Evolving a Robot Controller in Hardware

Pauline C. Haddow and Gunnar Tufte

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

Abstract. In this paper, the concept of hardware evolution is presented along with a newer approach to hardware evolution termed Complete Hardware Evolution (CHE). An experimental robot controller design — GERC (Genetically Evolved Robot Controller), is described which uses CHE in the design process. The robot controller steers the robot around a given area trying to move as straight and as far as possible, exhibiting wall avoidance behaviour. By using CHE the whole evolution process is implemented on the same chip as the evolved circuit — the robot steering circuitry. No external interaction is thus required to evaluate the robots behaviour during the evolution process.

1 Introduction

Two main methods have been established for applying artificial evolution to the design of hardware systems. These are Extrinsic and Intrinsic evolution [dG93,ZPV96]. 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 termed 'Complete Hardware Evolution' (CHE) is presented. 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. An example of CHE in practice is given through the evolution of a robot controller.

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? In the search for solutions for a given task or problem, the problem has to be specified. The solution to the problem is often dependent on the specification of the problem as this steers the implementation of the solution. This places a requirement of accuracy on the specification. If the problem is very complex or incomplete then the required level of accuracy of the specification can be hard to achieve. To find solutions to the problem of developing large and complex designs new design paradigms are required [Ass97].

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. This technique is termed hardware evolution or equally, evolvable hardware. Artificial evolution is a technique that may be applied to problems where the problem space is too large for exhaustive searching. Evolvable hardware may be considered to be a subset of artificial evolution, where the evolved solution is represented in hardware instead of software.

For a given problem where the choice of a hardware or software implementation is available, such as is the case for a robot controller, a cost verses performance trade-off exists. For complex problems where the number of parameters effecting the model are very high or where a large pool of suggested solutions is required so as to achieve an optimal or near optimal solution, the software solution will be limited by the speed of the simulation. Using the same evolutionary techniques but implementing the problem in hardware instead of software allows the raw speed of the underlying technology to be exploited.

At present, most mobile robot controllers are preprogrammed [Ebn98]. However, since a robot controller may be said to be a complex design in that it has to control both asynchronous and synchronous events from it's external components this is no simple task. Evolutionary techniques may be applied to a robot controller to simplify the design process by allowing the evolution process to evolve a design instead of preprogramming the design. That is that evolution may be used to automate the design task.

3 Artificial Evolution

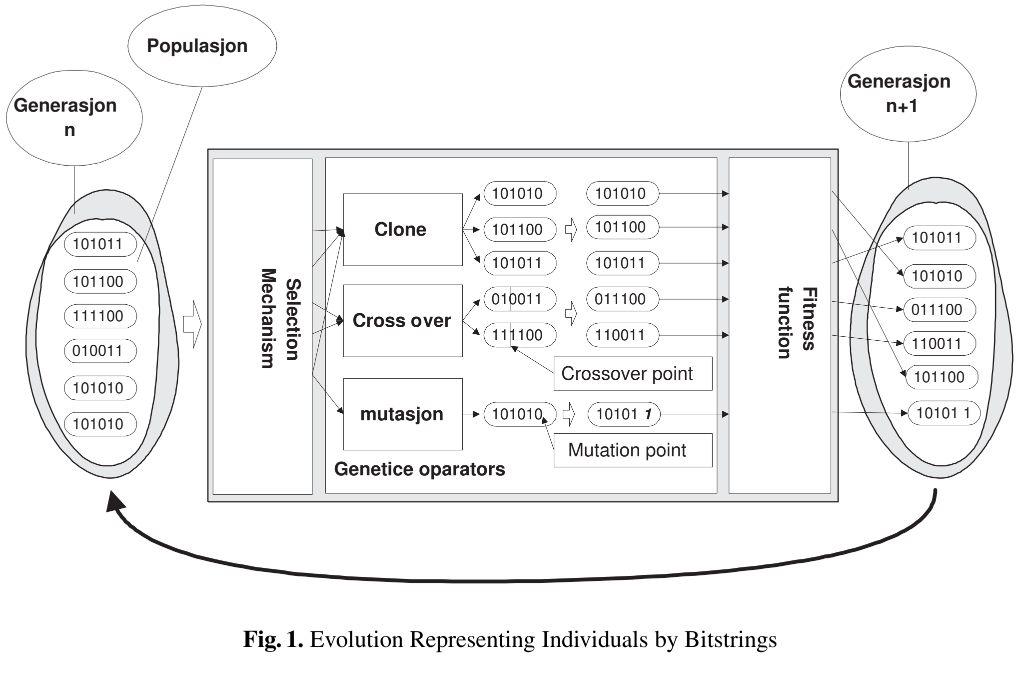

Fig. 1. Evolution Representing Individuals by Bitstrings

The principles of artificial evolution are based on Darwin's theories of evolution by natural selection. These theories were first adapted to the field of artificial evolution in 1975 [Hol75]. The main features of reproduction by cloning, mutation and crossover through the exchange of and alteration of genetic material are still present. However, the selection process is artificial instead of natural and selection is based on how well the individuals solve a predefined task.

Evaluation of individuals is based on a fitness function, which generates a fitness value for the given individual. The fitness value is used by the selection mechanism to select individuals after a given algorithm. The selection mechanism thus puts a selection pressure on the evolution process, the most fit individuals having a greater probability of selection. The evolution process continues from generation to generation, each new generation containing better and better individuals — more 'fit', until an optimal or near optimal solution is found or the evolution process is stopped.

Figure 1 provides an illustration of the evolution process where individuals are represented as bit strings. The evolution process consists of a generation n which holds a fixed sized population of individuals. Each individual represents a possible solution to the problem solution or design to be evolved. At startup, a population is randomly generated.

Individuals are selected for reproduction from generation n by the selection mechanism. Selected individuals are reproduced by cloning, crossover or mutation. If individuals are selected for cloning then they are just copied to the next generation, generation n + 1. If they are selected for crossover, the individuals exchange genetic material around a crossover point (or points) and produce two new individuals for generation n+1. If mutation is chosen, the genetic material is altered rather then swapped as illustrated in figure 1. These mechanisms for reproduction are termed genetic operations.

Selected individuals are then tested by the fitness function and assigned a fitness value. The fitness value is a measurement of how well the individual performs the predefined task. After fitness evaluation the individuals are placed in generation n+1. The selection mechanism selects individuals, based on their fitness value, until the population in generation n+1 is full i.e. a new generation has been evolved. Although selection is based on fitness values, different selection algorithms exist. One such selection algorithm is based on a roulette wheel which is illustrated in figure 2. Distribution of individuals on the roulette wheel is in accordance with the fitness value of the individuals. Here, individual number 1 has twenty percent of the roulette wheel whereas individual number 5 has only three percent of the wheel. As such, individual 1 is more likely to be selected than individual number 5. The selection mechanism spins the wheel and selects individuals. The wheel has to be spun a sufficient number of times such that enough individuals are generated to retain the size of the population in the new generation.

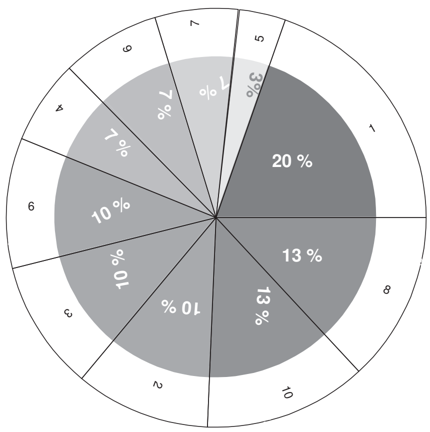

Fig. 2. Roulette Wheel for Selection Mechanism

4 Evolvable Hardware (EHW)

The application of evolutionary techniques to hardware design is termed evolvable (evolutionary) hardware. The main goal being to replace traditional design methods with evolutionary techniques for given applications which are either not achievable using traditional methods or which benefit from an evolutionary approach.

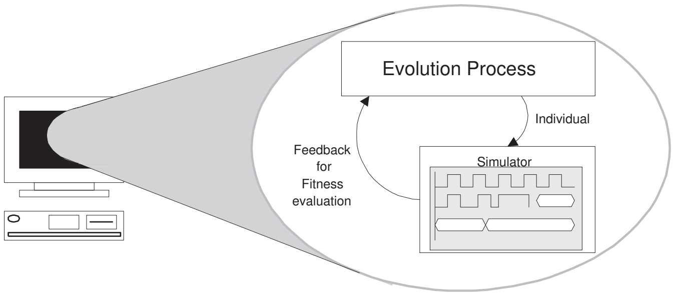

Fig. 3. Extrinsic Evolution

Figure 3 illustrates extrinsic evolution where the entire evolution process including fitness evaluation of the individuals is implemented in software. Extrinsic evolution uses a software simulation of the underlying hardware to evaluate the fitness value of each individual. This may be an advantage if you do not wish to be too technology specific such that the model of the hardware may be quite abstract. On the other hand if technology is the goal then more accurate fitness values may be obtained from an actual implementation than those obtained through an abstraction of the underlying technology i.e. a simulation. Since fitness steers the selection process and thus evolution, then abstracting from technology can lead to a less optimal solution.

Evolutionary techniques may be applied not only to gate-level designs i.e. technology specific designs, but register-transfer level designs. In the work of Hemme et al. extrinsic evolution is used to evolve HDL code [HMS96] i.e. to generate a functional description of a design. Another example of extrinsic evolution, but where a more technology dependent simulator is implemented, is Higuchi's multiplexer design [HIM94]. The work of Koza evolving analogue circuits [Koza97] presents an even more detailed technology model. A Spice net-list with physical models of the components was used to evaluate the simulation design. However, problems incurred in this work highlight the problems inherent in designing a sufficiently detailed accurate model within a reasonable time limit.

In Intrinsic evolution fitness evaluation, is based on a hardware implementation where each individual is implemented and evaluated on the target technology. This approach can be used to explore properties of the technology which can not be utilised using traditional design methods. In Thomson's work [Tho96], for example, intrinsic evolution was chosen so as to take advantage of all the physical parameters that the target hardware had to offer.

Figure 4 illustrates intrinsic evolution of a logical function on a Field Programmable Gate Array (FPGA). The evolution process runs on a host computer responsible for selection and the performance of genetic operators. Each individual is down-loaded to the chip as configuration data — design descriptions. Fitness evaluation of a given individual is achieved by applying test vectors to the implemented individual and then calculating the fitness value from the response.

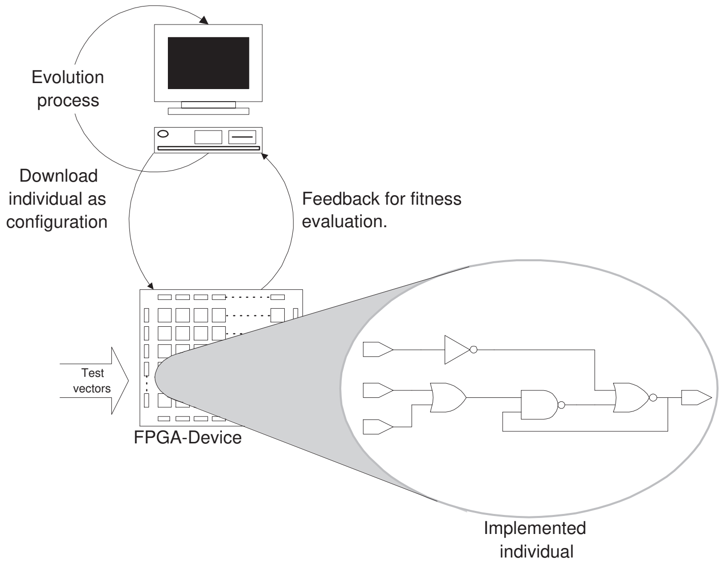

Fig. 4. Intrinsic Evolution

Both Intrinsic evolution and CHE approaches include 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 technology is the number of possible re-configurations. A near unlimited number of re-configurations 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. FPGA technology offers these features and is, therefore, chosen as the implementation technology for the prototype CHE based on a genetic algorithm presented herein.

5 Complete Hardware Evolution (CHE)

Figure 5 illustrates CHE where the genetic algorithm (GA) is implemented on the same chip as the evolving design. The GA implementation configures the evolving design by placing individuals in Random Access Memory (RAM). The fitness value is calculated by the GA from the feedback signals originating in the evolving design. Since the GA and the evolving design are implemented on the same chip, the evolution process may continuously observe the evolving design.

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.

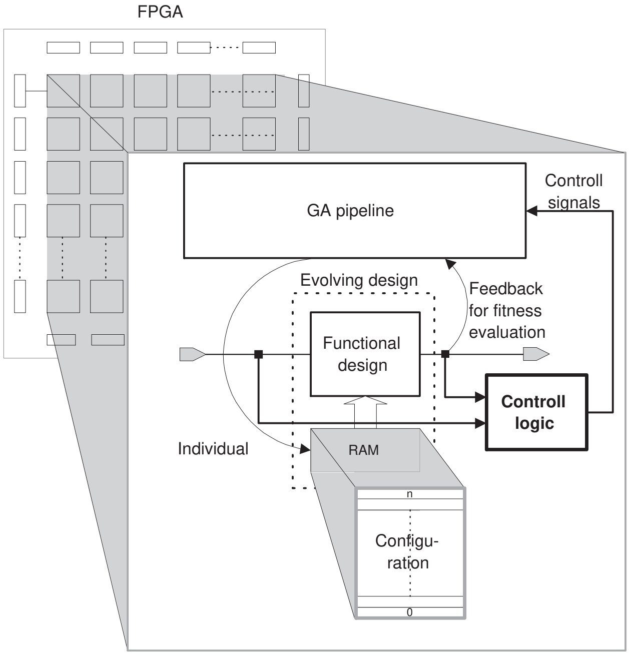

Fig. 5. Complete Hardware Evolution (CHE)

6 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 design itself — initially as a randomly generated design and finally the evolved desired design. External signals are connected to the evolving design to drive fitness evaluation of the current design. These results are then fed into the GA Pipeline for further processing.

Figure 5 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 design.

The main functions of the GA pipeline are selection, genetic operation, fitness and sorting. Each of these functions are implemented in a separate module 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 6 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. It should be noted that this

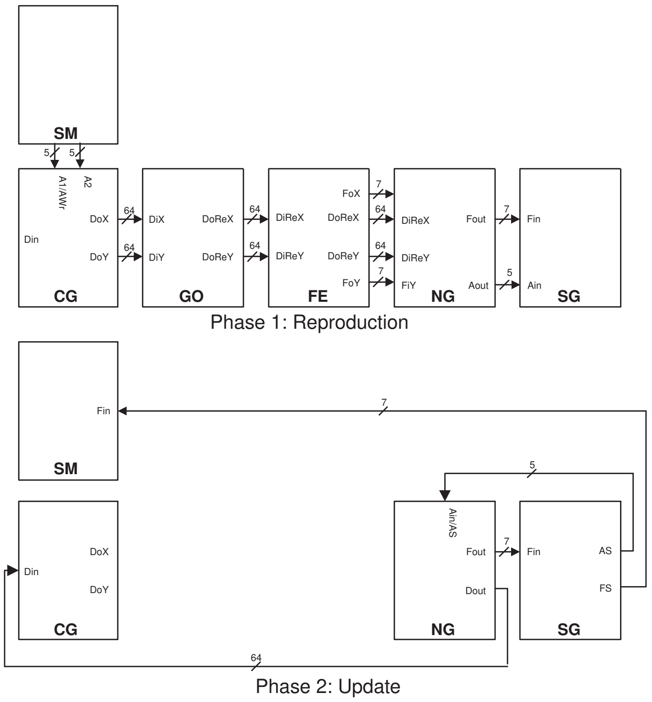

Fig. 6. Genetic Algorithm Pipeline (GA Pipeline)

problem is addressed using what is termed steady-state selection but this selection method is not the focus of this work. In addition to these two phases there is an initial phase which evaluates and sorts all the individuals in the initial random generated population. Further information regarding the individual modules may be found in [Tuf99].

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. 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 respective parameter in the genetic operator module and re-synthesis of the module design. Other changes may require more fundamental adaption of the code such as changing the sorting algorithm in SG.

7 Evolving a Robot Controller

The main component of a robot controller is a state machine. Using traditional hardware design or programming techniques, all transitions and state actions are decided at design time. The state machine is designed to control the external devices of a given robot and to provide the robot with the chosen functionality. Different robots with different external devices will thus require different controllers even if their functionality is the same.

A further problem facing the robot controller designer is the fact that different technologies are controlled from a central controller. That is, the controller has to interface

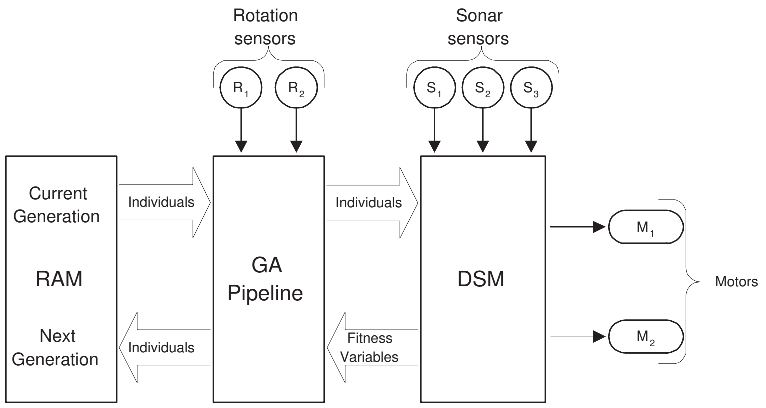

Fig. 7. Data Flow in GERC

both analogue and digital components requiring both asynchronous and synchronous control in the same design. To attack this problem, Thomson [Tho95], introduced the concept of a dynamic state machine (DSM) which provides support for both asynchronous and synchronous transitions.

7.1 GERC overview

The robot presented in this work — GERC a Genetically Evolved Robot Controller, is based on the work of Thomson, using a similar Dynamic State Machine where the outputs and the next state of the state machine are evolved during the evolution process. However, where Thomson's robot was designed to move within a given specified symmetric area, our robot is designed to adapt to the area it is placed in. The goal for the robot is to go as far and as straight as possible and to avoid walls in it's path. The other main difference in our design is the evolutionary hardware approach used. In Thomson's work, intrinsic evolution was used whereas we have moved the complete evolution process into hardware — CHE.

7.2 System Design

The GERC system can roughly be divided into three parts: RAM for storing individuals of the population, the Genetic Pipeline for steering and evaluating evolution of the DSM and the DSM itself, as shown in figure 7. The Genetic Pipeline does all the genetic manipulations and calculates the fitness values. It selects individuals from the current generation (in RAM) based on their fitness values, applies genetic operations and forwards them one at a time to the DSM which controls the robot based on the information in the given individual. The DSM then provides the GA pipeline with variables for fitness calculation of the current individual and the created individuals are added to the new generation also stored in RAM. It should be noted that to preserve a single complete generation for selection purposes, two generations are stored in RAM — the current generation and the next generation under creation.

7.3 Robot Construction

The robot was built using Lego mindstorm for the robot body and motors. Lego parts were chosen as these are easily available, inexpensive and the robot once built is easy to modify. The robot is a two wheel construction, each wheel with it's associated motor. The robot controller controls each motor which can be in one of two states — running or stopped. Since the wheels are placed on opposite sides of the robot, and adjusting the stopped or running state of each of the motors it is possible to steer the robot in any direction.

Sensors attached to the wheels measure the rotation of the wheels. In the current prototype two light detectors are attached to the front of the robot to measure the distance from the robot to the walls. To achieve distance measurements using light detectors the experimental area is colour graded. The centre of the experimental area is white and is gradually colour graded towards dark grey/black at the walls. The information received from all the sensors is fed into the GA pipeline for fitness calculation.

The controller, GERC is implemented on an FPGA chip implemented on a prototype PCB which can be attached to the robot. This prototype card includes an RS232 interface enabling both external power and logging of the evolution process to be achieved, if required. Logging of the evolution process during prototype testing is off course desirable.

7.4 Fitness Evaluation

When designing a static design where one optimal solution is sought such as designing a multiplexer circuit, fitness evaluation drives the evolution process towards an optimal or near optimal single solution. An evolved static design example using CHE may be found in [TH99]. However, for a dynamic design such as is the case for a robot controller, the design solution varies with each step the robot takes. The current solution is to provide an optimal next state and the required outputs. A different optimal solution for the following state may be required, depending on the position of the robot with respect to it's environment.

The fitness function uses the response from the light detectors and the rotation sensors to calculate a fitness value for the implemented individual. The evolution process is continued until a satisfactory next step solution is found.

It should be noted that if the optimal solutions for each step in a given test area are stored then these results may be used to steer a robot with similar external devices and within the same area without the need for evolution.

8 Ongoing and Future Work

The work on the robot is continuing and we expect to obtain performance results from the robot during the early autumn. We are also looking into ways to insert faults into the robot i.e. to simulate a fault in a motor, so as to force the evolution process to evolve a new solution to avoid this fault. That is we are trying to introduce fault tolerance into the robot design.

Sonar sensors will eventually replace the light detectors when the necessary transceiver and receiver circuitry are completed.

Future work in this area will focus on the use of CHE in applications where adaption to changes is important not just due to faults but where adaptive hardware is desirable. In addition, we are looking into the possibility of applying evolutionary techniques, using the capacity now available in reconfigurable technology to investigate ways of modelling in hardware. That is to use evolvable hardware as a problem solver for complex problems arising in other areas of science such as, for example, within medicine.

9 Acknowledgements

We would like to thank Morten Hartmann and Espen Tislevoll for their work in the early stages of the robot design and Adrian Thomson of the University of Sussex for his contributions to their project.

References

[Ass97] Semiconductor Industry Association. *The National Technology Roadmap for Semiconductors*. 1997.

[dG93] 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.

[Ebn98] M. Ebner. Evolution of a Control Architecture for a Mobile Robot. In *2st International Conference on Evolvable Systems (ICES98)*, Lecture Notes in Computer Science, pp 303–310. Springer, 1998.

[HIM94] T. Higuchi, H. Iba, and B. Manderick. *Massively Parallel Artificial Intelligence*, chapter 12. Evolvable Hardware, pp 398–421. MIT Press, 1994.

[HMS96] 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*, pp 250–265. Springer, 1996.

[Hol75] J.H. Holland. *Adaption in Natural and Artificial Systems*. The University of Michigan Press, 1975.

[TH99] G. Tufte and P. C. Haddow. Prototyping a GA Pipeline for Complete Hardware Evolution. In *The 1st NASA/DoD Workshop on Evolvable Hardware*, pp 18–25, 1999.

[Tho95] A. Thompson. Evolving Electronic Robot Controllers that Exploit Hardware Resources. In *The 3rd European Conference on Artificial life (ECAL95)*, p ???, 1995.

[Tho96] A. Thompson. An Evolved Circuit, Intrinsic in Silicon, Entwined with Physics. In *1st International Conference on Evolvable Systems (ICES96)*, Lecture Notes in Computer Science, pp 390–405. Springer, 1996.

[Tuf99] G Tufte. *Prototyping of Complete Hardware Evolution (CHE), Masters Thesis*. The University of Science and technology, Norway, 1999.

[ZPV96] 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, pp 344–358. Springer, 1996.