the counter. Only at just the desired frequency is the signal making an excursion across the threshold long enough to be captured by the counter. All that can be seen on a cheap oscilloscope is a blur of high-frequency activity centered around an analogue voltage a short distance away from the logic threshold, with occasional faint traces of large-amplitude excursions.

Evolution really does seem to have tuned the circuit as a continuous-time analogue dynamical system, even though the manufacturers of the chip had discrete logic in mind. As an analogue system, the solution appears to be found more easily than in the equivalent purely digital noise-free simulation of the previous section. The physical characteristics of the FPGA have been exploited without any constraint whatsoever on the configurations allowed. The four runs show evolution manipulating the overall dynamics of the system on a timescale varying over five orders of magnitude.

6 Experiment 3: Application

The previous experiments show that *in principle* the evolution of unconstrained dynamical electronic circuits is possible, but the practicability of the approach in real applications is another matter. This section deals with a simple (but not 'toy') application intended as a first step into the pattern recognition domain identified earlier. Imagine a stream of binary data bits being sent down a telephone line coded as 10kHz and 1kHz tones: the task here is to evolve a circuit to decode the tones back to binary **0** and **1**. As for the oscillator experiment, a 10×10 corner of the real FPGA was used, but now with a single input as shown in Figure 6.

The GA and encoding were exactly as above, except that a random initial population was used, and the per-bit mutation rate was set such that the expected mutation rate was 2.7 per string (arrived at through trial and error). To evaluate an individual circuit, five half-second bursts of each tone were applied to the input in a random order (different each time). During each half-second interval, an analogue integrator integrated the voltage on the FPGA's output pin. At the end of each interval an analogue-to-digital conversion was performed on the integrator's output to give a measure of how high, on average, the voltage had been over the half-second. The integrator was then reset before the next tone. An individual's fitness was the absolute value of the difference between the mean integrator reading averaged over the cases when the input was 1kHz, and the mean reading when the input was 10kHz. Thus, the task was to maximise the difference in output voltage between the two cases, ideally producing a steady 0 Volts as soon as one tone is applied, and a steady 5 Volts immediately the other is present.

As shown in Figure 7, initially the circuits were as bad as possible, producing a steady 5V output in both

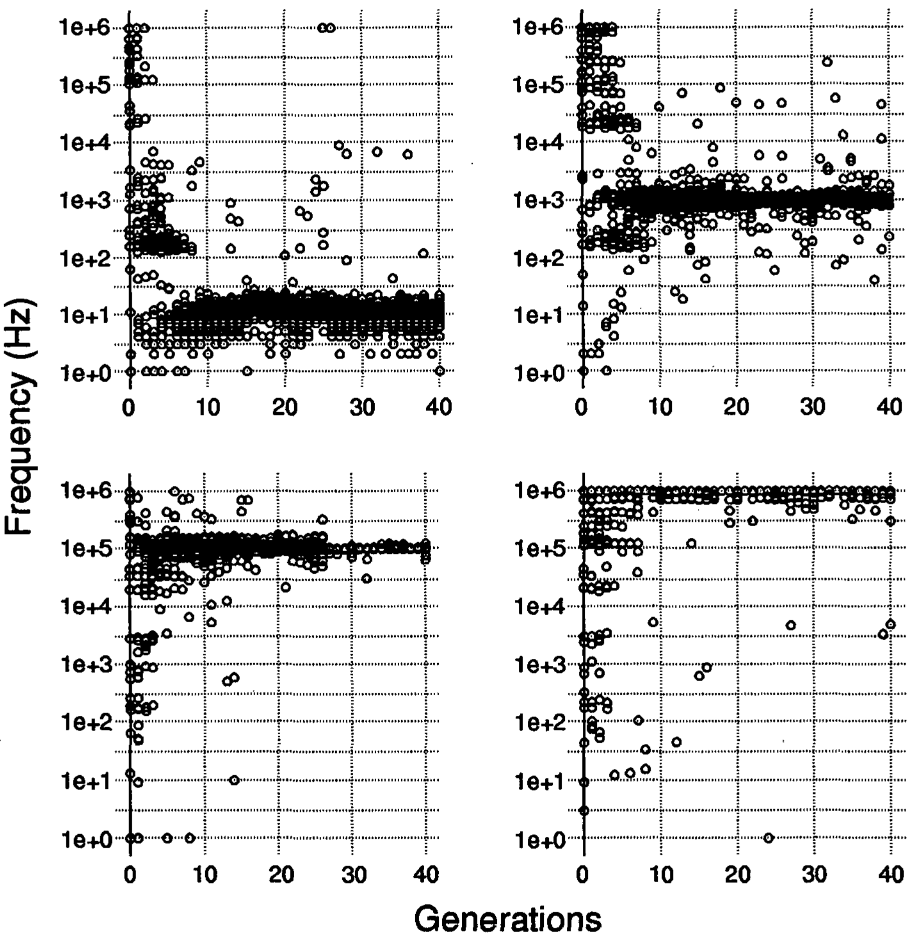


Figure 5: Frequency of oscillation of circuits in the population over the GA run (real FPGA hardware). The objectives were: Top left - 10Hz; Top right - 1kHz; Bottom left - 100kHz; Bottom right - 1MHz. Individuals with constant output (frequency = 0Hz) are not shown. Frequencies higher than 1MHz appear as exactly 1MHz due to the limited rate of the counter. Where many points are overlaid, they appear as one. Note that the frequencies are shown on a logarithmic scale. Each of these runs took just a few minutes to complete, due to the shortness of the evaluations.

cases. Over the generations, a wide range of partial solutions were seen (one is shown), usually involving complex waveforms with high-frequency components, and not respecting digital logic levels. After 3500 generations, the near-perfect solution shown was arrived at: eventually only tiny glitches (invisible in the figure) in the otherwise-perfect output were present, too small to be measurable by the integrator.

The task is simple, but it would be very difficult for a designer to solve this problem in such a small area, and with no external components. Normally, bulky external timing capacitors or an externally applied clock would be used: here evolution has arranged for the natural dynamics of the silicon to do everything on-chip. The complexity of the output seen for the intermediate solutions suggests that the circuit is exploiting rich and unusual dynamics even when producing the final orderly digital-looking output encouraged by the fitness function. This has been facilitated by the fitness evaluations being continuous-time analogue measurements.

The circuit operates over a 10 Celsius temperature range, but begins to gracefully degrade beyond this: many physical properties of VLSI chips are temperature sensitive. Ongoing research aims to tackle this problem by evaluating each individual circuit on multiple chips, each at different temperatures. This will not slow down evaluations, because the chips can be used in parallel. The objective is to evolve circuits that work over the full range of conditions (both external, like temperature, and internal properties peculiar to individual chips) which they could encounter in an industrial application.

7 Conclusion

Intrinsic hardware evolution has been presented as justifying a shift in the way electronic systems are conceptualised, towards an unconstrained exploitation of the physical dynamics. The first experiments evolv-

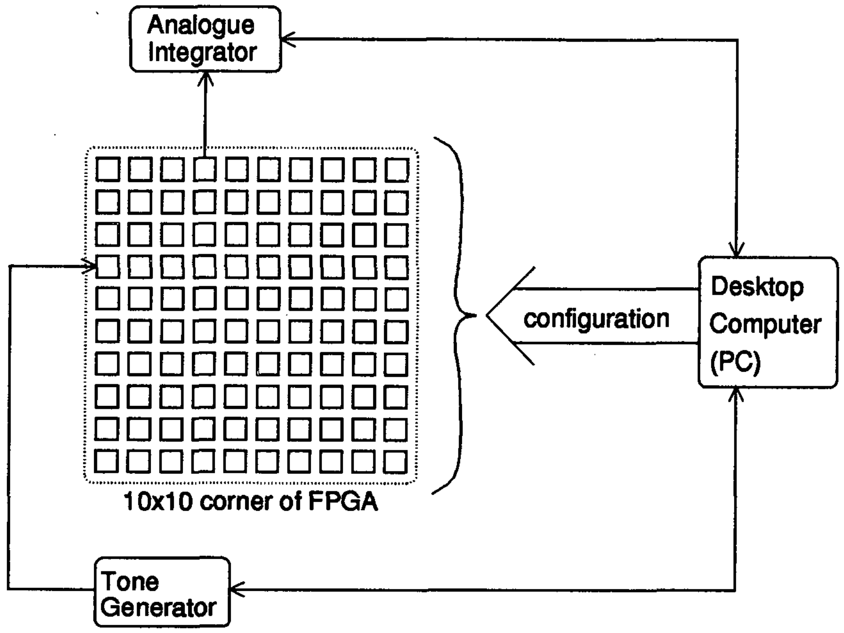


Figure 6: The experimental arrangement for evolution of the tone discriminator with the real FPGA.

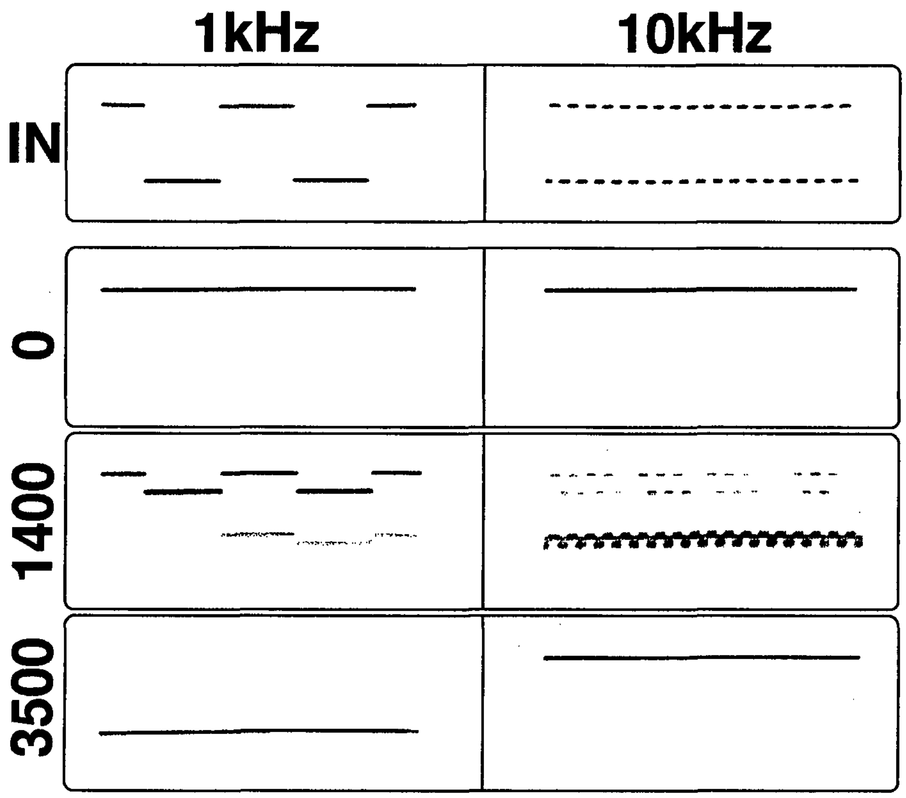


Figure 7: Photographs of the oscilloscope screen as the tone discriminator evolved on the real FPGA. (Top) the input waveforms; (Below) the corresponding output waveforms of the best of the population after 0, 1400 and 3500 generations. The probes were directly attached to the pins of the FPGA (the integrator is *not* part of the system).

ing FPGA configurations strongly reinforce the argument. High-speed pattern recognition and signal processing have been identified as particularly suitable applications; clearly there is much fundamental research still to be done.

Acknowledgements

Grateful thanks to Phil Husbands, Dave Cliff, Inman Harvey and Xilinx Scotland. Funding is by a D.Phil. scholarship from the School of Cognitive and Computing Sciences.

Bibliography

Hirst, A. J. (1996). (This paper gives an overview of the literature.) Notes on the evolution of adaptive hardware. In *Proc. of the 2nd Int. Conf. on Adaptive Computing in Engineering Design and Control (ACEDC96)*, Univ. of Plymouth UK. **http://kmi.open.ac.uk/~monty/evoladaphwpaper.html**

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*, 139–159.

Cliff, Harvey, & Husbands. (1993). Explorations in evolutionary robotics. *Adaptive Behaviour*, *2*(1), 73–110.

de Garis, H. (1991). Genetic programming : Artificial nervous systems, artificial embryos and embryological electronics. In Schwefel & Männer (Eds.), *Parallel Problem Solving from Nature : Proc. of PPSN I*, pp. 117–123. Springer-Verlag.

de Garis, H. (1993). Evolvable hardware: Genetic programming of a Darwin Machine. In Reeves, Albrecht & Steele (Eds.), *Artificial Neural Nets and Genetic Algorithms - Proc. of the Int. Conf. in Innsbruck, Austria*, pp. 441–449. Springer-Verlag.

Goldberg, D. E. (1989). *Genetic Algorithms in Search, Optimisation & Machine Learning.* Addison Wesley.

Hemmi, H., Mizoguchi, J., & Shimohara, K. (1994). Development and evolution of hardware behaviours. In Brooks & Maes (Eds.), *Artificial Life IV*, pp. 371–376. MIT Press.

Higuchi, T., et al. (1993). Evolving hardware with genetic learning: A first step towards building a Darwin Machine. In *Proc. of the 2nd Int. Conf. on the Simulation of Adaptive Behaviour (SAB92).* MIT Press.

IMP, Inc. (1994). *IMP50E10 EPAC Electronically Programmable Analog Circuit: Preliminary product information sheet.*

Mange, D. (1993). Wetware as a bridge between computer engineering and biology. In *Proc. of the 2nd Eur. Conf. on Artificial Life (ECAL93)*, Brussels, pp. 658–667.

Marchal, P., et al. (1994). Embryological development on silicon. In Brooks & Maes (Eds.), *Artificial Life IV*, pp. 365–366. MIT Press.

Mead, C. A. (1989). *Analog VLSI and Neural Systems.* Addison Wesley.

Miczo, A. (1987). *Digital Logic Testing and Simulation.* Wiley New York.

Mizoguchi, J., Hemmi, H., & Shimohara, K. (1994). Production genetic algorithms for automated hardware design through an evolutionary process. In *IEEE Conference on Evolutionary Computation.*

Murray, A. F., et al. (1991). Pulsed silicon neural networks - following the biological leader. In Ramacher, & Rückert (Eds.), *VLSI Design of Neural Networks*, pp. 103–123. Kluwer Academic Publishers.

Murray, A. F. (1992). Analogue neural VLSI: Issues, trends and pulses. *Artificial Neural Networks*, *2*, 35–43.

Thompson, A. (1995a). Evolving electronic robot controllers that exploit hardware resources. In Morán et al. (Eds.), *Advances in Artificial Life: Proc. of the 3rd Eur. Conf. on Artificial Life (ECAL95)*, pp. 640–656. Springer-Verlag LNAI 929.

Thompson, A. (1995b). Evolving fault tolerant systems. In *Proc. of the 1st IEE/IEEE Int. Conf. on Genetic Algorithms in Engineering Systems (GALESIA'95), IEE Conference Publication No. 414*, pp. 524–529.

Thompson, A., Harvey, I., & Husbands, P. (1996). Unconstrained evolution and hard consequences. In Sanchez & Tomassini (Eds.), *Towards Evolvable Hardware.* Forthcoming: Springer-Verlag LNCS.

Wagner, G. P. (1995). Adaptation and the modular design of organisms. In Morán et al. (Eds.), *Advances in Artificial Life: Proc. of the 3rd Eur. Conf. on Artificial Life (ECAL'95)*, pp. 317–328. Springer-Verlag LNAI 929.

Xilinx, Inc. (1995). *The XC6200 FPGA Family Advance Product Information Sheet.*

York, T. A. (1993). Survey of field programmable logic devices. *Microprocessors and Microsystems*, *17*(7), 371–381.