The 2nd Intl. Conf. on Simulation of Adaptive Behavior, 1992 (SAB92)

Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine

Tetsuya HIGUCHI[1] Tatsuya NIWA[1] Toshio TANAKA[1]
Hitoshi IBA[2] Hugo de GARIS[1] Tatsumi FURUYA[1]
1) Computational Models Section, 2) Machine Inference Section,
Electrotechnical Laboratory (ETL),
1-1-4 Umezono, Tsukuba, Ibaraki, 305, Japan.
higuchi@etl.go.jp, tel +81-298-58-5868, fax +81-298-58-5871

Abstract

This paper introduces an idea which the authors hope and believe will create not only a new branch of Artificial Life, but may also serve as the basis for a radically new approach to electronic and computer design. The idea can be expressed in two words, "evolvable hardware". Software configurable hardware, such as programmable logic arrays, are on the market which accept a bit string instruction which is used to configure or "wire up" a hardware circuit to give it a desired architecture. This can be done a large number of times. By treating the bit string instruction as a Genetic Algorithm "chromosome", one has the means to evolve hardware. This paper reports on a successful experiment to simulate the evolution of a GAL16Z8 hardware chip, which solves the 6-multiplexor problem. The concept of evolvable hardware can be used to build "Darwin Machines", i.e. special devices which are capable of adapting to their environment by modifying their own hardware architecture using genetic learning techniques.

Keywords: Evolvable Hardware, Programmable Logic Arrays, Software Configurable Hardware, Genetic Algorithms (GAs), Genetic Programing (GP), Animats, Biots (Biological Robots), Artificial Life, Generic Array Logic, GAL16Z8 chip, Darwin Machines, Multiplexor Problem, Boole.

1 Introduction

One of the major aims of Artificial Life is to build autonomous artificial creatures (animats or biots (biological robots)) which can adapt themselves to their environment using such techniques as reinforcement learning. This paper introduces an initial step towards building a biot which is capable of adapting its own architecture, i.e. its own hardware, according to rewards received from the environment. More specifically, this paper shows how the hardware architecture of an electronic circuit can be evolved directly, using genetic learning techniques. The concept of evolvable hardware was proposed by one of the authors and was given the label "Darwin Machine" [de Garis 91, 92, 93]. Initially, the concept of a Darwin Machine probably sounds like a dream, but recent hardware technologies (e.g. programmable logic devices) offer excellent capabilities which make it feasible to build evolvable hardware.

Programmable logic gate arrays consist of *logic cells* and their *interconnections*. A logic cell performs some logical function using flip-flops, and combinations of AND and OR gates. Its logic function can be selected by specifying special bits in the device. The interconnections are also specified in this way. Thus, application specific gate arrays are easily produced in a field-programmable manner, i.e. the user, and not the manufacturer, determines the architecture and hence the behavior of the device. This flexibility has made programmable logic gate arrays very popular.

The programmable bits of these devices are referred to as *architecture bits* in this paper. Once they are specified, the Boolean functions of the logic cells, their interconnections, and thus the total logical behavior of the gate array is determined. Therefore, in order to evolve a gate array chip which adapts to a given environment, we only have to find the desirable architecture bits. The hardware is given environmental inputs which determine its output according to the current architecture bits. By receiving rewards from the environment, the architecture bits can be evolved.

This paper reports on the first experiment conducted toward the longer term aim of building Darwin Machines, using a programmable logic chip, GAL16Z8, manufactured by Lattice Corporation (Hillsboro, Oregon). Initially we deal only with simpler combinatorial (rather than sequential) circuits. A software simulation of the evolution of the GAL16Z8 chip is presented, which solves the 6-multiplexor problem.

2 The "Darwin Machine" Concept

This section provides a brief introduction to the concept of a Darwin Machine (or DM). As mentioned earlier, the work presented in this paper introduces the idea of evolv-

able hardware, and takes a first step towards achieving the longer term goal of building Darwin Machines. *A Darwin Machine is a device which evolves its own architecture directly in hardware.*

Darwin Machines may be useful because,

1. The execution speed of the evolved system will be extremely fast (at least three orders of magnitude faster than a software implementation) because the result of adaptation is the hardware structure itself.
2. Fault tolerant design is realized because evolvable hardware can change its own structure in the case of hardware error or environmental change.

The Darwin Machine concept is a very broad one, which may be applicable to the evolution of virtually any form of hardware. It is also a very powerful concept, because evolved systems can be very complex, yet be functional, if they are evolved successfully. It may be possible in the future to evolve circuits more complex than are humanly designable. If so, Darwin Machines may eventually form the basis of a new electronics and computer industry. They may also provide the means to build biots with a greater behavioral repertoire than is possible today. DMs may end up playing a vital role in the field of biotics (i.e. artificial creature construction).

Since the Darwin Machine concept is very broad, it is likely that many different options will emerge on how DMs can be implemented. For example, two of the authors of this paper, Higuchi and de Garis, have diverse conceptions on possible DM architectures and application areas.

Higuchi's conception, for example, involves the application of reinforcement learning ideas to teach a biot's evolving hardware circuits to behave appropriately in an unknown environment. Signals coming from the environment would serve as reinforcers to reward or punish the biot's actions. These reinforcers would act directly upon the hardware by changing its architecture. This would enable the biot to respond in real time to its environmental signals, because its response would be implemented directly in hardware, not software.

de Garis's conception of a Darwin Machine is to implement a Genetic Algorithm [Goldberg 89] directly in hardware, and to use the DM to build/evolve useful electronic circuits, for example, fully connected neural networks, or (on a longer term basis) artificial nervous systems for biots. de Garis has helped establish a new "Brain Builder Group", which intends (starting 1993 in another lab) to use Darwin Machines to evolve neural circuit modules (and their connections) for biots with 10, 100, 1000,... behaviors.

3 Genetic Learning for Evolvable Hardware

Wilson's *Boole* is a classifier system which learns difficult multiplexor problems [Wilson 87]. He solved a 6-multiplexor problem with Boole.

Figure 1 shows a 6-multiplexor. One of 4 binary input channels is selected by the 2 address bits, to become the output channel. For example, if the input 4-vector (I1,I2,I3,I4) is (0,0,1,0), and the address bits (A1,A2) are (1,0), then I3(=1) is connected to the output OUT.

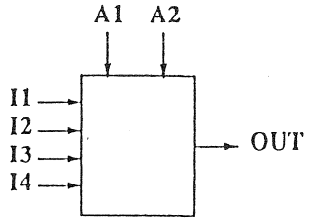


Figure 1: A 6-multiplexor

In *Boole*, rules necessary to construct the multiplexor are learned in the form of IF-THEN rules. The experiment in this paper also deals with multiplexor functions. However, in contrast with *Boole*, the knowledge acquired through adaptation is not IF-THEN rules, but a hardware design (as specified by its architecture bits). This section describes the basic hardware of the GAL16Z8 chip, and the genetic learning scheme employed for its evolution.

3.1 The Architecture of the GAL16Z8 Chip

The GAL16Z8 chip is a programmable logic device manufactured by Lattice Corporation. Earlier programmable logic chips were bipolar devices called PLAs (Programmable Logic Arrays), and were widely used in hardware systems. PLAs, however, can only be programmed once, because the logical functions are "burned in", and unchangable.

To address this weakness of the bipolar approach, PLAs based on UVCMOS (ultra violet CMOS) technology were introduced to enable reprogramming. However, ultra violet reprogramming took 20 minutes. This made it an expensive step in the manufacturing process. This delay would also make hardware evolution inefficient.

Although the GAL16Z8 chip is also a CMOS device, its erase (or reprogramming) time is only 50 milliseconds (24,000 times faster than a UVCMOS PLA). In addition to this, the GAL16Z8 chip can be reprogrammed a large number of times. Its logical functions are also more versatile than earlier programmable chips. Therefore, the GAL16Z8 chip is a promising tool to test the concept of evolvable hardware.

Figure 2 shows a logic diagram of the GAL16Z8 chip. It consists of a fuse array and 8 logic cells (called OLMCs, i.e. Output Logic Macro Cells). Pins 1 to 9 usually serve as inputs to the device, and pins 12 to 19 are the outputs of the 8 logic cells.

The fuse array is used to determine the interconnections between device inputs and logic cells, as well as to specify the logic cells' AND-term inputs. If a link on a particular row of a fuse array is set to "connected" (shown as a dot in Figure 2), then the corresponding (vertical) input signal is connected to the row. The input signal then becomes one of the AND-term members.

For example, Figure 2 shows the product-term input pattern to the logic cell, OLMC 19. The input signals

P2 and P3 are connected to the first row because the corresponding fuse links are "connected". These inputs generate the product-term, P2*P3 (i.e. P2 AND P3). Similarly, the input signals P4 and P5 are connected to the 8th row and generate the product-term, P4*P5. These product terms are then gated with "product term disable" bits, one bit for each row. If the bit is off, the product term is allowed to enter into the logic cell. In Figure 2, the two "product term disable" bits are off for the logic cell OLMC 19. Therefore, P2*P3 and P4*P5 are entered into the OLMC19 cell.

The logic cell has the structure shown in Figure 3. It contains a flip-flop, an OR gate, various multiplexors, etc. The function of a logic cell is selected from 5 possible operational modes, such as "registered device" or "combinatorial output device", by specifying 3 special bits (SYN, AC1, AC0). The polarity of the output is also selected by an XOR bit. By setting these bits at SYN=1, AC0=0, AC1=0 and XOR=1, the function of OLMC 19 in Figure 2 is determined to be "P2*P3 + P4*P5". The internal structure is selected as shown in Figure 4.

Thus, an arbitrary Boolean function can be specified by determining both the links of the fuse array pattern, and the logic cell function.

3.2 Genetic Learning Scheme

We now describe how we simulated the evolution of the GAL16Z8 chip. Firstly, the representation of the bit string chromosome used to configure the chip is described. This is followed by a description of the genetic learning algorithm that was used to perform the simulation.

3.2.1 Chromosome Representation

We simulated the hardware evolution of the solution to the 6-multiplexor problem. The solution takes the form of a Boolean function, and is evolved with one logic cell

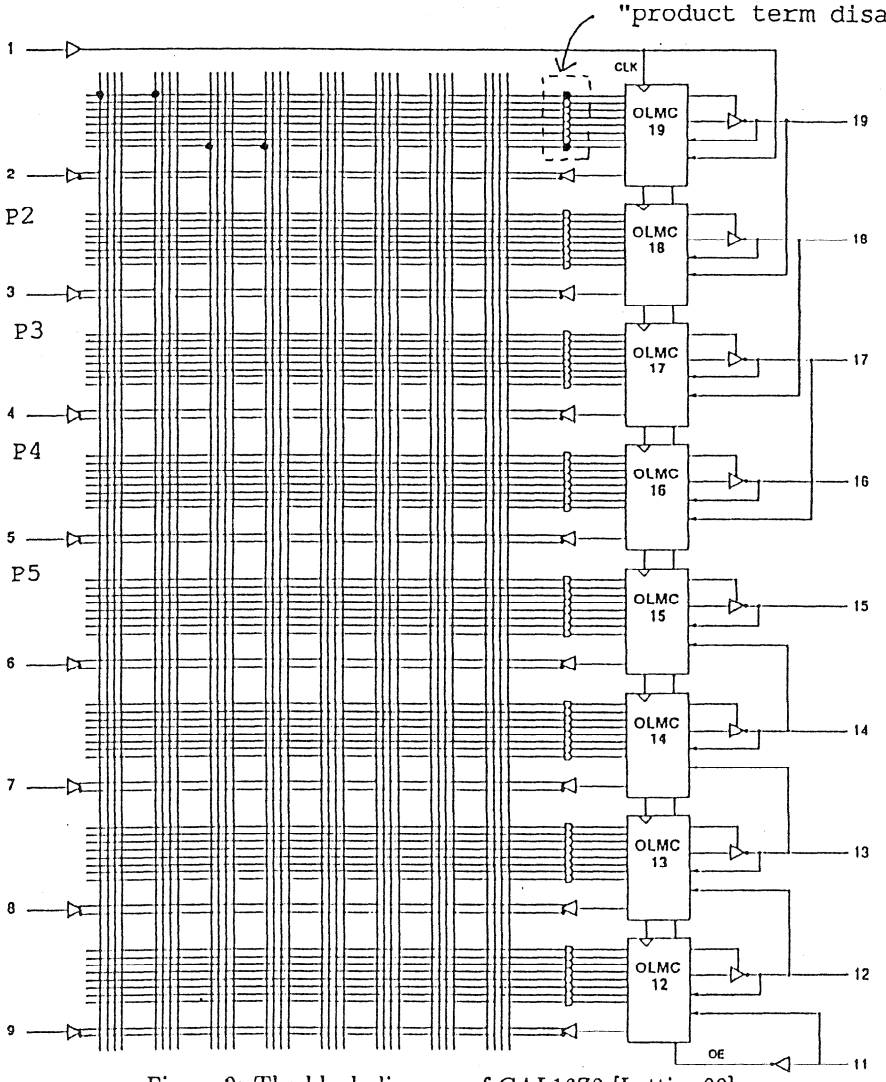


Figure 2: The block diagram of GAL16Z8 [Lattice 90]

of the GAL16Z8 chip. This boolean function only requires 6 input bits. Therefore, we do not need to use all the chip's logic cells. Only one logic cell and its corresponding fuse array were evolved, by representing them in one chromosome.

The resulting chromosome had 108 bits. Its representation is explained as follows. The fuse array uses 8 rows and 12 columns (as shown in Figure 2). The twelve columns correspond to the 6 input signals, because each input signal has both a positive and negative input. There are 8 rows as inputs to a logic macro cell. Therefore, the fuse array representation needs 96 bits (12*8).

In addition to this, 12 bits are needed for the logic cell definition, i.e. 8 bits are needed to specify the product term disables, and 4 bits are needed to specify the selection of the logic cell behavioral mode. Thus, the length of the chromosome is 108 bits.

Given this chromosome and a 6-bit input pattern, we can configure the GAL16Z8 chip's structure and then produce the output corresponding to the 6-bit input. The output is then examined to see whether the software configured chip gives the desired output or not.

3.2.2 Learning Algorithm

The genetic learning scheme employed in our experiments is shown in Figure 5. In practice, we simulated the evolution of the GAL chip using software. Instead of rewriting the GAL chip repeatedly, it is written only once, i.e. after the most desirable chromosome is found. We employed this policy at this stage because the current version of GAL chips do not guarantee an indefinite number of reprogrammings. This is one of the weaknesses of current PLGAs (Programmable Logic Gate Arrays) as a vehicle for evolvable hardware.

The GAL simulator in Figure 5 accepts inputs to the GAL, one chromosome at a time, and generates the output according to the structure specified by the chromosome. (The execution time of the GAL simulater is 100 micro seconds. If a GAL chip is actually used, the execution time is 10 nano seconds, i.e. 10,000 times faster) Next, the GAL output enters the fitness evaluation module for the 6-multiplexor problem. Once the fitness values have been determined, the conventional genetic algorithm operators (selection, mutation, crossover) are applied to the chromosome population as shown below.

Step 1 Calculate the fitness of each chromosome. The 6-multiplexor problem uses all 64 possible input patterns as the training set. To measure the fitness of a chromosome, all 64 6-bit input vectors are presented to the 6-multiplexor that it codes for. If the actual output bit from the circuit is the same as the desired bit, then the fitness score is incremented. The maximum raw fitness score is 64. The final fitness was defined to be the percentage of the 64 actual output bits which were correct.

Step 2 Select two chromosomes with a probability in proportion to their fitness values

Step 3 GA operations. Crossover and mutate the chromosomes selected in Step 2. Uniform crossover is executed.

Step 4 Replacement. Replace the entire old population with the new chromosome selected in Step 3.

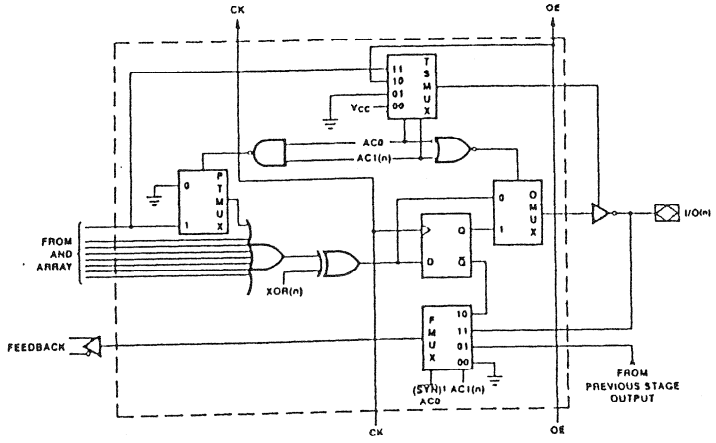
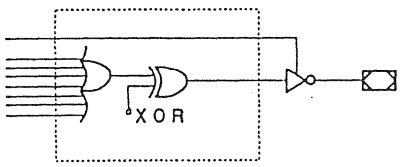


Figure 3: The structure of a logic macro cell [Lattice 90]



The operational mode:
"dedicated combinatorial output with programmable polarity"

Figure 4: The selected structure of OLMC19 [Lattice 90]

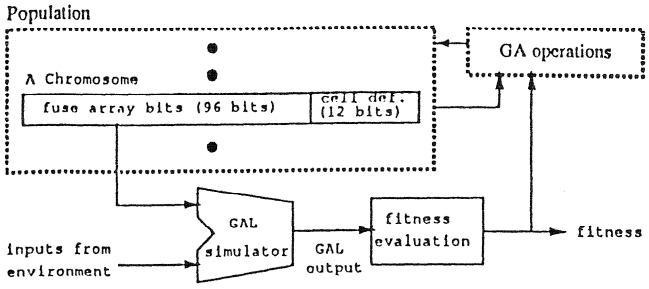


Figure 5: The genetic learning scheme for the experiment

4 Experimental Results

The experiment was conducted with a population of 100 chromosomes. The probability of uniform crossover was 20%. The probability of mutation was 0.1% per bit.

Figure 6 shows an example of the evolution, where the best and the average fitness of 100 chromosomes are shown. It attains the fitness of 100 % in about 2,000 learning cycles.

Figure 7 shows the on-line and off-line performance. Each is the average of 10 runs with different seeds for random number generation.

Figure 8(a) is a chromosome obtained as the result of the evolution. As mentioned earlier, a chromosome consists of fuse array bits, product term disable bits, and logic cell behavioral mode bits. Figure 8(b) explains what the chromosome of Figure 8(a) means. The first eight rows of the chromosome in Figure 8(a) represent the fuse pattern which corresponds to Figure 8(b). The bottom row of Figure 8(a) represents the product term disable bits and logic cell behavioral mode bits (i.e. XOR, AC0, AC1 and SYN).

With these specifications contained in the chromosome, the functions of each row of a logic macro cell is defined as shown in Figure 8(c). The first value of each row (i.e. 0 or 1) is that of the product term disable bits. If 0, the function is invalidated. The seventh row of Figure 8(c) is 1*(p2*$\tilde{p}$3*$\tilde{p}$4*p5*$\tilde{p}$5*p7) is also invalidated because the p5 inputs with both polarities are contradictory. Thus, the total function of a 6-multiplexor is represented as,

out=$\tilde{}$((p2*p3*$\tilde{p}$4)+(p2*$\tilde{p}$3*$\tilde{p}$5)+($\tilde{p}$2*p3*$\tilde{p}$6)+($\tilde{p}$2*$\tilde{p}$3*$\tilde{p}$7))

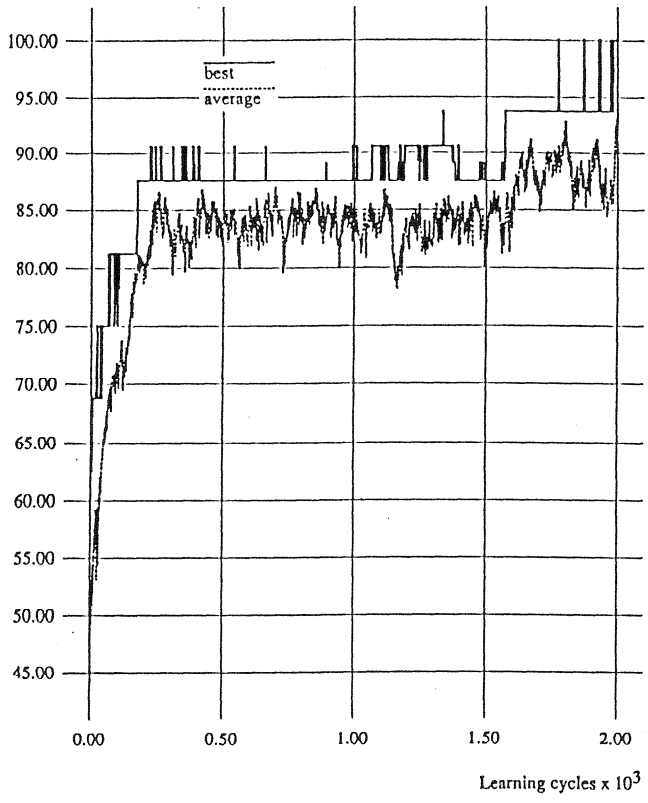


Figure 6: Learning of the 6-multiplexor problem

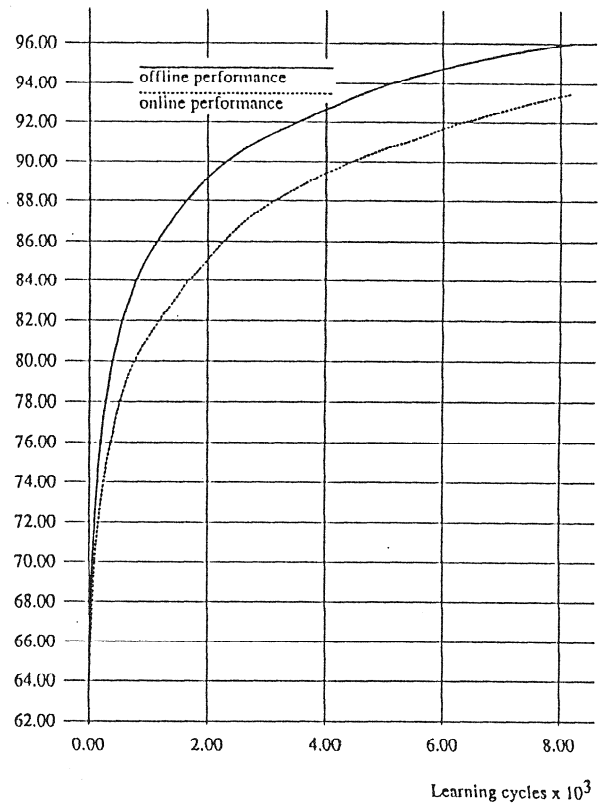


Figure 7: The on-line and off-line performance

The evolved circuit is shown in Figure 9.

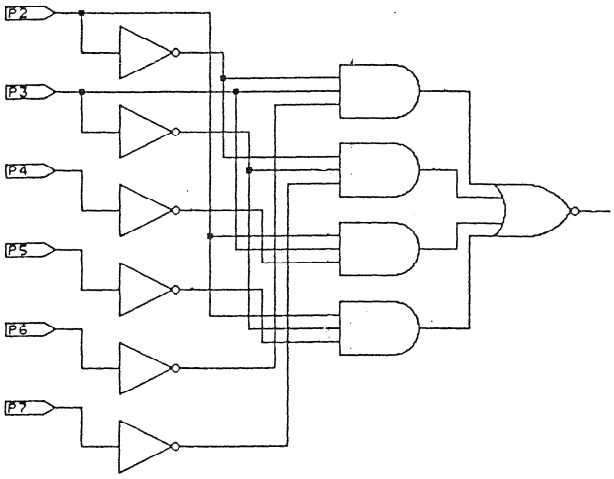


Figure 9: The circuit obtained by evolution

5 Discussion

The experiment in the previous section showed very promising results. In order to evolve more "intelligent"(or complex) hardware, however, two problems should be addressed.

The first is to deal with larger programmable logic devices. In the above experiment, we used only one of the eight logic cells contained in the GAL16Z8 chip. We need to evolve an entire GAL and much larger programmable devices called FPGAs (Field Programmable Gate Arrays), which can contain up to 20,000 gates. Even though FPGAs contain many more gates than a GAL chip, their hardware can still be configured by "programming". Therefore, it is expected that the genetic learning algorithms proposed in this paper can be extended to FPGAs.

It is necessary to process a large number of architecture bits, in order to evolve large devices. FPGAs require from 2000 to 30000 architecture bits to configure their circuits, while this experiment dealt with only 108 architecture bits. The current representation scheme of 108 bits is rather naive, and it is not confirmed yet that this representation scheme works well for a larger number of architecture bits. To evolve an entire GAL device, for example, roughly 2000 architecture bits are required. An experiment to do just this is currently under way. It may be necessary to devise a more precise representation for the architecture bits.

The second problem to be addressed is to evolve sequential logic circuits. The multiplexor function implemented on the GAL16Z8 chip is a combinatorial circuit, where no internal states exist, i.e. the output is a function of the input only. In order to evolve more intelligent hardware, sequential circuits become essential. Because GAL chips contain flip-flops and clock inputs to strobe states, sequential circuits can be implemented on GAL devices.

If the sequential circuit evolution succeeds, it may serve as a new learning paradigm for problems which have internal states. Those problems are very difficult for neural networks to handle, because neural networks can not express "states" effectively.

What the authors would like to do with sequential circuit evolution is the learning of finite state machines which could be used as the control structure for robots such as subsumption architecture [Brooks 89]. Adaptation with evolvable hardware may provide real-time control for animats.

```
Start of
chromosome
↓
100111111011        Fuse pattern              a function of each row
101011111110        10--01--11--11--10--11    1*(~p2* p3*~p6)
010110111111        10--10--11--11--11--10    1*(~p2*~p3*~p7)
111100100000        01--01--10--11--11--11    1*( p2* p3*~p4)
011011101111        11--11--00--10--00--00    0*( p4*~p4*~p5* p6*~p6* p7*~p7)
101101001111        01--10--11--10--11--11    1*( p2*~p3*~p5)
011010001101        10--11--01--00--11--11    0*(~p2* p4* p5*~p5)
111001100010        01--10--10--00--11--01    1*( p2*~p3*~p4* p5*~p5* p7)
111010111001        11--10--01--10--00--10    1*(~p3* p4*~p5* p6*~p6*~p7)
                    p2  p3  p4  p5  p6  p7
                ↑
            End     Product term
                    disable bits:  11101011

                    Behavioral Mode Select:
                    XOR:1, AC0:0, AC1:0, SYN:1

(a)                 (b)                       (c)
```

Figure 8: A chromosome bit pattern for the 6-multiplexor

6 Conclusions and Future Research

This paper has shown that programmable logic devices can be evolved using genetic learning. This paper has described a first step towards building Darwin Machines. Using only inputs to the device from the environment, and rewards resulting from the device's output, the architecture of the device can be reconfigured repeatedly to evolve desirable behaviours. This approach is the opposite of traditional hardware design, where the logical behaviours of the system are given first, and then the corresponding circuit design follows.

To give some ideas on possible DM architectures in the near future, two brief descriptions are given here.

Higuchi intends to extend the ideas of this paper by evolving more complex hardware circuits than the rather simple combinatorial circuits discussed in this paper. One such development Higuchi has just started, is to use GAL chips and Transputers to build a simple Darwin Machine which evolves such circuits as finite state machines, where the evolving chip is a GAL, and Transputers are used to measure the fitness of the evolving GAL chip, all done directly in hardware. The demonstration that evolvable hardware can be used to generate a finite-state machine in unknown environments, will have a significant impact on control design for animats.

For example, computer control of simple actions in animats may be implemented in Darwin Machine hardware, not by software. Before microprocessors became widely available, such control was implemented with sequential circuits in the form of finite state machines. With evolvable hardware, such finite state machines may be learned adaptively and replace the control programs. Current effort on learning finite state machines using evolvable hardware is reported in [Higuchi 92].

On the other hand, de Garis is thinking of using VLSI techniques to create new chips which are specifically designed with evolvable hardware concepts in mind. GAL chips for example, were not designed to be evolved, because the designers of GAL chips had no inkling of the concept of evolvable hardware. Figure 10 shows a tentative high level VLSI "evolvable chip" design, and how such chips could be linked in a two dimensional grid to implement a distributed (i.e. parallel) version of a Genetic Algorithm [e.g. Muhlenbein 91]. The electronic circuits being evolved would be fully connected neural network modules (GenNets) [de Garis 1993].

Building a Darwin Machine will require a lot of hard thinking and probably a lot of trial and error. For example, it is possible that the current candidate of FPGAs as the building block for Darwin Machines, may prove to be unsuitable. The manufacturers of FPGAs did not have the notion of evolvable hardware in their heads when their designers dreamed up the chips. Hopefully, a new generation of VLSI chip designers will be inspired by the notion of evolvable hardware and Darwin Machines, by creating new chips which are a lot more "genetically programmable".

Acknowledgement Authors thank Dr. Toshitugu Yuba for providing us opportunities to pursue this research.

References

[Brooks 89] Brooks, R. "A robot that works: Emergent behavior from a carefully evolved network" Neural Computation, 1989.

[de Garis 91] Hugo de Garis "Genetic Programming", Chapter 8 in book "Neural and intelligent system integration,"(ed. Branco Soucek) Wiley, 1991.

[de Garis 92] Hugo de Garis "Genetic Programming: Evolutionary Approach to Multistrategy Learning",

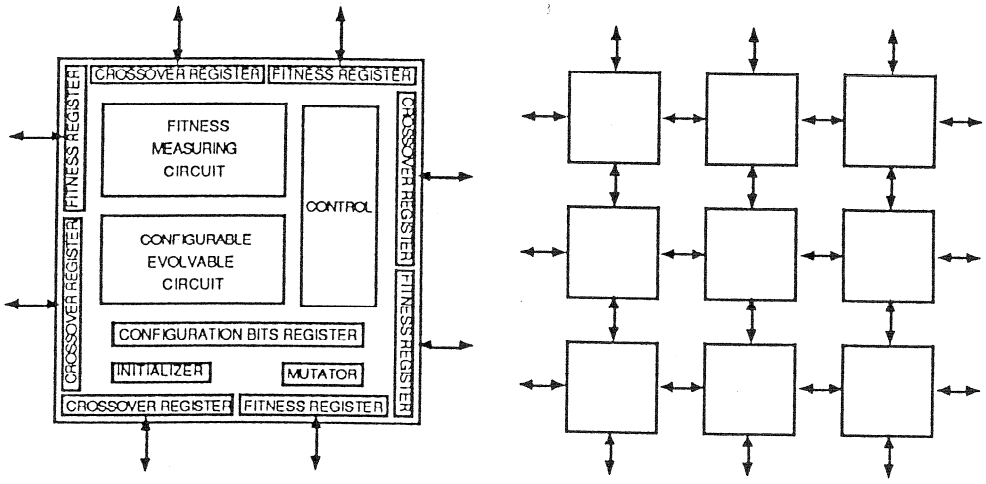


Figure 10: A distributed Darwin Machine architecture

Chapter 21 in book "Machine Learning: A Multistrategy Approach Vol.4"(ed. R.S. Michalski) Morgan Kauffman, 1992.

[de Garis 93] Hugo de Garis "Genetic Programming: GenNets, Artifificial Nervous System, Artificial Embryon", Wiley manuscript.

[Goldberg 89] Goldberg, D. Genetic Algorithms in Search, Optimization, and Machine Learning, Addison Wesley, 1989.

[Higuchi 92] Higuchi, T. et al. Genetic Based Hardware Evolution, Electrotechnical Lab. Tech. Report, Nov. 1992.

[Lattice 90] Lattice Semiconductor Corporation "GAL Data Book", 1990.

[Muhlenbein 91] Muhlenbein, H. "Evolution in time and space – the parallel genetic algorithm", in "Foundations of Genetic Algorithms", Morgan Kaufmann, 1991.

[Wilson 87] Wison,S. "Classifier Systemss and the animat problem" Machine Learning 2, 199-228, 1987.