Prototyping a GA Pipeline for Complete Hardware Evolution

Gunnar Tufte and Pauline C. Haddow
The Norwegian University of Science and Technology
Department of Computer and Information Science
O. S. Bragstadsplass 2E, 7491 Trondheim, Norway
gunnart@idi.ntnu.no, pauline@idi.ntnu.no

Abstract

In this paper a new approach to evolvable hardware is introduced termed 'Complete Hardware Evolution' (CHE). This method differs from Extrinsic and Intrinsic evolution in that the evolution process itself is implemented in hardware. In addition, the evolution process implementation, referred to herein as the GA Pipeline, is implemented on the same chip as the evolving design. A prototype implementation of the GA Pipeline is presented which uses FPGA technology as the implementation medium.

1. Introduction

Two main methods have been established for applying artificial evolution to the design of hardware systems. These are Extrinsic and Intrinsic evolution [1, 2]. In Extrinsic (off-line) evolution, the evolution process and the resulting evaluations are implemented in software. Each individual, design instance generated from the evolution process, is evaluated by a software simulation of the design described by the individual. When evolution is complete, the resulting design needs to be implemented in hardware.

Intrinsic (on-line) evolution, takes the design process closer to the real hardware in that each individual is tested out in hardware i.e. not an abstraction of it. Although the evolution process is still implemented in software, assessment of design quality is based on an actual implementation. When evolution is complete, the resulting design is already implemented in hardware.

In this work, a third method is introduced termed 'Complete Hardware Evolution' (CHE). Instead of having all (Extrinsic) or part (Intrinsic) of the evolution process in the host processor, a hardware implementation of the evolution process is used to drive evolution.

Both Intrinsic and CHE use a hardware implementation of each individual in the evaluation phase of the evolution process. This places a requirement of flexibility and, therefore, re-configurability on the implementation technology. Another important property of the target hardware is the number of possible re-configurations. A near unlimited number of re-configuration is desirable since the number of re-configurations required for a given design depends on the number of individuals in the population and the number of generations required to evolve a solution. This also places a requirement on configuration speed to prevent the evaluation phase becoming a bottleneck in the evolution process.

Full Programmable Gate Array Technology offers these features and is, therefore, chosen as the implementation technology for the prototype CHE based on a genetic algorithm presented herein.

2. Background and Motivation

The design productivity gap in the electronic industry is a well known fact. How can the design community utilise the design capacity that technology is offering and at the same time ensure it's correctness? To find solutions to the problem of developing large and complex designs, new design paradigms are required [3].

One possible solution is to turn away from traditional design techniques following the various design and testing phases and instead allow hardware to evolve until a correct solution is found. Hardware evolution is based on evolutionary techniques developed in the field of artificial intelligence [4]. The required circuit design may be evolved using software simulation to evaluate intermediate solutions — Extrinsic evolution, or by evaluating intermediate solutions implemented in hardware — Intrinsic evolution.

Extrinsic evolution, where the entire evolution process is implemented in the software simulator, evaluates the fitness of a given individual based on a model of the implementation technology. Modelling can be undertaken at different levels of abstraction. The more abstract a model is, the further it is from the reality of the underlying technology. Simulations based on more complex models come closer to the underlying technology and thus provide for a more realistic evaluation of the suggested design solution. However, the speed of the fitness evaluation of individuals reduces with increased complexity. For a given Extrinsic simulation, a trade-off may be made between accuracy and speed.

An example of a complex technology model used for Extrinsic evolution is Higuchi's multiplexer design [5]. This may be said to be a gate-level simulation based on on a specified device — in this case GAL16V8. Hemmi, on the other hand, uses a higher level of abstraction, using an RTL simulation for the evolution of behavioural HDL code [6]. The RTL description is off course independent of the target technology.

Abstracting away from the underlying technology may give a speed-up advantage when the goal is to explore a range of alternative high-level solutions for a given problem. However there is no guarantee that the solution identified can in fact be implemented. Also, where an accurate model of the hardware is required, it can be very time consuming and it may be that an accurate enough model is not possible. This speed problem is illustrated in Koza's evolving analogue circuit experiment [7] where a Spice net-list with physical models of the components was used in a software evaluation of the evolving design.

Intrinsic evolution moves the evaluation process into the actual target technology. Since each individual generated in the evolution process is implemented in hardware, evaluation of the design is also conducted in hardware. This not only increases the accuracy of the evaluation but provides the opportunity to explore the underlying technology's physical properties. It allows the evolution process to take advantage of all the physical parameters the target hardware has to offer to create a more optimal design solution. That is, instead of abstracting away from the technology, a design is evolved which is not only technology specific but specific to the chip used in the evolution process. During fabrication, a chip is deemed ok if it meets the fabricant's requirements. Over and above these requirements, it cannot be guaranteed that each chip has exactly the same physical parameters due to variations in the semiconductor material. During evolution of Thompson's tone-discrimination [8] circuit, the experiment pushed the circuit behaviour beyond that of a typical digital circuit. That is to say almost as far as analogue behaviour.

When using Intrinsic evolution for more general circuit design, that is where independent from fabrication variations is required, noise may be added to the design process. Chip independence using Intrinsic evolution is illustrated in Thompson's robot controller experiment [9]. In this work, evolution is based on the configuration of RAM to control a state machine. That is that each individual represents a RAM configuration.

For evolution to offer a viable alternative to traditional design techniques, speed is still a problem. For larger designs or designs with more complex fitness criteria, the software implementation part of Extrinsic and Intrinsic evolution processes can be said to be a bottleneck for sufficient speed-up. Therefore, a hardware implementation of the complete evolutionary process is proposed. To the authors' knowledge this is the first attempt at CHE in practice although Higuchi et al. have suggested, but not implemented, another possible genetic algorithm hardware implementation [5].

The discussion so far has focused on the speed of the evolution process and the accuracy of evaluation of intermediate results during the evolution process. Thus it compares the different evolutionary hardware methods for creation of given circuit design. However, what about other challenges facing the design community? Circuit failure may be caused by, for example, defect components or environmental factors such as temperature changes. To avoid these problems, self-repairing, fault tolerant hardware is required. External changes, although not considered as faults, can effect the lifetime of a design. These changes may effect the requirements of a system over time such that the original design is either no longer optimal or even no longer applicable. Functionality of a design can be adapted by external signals which trigger a change in the flow of signals through a design and thus change the designs functionality - this is called adaptive hardware. Adaptive hardware can be found today but adaptivity is limited to those functional changes and their implementation provided for at design time.

3. Complete Hardware Evolution (CHE)

In both Intrinsic evolution and CHE, each individual generated during the evolution process is implemented in hardware so as to enable accurate fitness evaluation. In Intrinsic evolution, this involves configuring the FPGA chip each time an individual is to be evaluated, testing the individual for fitness and returning the test results — fitness, to the host processor. In CHE, on the other hand, the evolution process is implemented on the same chip as the evolving circuit — that is the individuals under test. The problem that arises is that once the FPGA chip is configured for the evolution process, it is not desirable to reconfigure the complete chip each time a new individual is generated within the evolution process. This, off course, would involve reconfiguration of both the GA Pipeline and the new individual. Two possible solutions to this problem are suggested. The first is to set-off an area on the FPGA chip to be used as re-configurable memory that may be reconfigured internally within the chip or to use partially reconfigurable FPGA technology which allows selection of parts of the chip for configuration.

As in Intrinsic evolution, it is possible in CHE to exploit the physical parameters of the hardware. By implementing the evolution process in hardware, a speed up with respect to a software implementation may be expected. The question remains as to whether the fitness evaluation phase of the evolution process is faster in hardware than in software as this depends on a number of factors including the size of the hardware design to be evaluated and the level of abstraction assumed in the software model. However, comparing Intrinsic against Extrinsic evolution, since the evolvable hardware is configured by an individual in the evolution process, a close connection between the evolved hardware and the evolution process is expected to give an advantage.

The evolution process is responsible for configuration of the evolvable hardware, since the evolution process is never disconnected from the evolved hardware, it is possible to configure the evolvable part of the design at any time. This close connection between the evolving design and the fast evolution process driving it, possibilities within adaptive hardware.

If a CHE design which meets specified goals is created and the evolution process is designed to be triggered by changes in the surrounding hardware or the environment so as to restart evolution, then CHE may offer a flexible adaptive hardware design process. This is one of the goals of CHE. Since the evaluation of how well the proposed design solves the specified task is done by the fitness evaluation, exploration of a new design proposal could be started up when the current design no longer solves it's task satisfactorily. Using CHE, where the evolution process is on the same chip, adaption does not need to be decided at design time but the required change in the circuit is free to evolve dynamically with changing external factors.

Another goal of CHE is fault tolerant design. The use of adaptivity in hardware design can also be used as an approach to hardware with the ability to learn, where the exploration of solutions in the evolution process is used to adapt or learn a behaviour from a reference system. Such an approach for fault tolerance using Extrinsic evolution is the evolved duplication hardware for a welding arm done by Higuchi [10].

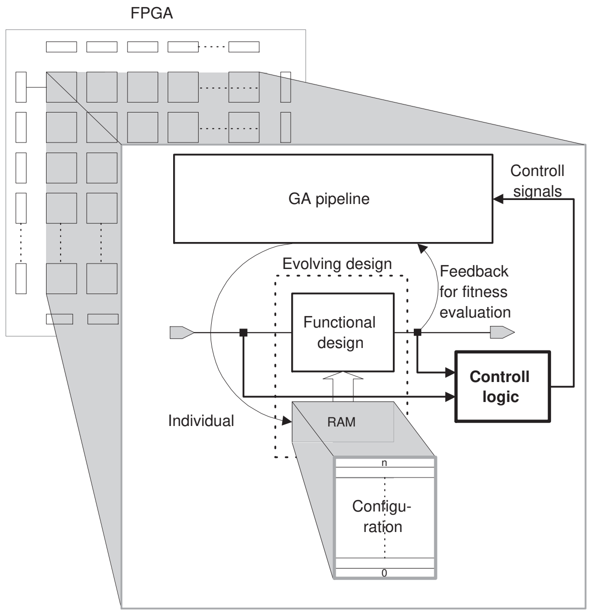

Figure 1. Complete Hardware Evolution

4. System View of the GA Pipeline

The main feature of the CHE implementation presented herein is the Genetic Algorithm (GA) pipeline which, as the name suggests, is a pipeline of modules representing the various functions or parts of the evolution process required to process a given generation, together with a feedback to handle the creation of further generations. In addition, control hardware allows for communication with the outside world — an external computer or with other units on a prototype board. The third main feature of the design is the evolving hardware itself — initially as a randomly generated design and finally the evolved desired design. External signals are connected to the evolving hardware to drive fitness evaluation of the current design. These results are then fed into the GA Pipeline for further processing.

Figure 1 illustrates the flow of individuals between the GA pipeline and the evolving design and the feedback of fitness values to the GA pipeline. The pads represent the external signals to the evolving hardware.

The main functions of the GA pipeline are selection, genetic operation, fitness and sorting which are

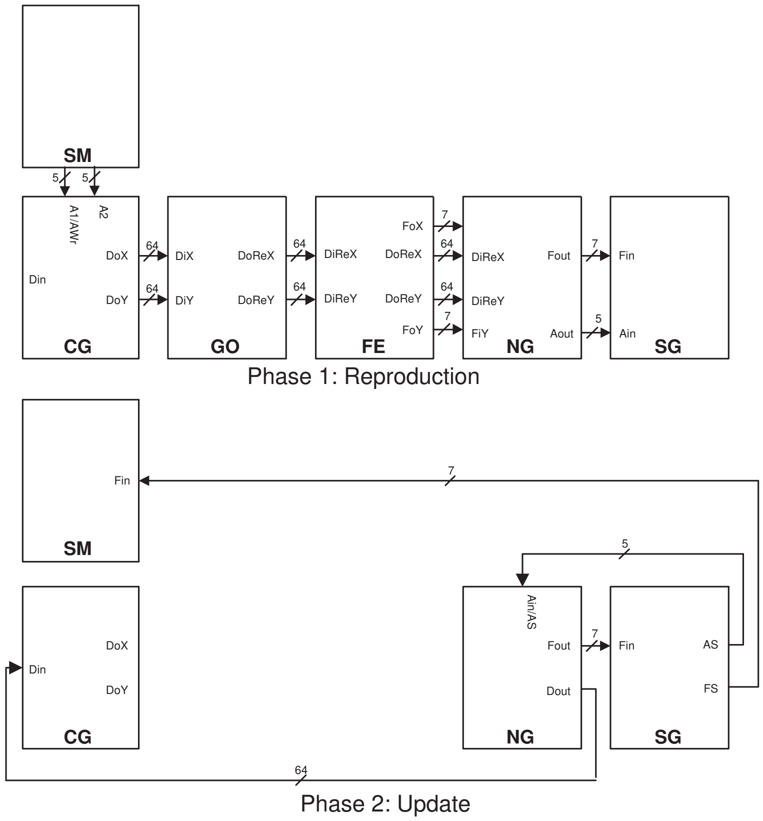

Figure 2. Genetic Algorithm Pipeline (GA Pipeline)

implemented in separate modules in the pipeline. In addition, two storage modules are included, one for the current generation and one for the new generation under creation. Control logic is responsible for applying the necessary control signals to run the GA Pipeline. Figure 2 illustrates the modular architecture of the GA Pipeline. As shown, there are two main phases in the pipeline: the first being the reproduction of individuals to form the next generation and the second, updating the current from the next generation. It should be noted that in theory such an update is not required and the next generation can automatically become the current generation after reproduction is complete. However, in practical terms sufficient storage is required for two generations due to the fact that the entire population of the current generation needs to be available for selection at the same time as created individuals need to be temporarily stored whilst awaiting completion of the next generation. In addition, to these two phases there is a initial phase which evaluates and sorts all the individuals in the initial random generated population.

4.1. Current Generation (CG)

Initially, the current generation holds a randomly generated population which is placed in the module before down-loading of the GA Pipeline architecture into the hardware. As stated, this population is replaced during the update phase of the GA Pipeline by the population in NG. The individuals of the population in CG are available for selection and processing to create the next generation.

4.2. Genetic Operators (GO)

The genetic operator module is responsible for performing the selected genetic operation(s) for the reproduction of selected individuals presented to the module. The options available for a given individual are crossover, mutation or no genetic operation. Selection of genetic operation or not, as the case may be, is controlled by the control logic. For the initial population no genetic operations are performed.

4.3. Fitness Evaluation (FE)

The fitness evaluation module performs fitness evaluation on the reproduced individuals presented to it from the GO module. Fitness evaluation is conducted through test vectors applied as external signals to the evolvable hardware, thus generating a fitness value for each individual evaluated.

4.4. Next Generation (NG)

The next generation module stores individuals for the new generation under creation by storing individuals arriving from FE. Each individual is stored in order of arrival and it's fitness value is forwarded to SG along with it's address in NG. As such, only a single bus is required for fitness values between NG and SG. In the update phase, the individuals in NG are rank selected from the ranking given in SG and forwarded to CG one at a time.

4.5. Sort Generation (SG)

To provide a full sort of the population, the sort generation module sorts each received fitness/address pair and store them in sorted order.

When NG is full, the next generation is complete, and all the fitness/address pairs have been received and sorted by SG. SG now holds a ranking of the individuals of the new population stored in NG. This ranking is used in the update phase to control transfer of the individuals from the unsorted NG such that the individuals are stored in CG in sorted order. As each individual from NG is selected and transferred to CG, it's fitness value is forwarded from SG to SM.

4.6. Selection Mechanism (SM)

The selection mechanism module is responsible for selecting individuals for reproduction based on their fitness value. A sorted population and the fitness value for each individual are available for the selection algorithm, if required. The information provided to SM is used to select individuals based on the implemented selection algorithm.

The fitness value of the individuals is provided to SM in the same order as individuals are received and stored in CG. In this way, unlike in SG, no address information for the respective individuals is required in SM due to the full sort of the population in SG.

SM selects two individuals from CG for reproduction based on the selection method implemented and their respective fitness values. This process continues until the next generation is complete. For the initial population, the module selects every individual in CG to be processed by the GA Pipeline.

5. GA Pipeline Implementation

One of the main goals in the implementation phase of the GA Pipeline was to achieve a flexible design which could easily be adapted, if necessary, to meet the requirements of different evolvable designs.

The nature of the pipeline is, in itself, a flexible structure providing a modular design where modules may be replaced or additional modules added where required. To make the design flexible with respect to the implementation technology, the goal was to implement all the modules in VHDL such that the design could be synthesised in the technology of choice. However, due to the requirements for two population stores, the memory requirements took up too much area on the chip. This problem was solved by using LogiBLOCK from Xilinx Foundation Series, the CAD tool used. Storage modules may be automatically generated using this tool so that properties of the technology, not available under the process of VHDL synthesis, can be utilised to create more efficient memory.

The choice of VHDL for the logic and control blocks makes these blocks flexible with respect to design changes. Simple changes like changing the genetic operator parameters may be achieved through a change to the respect parameter in the genetic operator module and re-synthesis of the module design. Other changes may require more fundamental adaption of the code such changing the sorting algorithm in SG.

For a genetic algorithm using FPGA technology, the genotype may be expressed in terms of RAM content or as configuration data. That is, the design that an individual represents is implemented in memory (RAM content) or logic (configuration data) during evaluation of the individual. This is also means that the final design is based on memory or logic respectively.

The genotype required to represent a RAM-based design will, in most cases, be much smaller than that required to represent logic or, in other works, configuration data. A smaller genotype means that each individual in the population is smaller and, as such, less storage is required to store the population. Also, since the size of the data buses in the pipeline are dependent on the size of the individual then a smaller genotype reduces bus width. However, it is hard to say whether a memory or logic genotype will lead to a larger or smaller area in the evolved design.

This previous discussion relates to area but what about the time used for evolution. As stated earlier, if partially reconfigurable technology is used then the evolving design alone may be reconfigured assuming a configuration genotype. For non-partially reconfigurable technology, the complete design must be reconfigured each time, thus increasing reconfiguration time.

In this work, non-partially reconfigurable FPGA technology is used due to the availability of a test board including a standard Xilinx XC4044XL chip [11]. In the current implantation, the maximal genotype size is 64 bits and may be of either logic or memory type. If the genotype is of type memory then this is sufficient for, for example, a robot controller [9]. Although it is possible to use 64 bits as configuration data, designs will typically require much larger genotypes as, for example, Thompson's tone-discrimination circuit which has a genotype of 1800 bits [8].

The current implementation of the GA Pipeline requires a minimum of a 7-bit bus to transfer fitness values and a 64 bit bus to transfer individuals. Selection of two individuals in parallel is controlled by the signal manager module and processing of these individuals continues in parallel through the pipeline, where possible. Since a genetic operation may involve either two — in the case of crossover or one individual — in the case of mutation or cloning, two copies of the mutation algorithm are included to enable parallel mutation. Once an individual is first chosen for crossover, the other individual will also be chosen for crossover since the crossover function requires two individuals. In the current implementation, single point crossover and mutation are implemented.

Two copies of the fitness evaluation process are included in the fitness evaluation module to enable fitness evaluation to be conducted in parallel. One phase of the pipeline that is not conducted in parallel is writing to memory which includes storage of the two individuals, storage of fitness values and write-back of the next generation. The choice of selecting and processing two individuals at a time was a balance between introducing parallelism to increase speed whilst limiting the increase in area required.

As described in section 4, the next generation is sorted before updating the current generation. This process was chosen to separate the sorting function from selection. Without prior sorting, the selection mechanism would be more complex as it would have to find the required individuals in the next generation. Separation of these two functions thus simplifies selection and supports testing of different selection algorithms, independent of sorting. In the current implementation, a parallel sorting algorithm is implemented in SG.

The fitness evaluation module may be redesigned, if required, to meet the fitness criteria for a given design. Both the fitness evaluation and the genetic operation module are asynchronous modules such that changes in design which effect the speed of these modules do not effect the other synchronous modules of the pipeline. This feature was an important consideration in the pipeline design so as to support use of the pipeline for more complex designs where the fitness function may become quite complex.

The storage modules limit the size of the population to the storage capacity within these modules. However, if space is available in the FPGA of choice, these storage modules may be extended to allow for larger populations without requiring a change to the GA Pipeline itself. The current GA pipeline implementation enables designs with a maximum genotype size of 64 bits to be evolved. Although this limit allows a wide range of designs to be evolved it may also be extended. This involves a change to the bus width in the VHDL coding of the GA Pipeline and re-synthesis of the modules before re-configuring the FPGA chip.

6. Testing of the GA Pipeline

Each module in the GA pipeline was simulated after synthesis to test for correct functional behaviour. The pipeline was built up and tested one module at a time. The control block was first tested on it's own and then as a part of the complete GA pipeline implementation.

A two-phase testing strategy was adopted. Interactive simulation at each level was conduced first to debug the design. This was followed by test bench simulation to control the testing process. Both of these strategies used Xilinx Foundation Series' Simulator.

After correct behavioural functionality was shown through simulation, the design was loaded onto the FPGA chip. It should be noted that the printed circuit board holding the FPGA chip was previously tested. The evolution process itself was not tested, being internal in the FPGA chip. However, each individual and it's respective fitness value was made available on the output pins so that these values could be read by the logic analyser. In this way, the evolution process could be followed.

7. Evolution of a multiplexer

The goal of this experiment was to find out if the GA Pipeline functioned correctly. In section 6 correctness testing of the GA Pipeline against our specification was conducted. However, to find out if the specification was correct, evolution of a circuit design was required. For

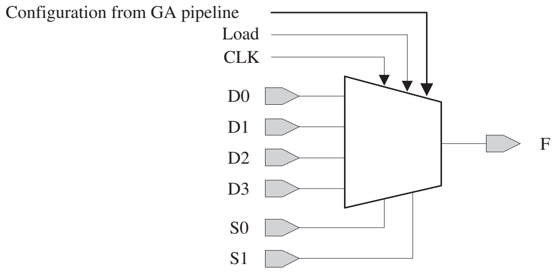

Figure 3. The Evolving Multiplexer: Interface to the GA Pipeline and External Signals

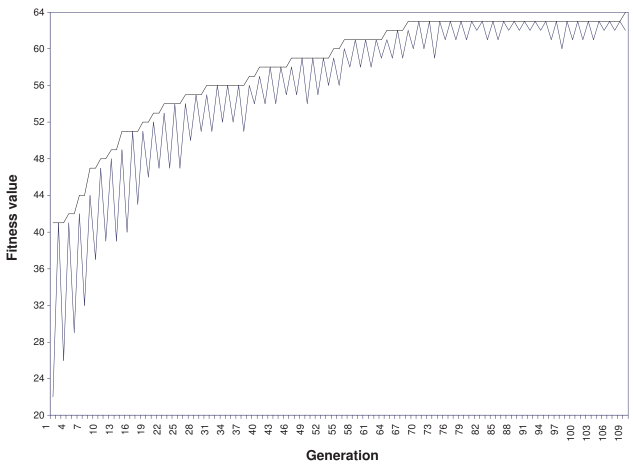

Figure 4. Fitness Profile for the Multiplexer Evolution

this purpose we chose a simple circuit, a multiplexer. When only functionality is considered, a multiplexer provides a static design, that is it has only one correct and predefinable solution. The simplicity of the multiplexer design allows fitness evaluation of the evolving multiplexer to be based on a complete test suite for the design.

During evolution of a simple design such as a multiplexer the evolution process may be monitored as it explores for a design solution. In this case, as stated, only correct functionality was sought. However, by extending the fitness criteria, other design features such as speed and area could also have been measured thus allowing the evolution process to explore the multiplexer design space.

The multiplexer chosen is a 4 data, 2 control input multiplexer. The genotype is a lookup table with 64 entries - one for each possible input combination. A randomly generated first generation was placed in CG, rank selection with elitism was implemented in SM and crossover and mutation rate were set to 0.6 and 0.39 respectively.

Figure 3 shows the multiplexer interface to the outside world. D0 - D3 and S0 - S1 are external input pads to and F, external output pads from, the FPGA chip. Configuration data from the GA Pipeline is controlled by CLK and LOAD. Each configuration represents an individual of the evolution process.

It was chosen to generate the test vectors for fitness evaluation of the multiplexer individuals from within the FPGA chip. The result of this evaluation was then fed to the FE module of the GA Pipeline for further processing. Once a phenotype with 100% fitness was found i.e. a correct design, the evolution in the GA pipeline was stopped. 100% fitness is equivalent to a fitness value of 64. The correct multiplexer design was evolved after 109 generations as shown in figure 4.

8. Ongoing Work

The size of the FPGA used in this prototype limits the possibility of evolving larger more complex designs where complexity may be measured in terms of the complexity of both the genotype required to represent the design and the fitness function required to evaluate the design. Limited chip size also effects the population size which limits the amount of genetic information available to help the evolution process towards an acceptable fitness level. Limited genetic information in the form of smaller populations may cause the evolution process to evolve towards a non-optimal solution.

A prototype cPCI card for the new Virtex XCV1000 FPGA from Xilinx [12] is under construction. Not only is this a larger FPGA chip but it also offers partial reconfigurability. This card will soon be available so that work in this area can continue.

One of the goals of CHE is to meet the needs for adaptive design. One such area where adaptivity is important is robotics. Two projects focusing on adaptive hardware using the GA Pipeline are in progress. The first is a robot controller and the second involves audio signal processing.

The robot controller, currently under development, is based on the design proposed by Thompson [9]. The goal is to evolve the robot controller, using the GA Pipeline presented, on a Xilinx XC4044XL FPGA chip [11]. This project provides the opportunity to test out the GA-pipeline for evolution of a FSM based design.

The robot controller is not based on a standard FSM but on Thompson's Dynamic State Machine (DSM).

The evolution process is placed on the robot together with the evolving robot controller in accordance with CHE principles. If a fault in the robot's hardware occurs e.g. a motor changes it's response with respect to applied signals, the evolution process should be able to compensate for the fault in the robot hardware.

This experiment has no predefined best or correct solution, as is the case in the multiplexer experiment. The hope is that the evolution process, by continuously evaluating the robot's behaviour, will be able to compensate for errors or changes in the robot hardware such that the controller will continue to function correctly.

The second project, which focuses on processing of audio signals has a simpler test environment than that in the robot experiment. Assuming that the evolution process within CHE is fast enough then it is intended to evolve real-time solutions for signal processing of audio applications. This places a requirement on CHE that fitness evaluation is faster than the data rate of audio signals.

9. Conclusion

There were two main goals for this work. The first, was to investigate ways to implement the evolution process in hardware and to create a prototype solution in FPGA technology. The second, was to evolve a hardware design on the same FPGA chip. These goals can be said to be achieved. Further work will involve exploring the design space of evolvable designs using and improving the GA Pipeline. This will also involve experimentation towards using CHE to implement adaptive hardware.

References

[1] H. de Garis. Artifical life: Growing an artificial brain with a million neural net modules inside a trillion cell cellular automata machine. In *4th International Symposium on Micro Machine & Human Science*, 1993.

[2] R.S. Zebulum, M.A. Pacheco, and M. Vellasco. Evolvable system in hardware design: Taxonomy, survey and applications. In *1st International Conference on Evolvable Systems (ICES96)*, Lecture Notes in Computer Science, pages 344–358. Springer, 1996.

[3] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*, 1997.

[4] J.H. Holland. *Adaption in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[5] T. Higuchi, H. Iba, and B. Manderick. *Massively Parallel Artificial Intelligence*, chapter 12. Evolvable Hardware, pages 398–421. MIT Press, 1994.

[6] H. Hemmi, J. Mizoguch, and K. Shimohara. Development and evolution of hardware behaviours. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware : the Evolutionary Engineering Approach*, volume 1062 of *Lecture Notes in Computer Science*, pages 250–265. Springer, 1996.

[7] J.R. Koza et al. Automated synthesis of analog electrical circuits by means of genetic programming. *IEEE Transactions on Evolutionary Computation*, 1(2):109–128, July 1997.

[8] A. Thompson. An evolved circuit, intrinsic in silicon, entwined with physics. In *1st International Conference on Evolvable Systems (ICES96)*, Lecture Notes in Computer Science, pages 390–405. Springer, 1996.

[9] A. Thompson. Evolving electronic robot controllers that exploit hardware resources. In *The 3rd European Conference on Artificial life (ECAL95)*, page ???, 1995.

[10] T. Higuchi et al. Evolvable hardware and it's application to pattern recognition and fault-tolerant systems. In E. Sanchez and M. Tomassini, editors, *Towards Evolvable Hardware : the Evolutionary Engineering Approach*, volume 1062 of *Lecture Notes in Computer Science*, pages 118–135. Springer, 1996.

[11] *Xilinx Databook: "XC4000E and XC 4000X Series Field Programmable Gate Arrays Product Specification*, 1.4 edition, 1997.

[12] *Xilinx Virtex 2.5V Field Programmable Gate Arrays Product Specification*, 1.1-advance edition, 1998.