A Divide-and-Conquer Approach to Evolvable Hardware

Jim Torresen

Department of Informatics, University of Oslo, PO Box 1080 Blindern
N-0316 Oslo, Norway
E-mail: jimtoer@idi.ntnu.no

Abstract. Evolvable Hardware (EHW) has been proposed as a new method for designing systems for complex real world applications. One of the problems has been that only small systems have been evolvable. This paper indicates some of the aspects in biological systems that are important for evolving complex systems. Further, a divide-and-conquer scheme is proposed, where a system is evolved by evolving smaller sub-systems. Experiments show that the number of generations required for evolution by the new method can be substantially reduced compared to evolving a system directly. However, there is no lack of performance in the final system.

1 Introduction

Evolvable hardware (EHW) has been introduced as a target architecture for complex system design based on evolution. So far a very limited number of real applications have been proved to be solvable by this new scheme. There are several reasons for this. One is the problem of evolving systems based on a long chromosome string. The problem has been tried solved by using variable length chromosome [1]. Another option, called functional level evolution, is to evolve at a higher level than gate level [2]. Most work is based on fixed functions. However, there has been work in Genetic Programming for *evolving* the functions [3]. The method is called Automatically Defined Functions (ADF) and is used in software evolution.

Both gate and function level of evolution have been applied to real applications. Simulations of data compression using function level evolution indicates performance comparable to other compression methods like JPEG compression [4]. The scheme is designed for implementation in a custom ASIC device. A function based FPGA has been proposed for applications like ATM cell scheduling [5] and adaptive equalizer in digital mobile communication [6]. Except for the few real problems studied, there is a larger range of small and non-real problems, see [7, 8].

This paper presents some concepts from biological systems and how they can be applied into architectures for evolvable hardware to be used for real applications.

The next section introduces the aspects from biological systems influencing on evolvable hardware. Section 3 presents a new scheme for evolving hardware.

Results from experiments are given in Section 4, which are followed by conclusions in Section 5.

2 A Framework for Evolvable Hardware

2.1 The Inspiration from Nature

The idea behind evolutionary schemes is to make models of biological systems and mechanisms. From nature the following two laws seem to be present:

1. The *Law of Evolution.* Biological systems develop and change during generations by combination and mutation of genes in chromosomes. In this way, new behavior arises and the most competitive individuals in the given environment survive and develop further. Another expression for this law is *phylogeny* [9].

2. The *Law of Learning.* All individuals undergo learning through its lifetime. In this way, it learns to better survive in its environment. This law is also referred to as *epigenesis* [9].

The two laws are concerning different aspects of life and should be distinguished, when studying artificial evolution. Most work on genetic algorithms are inspired by the Law of Evolution, while artificial neural networks are considering learning. Biological memory is still not fully understood. However, a part of the memory is in the biological neural networks. An interesting approach to design artificial evolutionary systems would be to combine the two laws into one system. One approach to this is Evolutionary Artificial Neural Networks (EANNs), which adapt their *architectures* through simulated evolution and their *weights* through learning (training) [10]. Neural networks are based on *local* learning, while the evolutionary approach is not [11].

Concerning FPGAs, this approach could be implemented by evolving a configuration bit string into a network of cells. After evolution their connections (weights) become trained.

Multi-Environment Training. So far artificial systems have mainly been trained by a single training set or in limited environment. This is in contrast to biological organisms trained to operate in different and changing environments. When training in a limited environment, the system with maximum fitness is selected. This fitness value does not have to represent the system with the best generalization [11]. It would be an interesting approach to investigate if an artificial system trained to operate in one environment could be combined with a similar system trained in other environments. If the resulting system could be able to contain knowledge about the different environments it would be able to operate in various environments *without* retraining. A possible implementation could be by including a soft switching mechanism between control systems, when the system enters a new environment. This would be an interesting approach for e.g. a vacuum cleaner moving from one room to another.

Online Adaptation in Real-Time. Research on evolutionary methods have been based on one-time learning. However, there is a wish of being able to design on-line adaptable evolvable hardware. The hardware would then have to be able to reconfigure its configuration dynamically and autonomously, when operating in its environment [12]. To overcome the problem of long evolution time, local learning should be investigated. Further, a time switching approach between learning and performing could be investigated.

An Appropriate Technology for Biological Modeling. Reconfigurable hardware like FPGAs are slower than microprocessors. However, a highly interconnection ratio is possible. These aspects are similar to those in the biological neural networks, where the neurons are massively interconnected. However, their speed of operation is slow. This indicates that reconfigurable technology should be an interesting approach for solving problems, where the biological brain is superior to ordinary computers.

2.2 The Basic Building Blocks and Their Interconnections

An evolutionary system would have to be based on some kind of basic unit. The model of a multi-cellular living organism would be to use a cell, which is able to reproduce itself by duplicating the full description of the complete organism. This corresponds to developing an individual (phenotype) from a chromosome (genotype). One cell could then be used to start reproduction to be carried out until the full organism is populated with cells [13]. Then, a specialization phase starts, where each cell interprets one piece of the chromosome depending on the location in the system. The major problem of this approach – named embryological development, is the tremendous amount of memory required within each cell for storing the complete chromosome. Thus, simpler logic gates or higher level functions have been used as the basic building blocks for evolution.

Most research on evolvable hardware is based on gate level evolution. There are several reasons for this. One is that the evolved circuit can be partly inspected by studying the evolved Boolean expressions. However, then the complexity of the evolved circuit is limited. Functional level evolution has been proposed as a way to increase the complexity. However, so far experiments have only been undertaken using ASIC chips. In the work presented in this paper, it is of interest to study the use of commercial FPGAs to higher than gate level evolution. Table 1 lists the possible combinations of low level building blocks and their interconnections.

Combining flip-flops[1] and feed-back connections have not been applied in EHW. One of the reasons for this is probably the difficulties and time consuming fitness evaluation of each individual. However, including flip-flops could be required to solve more complex problems by EHW.

[1] Flips-flops are essentially logic gates with feed-back connections, but are here listed as a special building blocks as they are in FPGAs.

Feed-back.-conn.	Flip-flops	Logic Gates	Applic.	Comments
No	No	No	-	
No	No	Yes	+	Applied for evolution in PLD and FPGA.
No	Yes	No	-	
No	Yes	Yes	+?	Could be applied to evolve delayed data networks.
Yes	No	No	-	
Yes	No	Yes	+	Applied by Thompson [14]
Yes	Yes	No	-	
Yes	Yes	Yes	+?	This would imply evolving state machines.

Table 1. Possible basic evolvable units and interconnections. "-" indicates a not applicable combination. "+" means that the combination has already been applied, while "+?" indicates a possible useful combination.

2.3 Local Learning

The internal operation of FPGAs available today is given by the configuration bit string loaded from an external source. This is in conflict to the wish that the operation of each cell in a device should only be based on local interactions with no global control. This is not a problem during normal operation, but rather during evolution. To make an FPGA device evolvable based on local interaction, it is possible to arrange sets of FPGA cells into subsets of cells. Each subset should be able to be evolved based on local interaction. In this scheme, the configuration bit string will not be changed during evolution, but rather the content of the internal registers. The application of local interactions would make local learning possible. The main challenge of this approach would be how to implement the genetic operators inside the FPGA.

Cellular Automata (CA) has been introduced to design systems based on local interactions. An experimental system based on FPGAs has been built for CA and local learning [9, 15]. The states of the cells in the system are evolved to oscillate between all zeros and all ones on successive time steps like a swarm of fireflies.

The long computation time has been one of the major problems for genetic algorithms, which is present also for evolvable hardware [14]. Evolving a 10x10 corner — a configuration string of 1800 bits, of a 64 x 64 array XC6216 FPGA required 2 – 3 weeks. The circuit was evolved to discriminate between two tones. The long evolution time even for a small circuit indicates that alternative evolution schemes should be investigated. The inherent parallelism in programmable hardware should be applied not only for operation, but also for the evolution.

The basic units in FPGAs are logic gates and 1-bit registers. Neural networks, on the other hand, have been shown to acquire at least 16 bits floating point values for inputs and weights for the best performance. Thus, further investigations should be conducted to find the optimal macro-cell size compared to the network size for evolution at a higher level than gate level. A large or complex cell structure would lead to a smaller number of such cells within a single FPGA device. Thus, the network will be small compared to using a simple macro-cell.

3 A New Approach to Hardware Evolution

Evolving systems for real applications of average complexity seem to be a near unobtainable task by the computer hardware available today. As mentioned earlier, several approaches have been suggested to overcome the problems, including variable chromosome length and function level evolution.

In this section, a scheme based on the principles described in the previous sections will be investigated. The method introduces a new approach to evolution called *increased complexity evolution*. The idea is to evolve a system gradually as a kind of divide-and-conquer method. Evolution is first undertaken individually on a large number of simple cells. The evolved functions are the basic blocks used in further evolution or assembly of a larger and more complex system. This may continue until a final system is at a sufficient level of complexity.

The main advantage of this method is that evolution is not performed in one operation on the complete evolvable hardware unit, but rather in a bottom-up way. The chromosome length can be limited to allow faster evolution. The problem of the approach would be how to define the fitness functions for the lower level sub-systems. However, for some applications it is possible to *partition* each training vector. Further, low level training vectors – e.g. speech *phoneme* recognition, can be used in the first evolution, followed by a higher level evolution using the evolved first level systems – e.g. to do *word* recognition. In this paper, the principles will be illustrated by a character recognition system. The evolution will be based on gate level, but can easily be changed to function level. The target EHW is an array of logic gates similar to that found in the Xilinx XC6200 and illustrated in Figure 1.

The array consists of n number of layers of gates from input to output. Except for layer 1, the Logic Gate (LG) is either a *Buffer*, *Inverter*, *AND* or *OR* gate. In layer 1, only *Buffer* and *Inverter* gates are available. Each gate's two inputs[2] in layer l is connected to the outputs of two gates in layer $l-1$. The function of each gate and its two inputs are determined by evolution. The evolution is undertaken off-line using software simulation. However, since no feed-back connections are used and the number of gates between the input and output is limited to n, the real performance should equal the simulation. Any spikes could be removed using registers on the output gates.

The application to be used in the experiments are the problem of recognizing characters of 5 x 6 pixels size, where each pixel can be 0 or 1. Each of the pixels is connected to *one* input gate. In addition to the 30 pixel inputs, two extra inputs are included as a bias of value 0 and 1, respectively. The output of the gate array

[2] *Buffer* and *Inverter* gates have only one input.

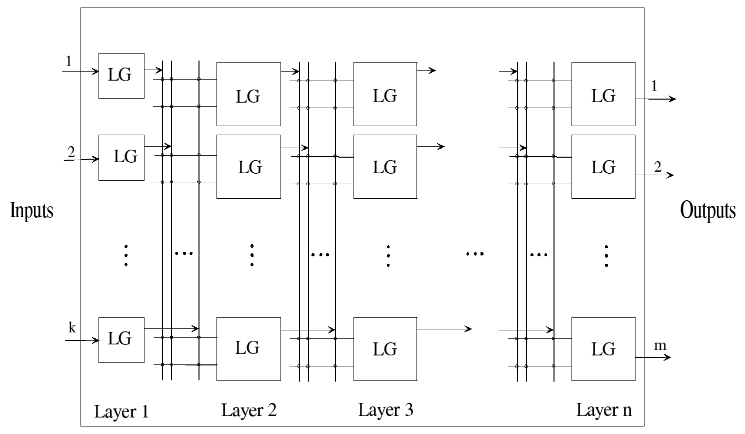

Fig. 1. An array of gates. "LG" indicates logic gate and can be *Buffer Inverter*, *AND* or *OR* gate.

consists of one output gate for each character the system is trained to recognize. During recognition, the output gate corresponding to the input character should be 1, while the other outputs should be 0. Thus, if the training set consists of m *different* characters, the gate array should consist of m output gates.

The aspect of *increased complexity evolution* is introduced by the way the evolution is undertaken. We compare evolving a system directly to evolving sub-systems. In the former case, the system is evolved to classify all training vectors in the training set. In the latter case, an evolved sub-system is able to classify a *subset* of the training characters. That is, each subsystem input *all* the 30 input pixels and *all* training vectors are applied during fitness evaluation. However, each subsystem has a limited number of *output* gates. In this way, the sub-systems are evolved without lack of generalization. The benefit is that each gate array is smaller and thus, should easier become evolved to perform the correct operation. For this application, the integration of sub-systems are straightforward by running them in parallel. For more complex applications, like speech recognition, a next level of evolution could be applied, where the sub-systems are the basic block in the evolution. The number of gate layers in the array is flexible and different numbers will be tested in the experiments. A large number would allow for more complex logic expressions. However, the chromosome becomes longer and obtaining a correctly evolved circuit could be more difficult.

Except for the output layer in the gate array, 32 gates are applied in each layer in the array. Thus, the *complete* system will be larger as the number of sub-systems increases. The main motivation for this work is to allow for evolution of complex systems and limiting the number of gates is not regarded as an important topic. The reason for this is that the main problem of today's research seems to be the lack of evolutionary schemes overcoming the chromosome length problem, more than the lack of large gate arrays.

The basic Goldberg style of GA was applied for the evolution with a population size of 50. For each new generation an entirely new population of individuals is generated. The mutation rate is 0.001. For each test, ten circuits were evolved and the circuit requiring the least number of generations was picked as the best.

4 Results

This section describes the results of evolving the character recognition system. Two separate experiments have been conducted. One with four characters and one with eight characters. A larger number of characters was also tested, but it was impossible to evolve a complete system in one operation.

Type of system	3 gate layers in array	4 gate layers in array
4 characters (A,B,C,D)	274	101
2 characters (A,B)	1	7
2 characters (C,D)	35	9

Table 2. The result of evolving a circuit for classifying four patterns, for three and four layers of gates in the array.

First, the experiment using four characters was undertaken. Table 2 shows the required number of generations used for obtaining a circuit that correctly classifies the patterns in the training set. In average, over ten times as many generations are required for evolving the system for recognizing four characters compared for each of the two sub-systems. Each of the sub-systems recognizes two characters. When other characters in the training set are input, the system is evolved to output the value 0 on each output.

The chromosome length is not very different for the two systems – 846 and 822 bits for the 4 and 2 character system[3], respectively. However, the number of outputs influencing the fitness evaluation is half the number for the complete system. Thus, a correct circuit can easier be found. In average, less number of generations is required for systems using four layers of gates compared to three. One layer of gates requires 384 chromosome bits.

Table 3 shows the results for the same kind of experimental setup, but with 8 patterns to be classified. As for 4 patterns, the number of generations required for evolving a classification system increases with the number of patterns. However

[3] This is for systems with four layers of gates.

Type of system	3 gate layers in array	4 gate layers in array
8 characters (A,...,H)	2709	3514
4 characters (A,B,C,D)	697	941
4 characters (E,F,G,H)	776	179
2 characters (A,B)	275	49
2 characters (C,D)	93	63
2 characters (E,F)	133	75
2 characters (G,H)	4	38

Table 3. The result of evolving a circuit for classifying eight patterns, for three and four layers of gates in the array.

as mentioned earlier, the final systems still has the same classification ability. It is interesting to observe in this experiment that for the systems evolved to classify many patterns, a larger number of generations is required for four layers of gates compared to three layers of gates. This is in contrast to the first experiment and may indicate that a shorter chromosome string is beneficial for a larger training set.

In this work, no separate test set has been used and thus, no concern about the overall generalisation and noise robustness has been included. The main goal has been to show that the evolution time can be reduced by dividing the systems into smaller sub-systems.

Both experiments show that the evolution time can be reduced by dividing the problem into sub-tasks to be evolved separately. This shows that the *increased complexity evolution* method should be promising for evolving complex systems for real applications. The scheme should, in future work, be tested for more complex applications and by using higher level functions.

5 Conclusions

This paper has introduced some aspects of biological systems to be applied for evolvable hardware. A scheme, called *increased complexity evolution*, is introduced. The method is based on sub-system evolution for the design of complex systems. A character recognition application is used to present one implementation of the scheme. It is shown that the number of generations can be drastically reduced by evolving sub-systems instead of a complete system.

Acknowledgements: The author would like to thank the group leader Tetsuya Higuchi and the researchers in the Evolvable Systems Laboratory, Electrotechnical Laboratory, Japan for inspiring discussions and fruitful comments on my preliminary work, during my visit there in November 1997.

References

1. M. Iwata et al. A pattern recognition system using evolvable hardware. In *Proc. of Parallel Problem Solving from Nature IV (PPSN IV)*. Springer Verlag, LNCS 1141, September 1996.
2. M. Murakawa et al. Hardware evolution at function level. In *Proc. of Parallel Problem Solving from Nature IV (PPSNIV)*. Springer Verlag, LNCS 1141, September 1996.
3. J. R. Koza. *Genetic Programming II: Automatic Discovery of Reusable Programs.* The MIT Press, 1994.
4. M. Salami et al. Lossless image compression by evolvable hardware. In *Proc. of 4th European Conf. on Artificial Life (ECAL97)*. MIT Press, 1997.
5. W. Liu et al. Atm cell scheduling by function level evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First Int. Conf., ICES 96*, pages 180 – 192. Springer-Verlag, 1997. Lecture Notes in Computer Science, vol. 1259.
6. M. Murakawa et al. Evolvable hardware for generalized neural networks. In *Proc. of Fifteenth Int. Joint Conf. on AI (IJCAI-97)*. Morgan Kaufmann Publishers, 1997.
7. J. Torresen. Evolvable hardware — A short introduction. In *Proc. of International Conference On Neural Information Processing (ICONIP'97, Dunedin, New Zealand)*. Springer-Verlag, November 1997.
8. J. Torresen. Evolvable hardware — The coming hardware design method? In N. Kasabov and R. Kozma, editors, *Neuro-fuzzy tools and techniques for Information Processing*. Physica-Verlag (Springer-Verlag), 1998. To appear.
9. M. Sipper et al. A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems. *IEEE Trans. on Evolutionary Computation*, 1(1):83–97, April 1997.
10. X. Yao and Y. Liu. Evolutionary artificial neural networks that learn and generalize well. In *Proc. of IEEE Int. Conf. on Neural Networks, Washington DC*. IEEE Press, 1996.
11. X. Yao and T. Higuchi. Promises and challenges of evolvable hardware. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First Int. Conf., ICES 96*. Springer-Verlag, 1997. Lecture Notes in Computer Science, vol. 1259.
12. T. Higuchi et al. Evolvable hardware and its applications to pattern recognition and fault-tolerant systems. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware: The evolutionary Engineering Approach*. Springer-Verlag, 1996. Lecture Notes in Computer Science, vol. 1062.
13. P. Marchal et al. Embryological development on silicon. In R. Brooks and P. Maes, editors, *Artificial Life IV*, pages 371–376. MIT Press, 1994.
14. A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In T. Higuchi et al., editors, *Evolvable Systems: From Biology to Hardware. First Int. Conf., ICES 96*. Springer-Verlag, 1997. Lecture Notes in Computer Science, vol. 1259.
15. M. Sipper. *Evolution of Parallel Cellular Machines: The Cellular Programming Approach.* Springer-Verlag, 1997. Lecture Notes in Computer Science, vol. 1194.

This article was processed using the LaTeX macro package with LLNCS style