An Evolving and Developing Cellular Electronic Circuit

Daniel Roggen, Yann Thoma* and Eduardo Sanchez*
Autonomous Systems Laboratory, Institute of Systems Engineering
*Logic Systems Laboratory, Institute of Computing and Multimedia Systems
EPFL, Lausanne, Switzerland
http://asl.epfl.ch, *http://lslwww.epfl.ch
E-mail: name.surname@epfl.ch

Abstract

A novel multi-cellular electronic circuit capable of evolution and development is described here. The circuit is composed of identical cells whose shape and location in the system is arbitrary. Cells all contain the complete genetic description of the final system, as in living organisms. Through a mechanism of development, cells connect to each other using a fully distributed hardware routing mechanism and differentiate by expressing a corresponding part of the genetic code thereby taking a specific functionality and connectivity in the system. The configuration of the system is found by using artificial evolution and intrinsic evolution at the schematic level is possible. Applications include the approximation of boolean functions and the evolution of a controller capable of navigating a Khepera robot while avoiding obstacles. The circuit is suited for a custom chip called POEtic, which is a generic platform to implement bio-inspired applications.

Introduction

New approaches to the creation of electronic circuits have been explored in the last 10 years. Evolvable Hardware (EHW) (Higuchi et al., 1993) consists of using artificial evolution (e.g. genetic algorithms) to create electronic circuits. This approach showed that more efficient circuit implementations than those obtained with traditional techniques may be found (Coello et al., 2000; Vassilev et al., 2000). Evolved circuits may also operate using different principles than those which are obtained by design. They may exploit physical characteristics of devices to implement rich dynamical behaviour from simple building blocks, so better use of the hardware resources may be achieved (Thompson, 1997). EHW is believed to have a lot of potential (Yao and Higuchi, 1999), for instance in adaptive hardware (Kajitani et al., 1998), or in fault-tolerant hardware (Keymeulen et al., 2000), and it becomes more of an industrial reality (Higuchi et al., 2000).

Another approach is to mimick the way living organisms develop from a single cell to form a complete multicellular organism. In Embryonics (Marchal et al., 1994; Mange et al., 1996), cellular decomposition and development are key features which are used to provide a self-repairing substrate for the implementation of electronic circuits, as illustrated by the self-repairing BioWatch (Stauffer et al., 2001). As embryonic cells contain the complete description (genetic code) of the circuit, mechanisms such as self-reproduction and self-repair become possible. Self-repair is an alteration of the development process in which faulty cells are avoided. Faults may be detected by duplication of computational unit (Mange and Tomassini, 1998, 251–258). However the cellular paradigm allows new ways to perform fault-detection. Immunotronics takes inspiration from the way the immune system is capable of discriminating between the normal and anormal behavior of a living cell and transposes such concept to electronic circuits (Bradley et al., 2000). Cellular system also open new possibilities in genotype to phenotype mappings which can improve evolvability of circuits.

Motivated by the previous points, we are interested in the combination of evolution and development in hardware. Such a circuit may combine efficient implementations thanks to evolution with the advantages of the cellular and developmental paradigm to explore bio-inspired hardware. Novel in this paper is the development mechanism combined with evolution, and their implementation on a prototype of the custom chip *POEtic* (Tyrrell et al., 2003). POEtic is a generic platform to implement bio-inspired applications comprising mechansisms such as evolution (Phylogenesis), development (Ontogenesis) and learning (Epigenesis). Retaining the POEtic terminology we call the circuit described here a *PO circuit*. The PO circuit consists of cells which all contain the complete genetic description of the final circuit. Cells can be of any size/shape and can be located anywhere in the circuit. The development process uses a fully distributed dynamic hardware routing mechanism which is available in POEtic. Cells connect to each other at run-time and get to know which part of the genetic code they must express and take a specific connectivity and functionality in the system. The functionality of the system is found by evolving the circuit genotype using genetic algorithms. Applications include the approximation of boolean functions (adder and multiplexer) and the evolution of a controller capable of navigating a Khepera robot while avoiding obstacles. Self-

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023

repair and self-replication issues are discussed. The next section describes the POEtic hardware on which the PO circuit is implemented. Afterwards the cell structure and the development mechanism are discussed, followed by a section showing how the circuit can be evolved. Finally the results are discussed before concluding.

POEtic hardware

The multi-cellular PO circuit is implemented on a novel substrate: the POEtic chip (Tyrrell et al., 2003). The POEtic chip is a platform to test bio-inspired mechanisms, such as mechanisms of evolution (Phylogenesis), development (Ontogenesis) and learning (Epigenesis). Its architecture is summarized below and described extensively in (Thoma et al., 2003).

The POEtic chip is composed of a CPU and an *organic subsystem*. The CPU is a 32-bit RISC with bit manipulation and random number generation capabilities which make it suited to run evolutionary algorithms.

The organic subsystem is where multi-cellular PO circuits are implemented. It is composed of two layers: a layer of *molecules* and a *routing layer* (see fig. 1, right).

Molecules contain a 16 bits lookup table (LUT), a flip-flop and a switchbox for local routing. The molecule output can be registered or combinational. Molecules can operate in different modes. In the *3/4-LUT* mode, the molecule computes a logic function of 3- or 4-inputs. Modes *input* and *output* allow molecules to interact with the routing layer. Mode *reconfigure* is used to reconfigure another molecule. This is extensively used for the development mechanism of the PO circuit.

The routing layer establishes long distance connections between molecules (e.g. for inter-cell communication) and interfaces molecules to physical pins of the circuit. It is capable of *dynamic routing* to create connections between molecules automatically and at *run-time*. The routing layer is composed of routing units (RU) which are identified by a 16-bit ID and tagged as either *sources* or *targets*. When dynamic routing is triggered, logic within the RU connects sources and targets which have the same ID using a breadth-first search algorithm (Thoma et al., 2003; Moreno Aróstegui et al., 2001). Molecules can change the identifiers in the RU and retrigger the routing mechanism. Therefore new connections can be built at runtime. This may be used to respond to environmental changes, for example changes in the location of sensors or actuators in a robot. Note that this feature does not exist in FPGAs where physical connections are mapped at *design-time* to the FPGA.

Because the final POEtic chip will be available by the summer, prototyping is done by implementing the key parts of POEtic on an FPGA. The main difference is that here the organic subsystem is serially configured whereas in the final POEtic chip it is mapped in the address space of the CPU and accessed in parallel. The architecture of the system is

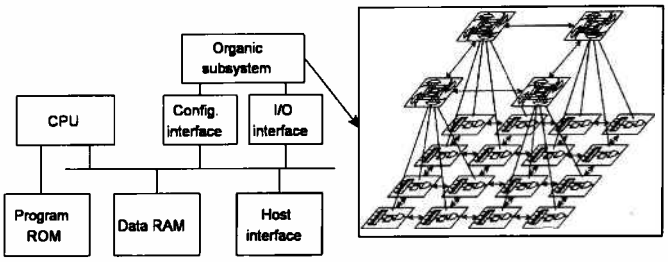


Figure 1: Left: the architecture of the system implemented on the FPGA. Right: the organic subsystem is composed of an array of molecules and an array of routing units.

shown in fig. 1, left. A 4K-word program ROM and an 8K-word RAM is interfaced to the CPU. The configuration interface takes care of the serial configuration and the I/O interface is used to read and write to the I/O of the organic subsystem, to set control bits and to read its status. The host interface allows the CPU to send data to the host (the PC hosting the FPGA board) for example to monitor the status of evolution.

Synthesis has been done for a Xilinx Virtex XC2V3000-4 FF1152. Ressource usage are around 5% of the LUTs for the CPU and 66% for an organic subsystem composed of 80 molecules, while some other logic serves to connect both together. Placement and routing of the system ended up with a 97% ressource usage, and 51% of RAM blocks usage. Placement could succeed with 100 molecules but routing could not. After place and route no theoretical maximum speed could be calculated, as there are combinational paths because of the molecules switch boxes. However, successful execution could be achieved at 10MHz.

One could argue that 80 molecules is a very small number compared to a Xilinx that owns more than 14000 slices, but we have to keep in mind that an FPGA is not designed to emulate another FPGA. For instance, as 76 configuration bits define a molecule, at least 76 flip-flops are needed, only for this purpose. Therefore, a hardware realization of POEtic is the only way to obtain a efficient system.

Circuit development

Cell architecture

The basic unit of the PO circuit is the cell. Key characteristics are that all cells are identical and contain the complete genetic description of the whole organism. A development mechanism is used to build the multi-cellular circuit from initially unconnected and undifferentiated cells. Following the view introduced in (Tyrrell et al., 2003) the cell can be viewed as the three layered structure illustrated in fig. 3. The genotype is a memory storing the genetic code of the entire circuit. The mapping layer is where development mechanisms take place. The phenotype layer is the functional part of the cell. In this implementation it has three inputs and

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023

one output and can compute any logic function of the inputs with a lookup table.

A cell composed of 40 molecules has been designed (fig. 2) which allows to fit 2 cells in the system. The molecules are composed of three different blocks, corresponding to the three layers depicted in fig. 3:

- The genotype layer is implemented with 16 molecules which store the genome. Each cell of the organism needs 4 molecules. Therefore the same cell can be used for organisms of up to four cells.
- The mapping from genotype to phenotype is done by 18 molecules: 8 molecules are responsible for the growth process, and 10 molecules used to differentiate cells.
- The phenotype layer is implemented by 5 molecules. Three of them serve as inputs, one as the functional part of the cell (a 3-LUT), and one as the output.

Development mechanism

Development maps the functionality of the cell from its genotype. It operates in two phases: growth and differentiation. The mechanism is based on a unique identifier for each cell, encoded in one-hot (a single bit is 1 at any time and indicates the ID) to reduce the cell size (binary encoding is possible but takes more space). For a system of n cells (n being a number in [1,16]), the first cell has an ID of n-1. For instance, if n=4, its ID would be 0...01000 (in this prototype n=2). IDs are 16-bits.

Growth The growth phase lets the organism grow from one cell to the whole organism. It is initiated by the CPU which first configures the entire organic subsystem with the cell description, including the genome. All cells are identical, totipotent, but are unconnected and undifferentiated. Cells have an *input* molecule waiting for a connection, with address 1...111. An external agent (e.g. the CPU) starts a dynamic routing, by configuring an I/O routing unit to be a source, with address 1...111. Dynamic routing creates a path to the closest cell. Once the routing process is over, the external agent sends serially the ID of the cell (number n-1). The cell stores it and computes the ID of the next cell by shifting its own ID by 1 (if cell is 0...01000, then next one is 0...0100). Afterwards, an output of the cell, with address 1...111 launches a new routing process, to connect to the nearest available cell. It then transmits the newly calculated ID, and the process continues until a cell receives the ID 0...01. It recognizes it is the last one, and does not start a new routing process.

The end of the growth phase is detected by the way of a global enable. This special feature of POEtic allows molecules which are sensitive to this signal to act on a global enable. In our system, the molecules involved in the growth process are not sensitive to this enable, but the others are. Once a cell has received its ID, it sets its global enable line, and so, the differentiation phase can not start before the organism is totally built.

Differentiation Cells know their ID and can differentiate to express the corresponding part of the genome. The genome codes the functionality (3-LUT content) and the connectivity (three inputs addresses) of each cell. Hence the topology on the phenotype layer needs not be the same as that of growth which is topologically linear on the mapping layer. The topology of the phenotype is given by the genotype and can be anything. For example, at the phenotype level, cells could be connected to form an array.

Four shift-memory molecules are required to store the genome of a cell. The partial reconfiguration capabilities of POEtic are fully exploited in the differentiation phase, where the 3 input molecules and the 3-LUT (functional) molecule are reconfigured with the corresponding part of the genome.

During the differentiation phase, the genome is shifted, the output being redirected to the input. Based on the ID of the cell, counters are used to enable the partial reconfiguration of the phenotype at the right time, that is when the output of the genome corresponds to the current cell. The reconfiguration lasts 4x16=64 clock cycles for each cell, but as the genome has to shift entirely it makes the differentiation longer.

At the end of the differentiation phase, every cell has its functional part ready, but cells are not yet inter-connected at the phenotype level. Therefore, every cellular input launches a routing process, until all cells are connected on the phenotype layer. Finally, the organism is ready to operate, the microprocessor can apply the inputs, retrieve the outputs, and calculate the fitness of this newly created individual.

Circuit evolution

In the PO concept, circuits are *evolved* rather than *designed*. The behaviour of the PO circuit is determined by its *configuration*: how cells are interconnected and what functionality cells take. This information is stored in the genotype layer of cells and can be evolved using genetic algorithms (GA) to obtain the desired functionality.

To demonstrate that the PO circuit can be evolved, two applications are considered. The first consists in implementing logic functions (adders and multiplexers) in the PO circuit, and the second consists in evolving a Khepera robot controller to perform navigation with obstacle avoidance using proximity sensors. In the latter case the relation between inputs and outputs which give the desired robot behaviour may potentially be difficult to determine which makes evolution well suited for such problems (Harvey et al., 1993).

Evolved "organisms" (PO circuits) are implemented in the organic subsystem of the POEtic hardware. A PO circuit is composed of two three-input cells (see 4, left). It has six inputs and two outputs which are connected by design to the output of the cells. Cells can compute any logic function of three inputs by the mean of a lookup table. The inputs of

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023

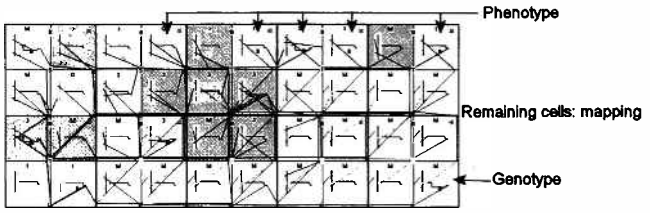


Figure 2: The cell is composed of 10 by 4 molecules. The complete circuit contains two such cells. Within a molecule is drawn its configuration. Five molecules form the phenotype, 16 molecules store the genotype and the remaining cells are used for the growth and differentiation.

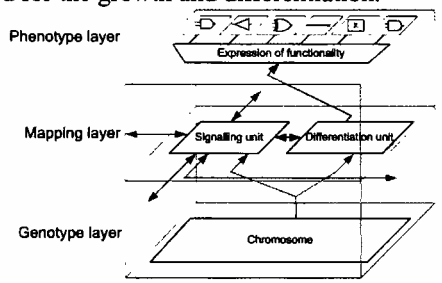


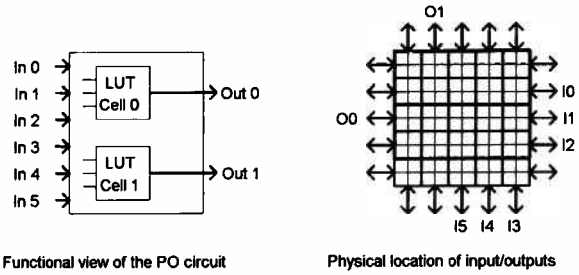
Figure 3: The 3-layer decomposition of a cell. The genotype layer holds the genetic code. The mapping layer handles development. The phenotype layer is the functional part of the cell, selected among a given repertoire of functionalities.

the cells can be connected to one of eight possible locations: one of the six inputs, or the output of one of the two cells. Input and outputs of the PO circuit can be accessed by the CPU which is in charge of running the GA and computing the circuit fitness. The physical location of the inputs and outputs in the organic subsystem are shown on the right of fig. 4. The outputs are placed close to the output molecule of the cells to minimize the use of RU. The only consideration regarding the placement of the inputs is to avoid the leftmost column of routing units, which are used by the development mechanism.

Evolution is applied to the input connectivity of the cells and to the content of their lookup table. The genetic code is a compact version of what is stored in the genotype layer of the cells to reduce the size of the search space. Eight bits are used to encode the content of the LUT and three bits are used for each cell input to encode the connectivity. Therefore the complete genetic code takes 34 bits (17 bits per cell).

Evolving logic functions

The PO circuit is evolved to implement logic functions. Two functions are considered: a multiplexer and a full adder. Three inputs are used (inputs 0 to 2) while inputs 3, 4 and 5 are set at all time to constant values 0, 1 and 0 respectively. The multiplexer uses one output whereas the adder uses two which represent the sum and the carry out. Circuits



Functional view of the PO circuit

Physical location of input/outputs

Figure 4: Left: functional view of the PO circuit with two cells which can compute logic functions of three inputs. The circuit has two outputs and five inputs. The outputs are connected by design to the outputs of the cells. The connections of the inputs of the cells are evolved together with the logic function. Right: physical location of the input and outputs of the circuit.

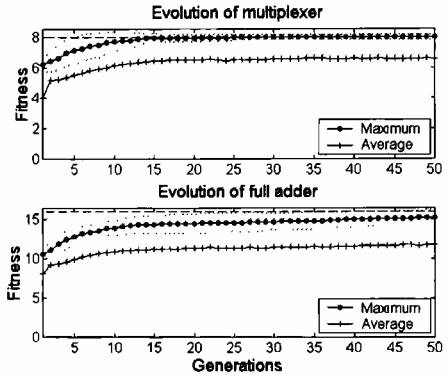


Figure 5: Maximum and average fitness over the generations when evolving logic functions. The horizontal dashed line represents the maximum fitness. The dotted lines either side the maximum represent the standard deviation between 20 runs.

are evolved by a GA with the following parameters: population size of 200, rank selection of the 20 best individuals, 5% of mutation rate, one-point crossover rate of 30% and elitism.

The fitness is evaluated by comparing the output of the circuit with the desired output for all possible inputs. It is equal to the number of times the outputs take the correct value. The maximum fitness is 8 and 16 for the multiplexer respectively the full adder. Fig. 5 shows the fitness over the generations averaged on 32 runs for the multiplexer and full adder. Evolution managed to implement the multiplexer in 31 of the 32 runs (in one run the maximum fitness obtained is 7). The full adder is evolved in 20 of the 32 runs (remaining runs achieve a maximum fitness of 14). As expected the multiplexer is easier to evolved than the full adder because it is a simpler circuit which can be implemented in only one cell whereas the full adder needs two.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023

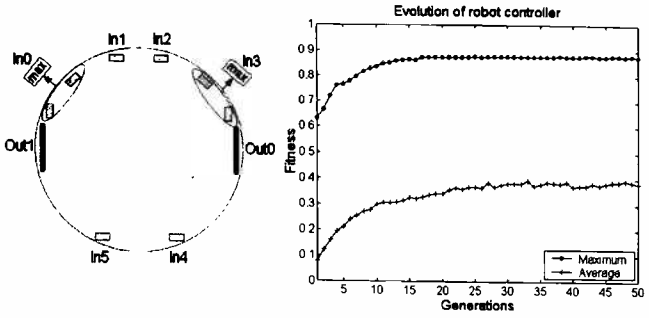


Figure 6: Left: mapping of the Khepera sensors and actuator on the PO circuit. Right: evolution of the maximum and average fitnes of the robot controller. Maximum fitness is 1.

Evolving a robot controller

The PO circuit is evolved to control the navigation of a Khepera robot to avoid obstacles, that is to map sensory inputs to motor commands. The Khepera is a two-wheeled robot with 8 proximity sensors (K-TEAM, 1999). The proximity sensors are connected to the inputs of the PO circuit, and the motors of the Khepera are controlled by the output of the cells. Fig. 6 left shows the mapping of the sensors and motors to the PO circuit. Some sensors are grouped by taking the value of the most active sensor. The robot has a sensory motor period of 100ms during which the speed of the wheels remain constant. At the end of the period, the outputs of the PO circuit update the speed of the wheels. An output of 1 corresponds to a wheel speed of +80mm/s, while an output of 0 corresponds to a speed of -80mm/s. Obviously wheels can never stop. This is a limitation imposed by the small amount of cells which fit on the FPGA. Afterwards, obstacles are sensed and the inputs of of the circuit are set for the next sensory motor period.

The fitness of the controller is determined from the behaviour of the robot: straight motion should be maximized while minimizing contacts with the walls. The fitness function is the sum of the speeds of the wheels at each sensory-motor step when both wheels spin forward (Floreano and Mattiussi, 2001). It favors obstacle avoidance because the wheels of a robot which is stuck against an obstacle do not spin due to the friction with the ground.

The circuit has been evolved using the same GA parameters as previously. Fig. 6 right shows the evolution of the fitness over the generations. A good obstacle avoidance behaviour is already obtained after about 10 generations. Note that evolution was performed in simulation to speed up experiments and the best individual tested successfully on a real robot.

Discussion

The development mechanism is novel in many respects. An important difference compared to embryonics development is that cells can be of any shape and can be physically placed anywhere in the organic subsystem even at irregular intervals. This may be interesting for example if parts of the organic subsystem are damaged: cells may be placed on the functional molecules (an off-line test may reveal the damaged locations), and dynamic routing takes care connecting cells in a transparent way.

Compared to classic unconstrained evolution which consists of manipulating directly the configuration bits of an FPGA (Thompson, 1997), evolution could be performed at a higher level thanks to dynamic routing. Indeed the approach could be classified as *intrinsic schematic* evolution. Connections are evolved by encoding identifiers of source cells (like in a net-list), rather than by encoding the configurations of many switchboxes. Consequently the genetic coding is more compact and evolution may be faster. Note that the genetic coding resembles Cartesian Genetic Programming (Miller, 1999) which also encodes the functionality and connectivity of every cell. In particular circuits which were evolved in simulation in the latter paper can be intrinsically evolved in the PO circuit.

Several features unique to the POEtic chip have been used to realize the PO circuit. The close interaction between the CPU and the organic subsystem allow fast reconfiguration of the organic subsystem by the CPU when running evolutionary algorithms. The POEtic CPU containes features such as a hardware random number generator which may speedup the execution of evolutionary algorithms. Dynamic routing and the possibility of one molecule to reconfigure another molecule to achieve self-reconfiguration are also at the core of the development mechanism.

The POEtic chip implemented on the FPGA has been changed in a number of ways compared to the final POEtic chip to reduce the space taken. Notably the final POEtic chip will contain an AMBA bus to interface with internal and external peripherals. Additional peripherals such as timers and hardware multipliers will be available. Also several chips may be cascaded to form a larger organic subsystem.

Conclusions

An evolving and developing circuit has been implemented on a novel POEtic chip. The circuit is composed of identical cells, all of them containing the complete genetic description of the final system, as in living organisms. A hardware development mechanism using specific features of the POEtic chip is used to build the circuit starting from unconnected and undifferentiated cells through a growth and differentiation process. Evolution has been used successfully to find suitable configurations of the circuit in tasks such as the evolution of logic functions and the evolution of a robot controller.

The circuit described here can be improved in a number of ways. Self-repair although mentionned has not yet been explored. Self-repair requires a means to detect faults in the

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023

circuit. This can be done by functional redundancy within the cells or by following the immunotronics approach. Upon detection of a defective cell, it would go offline and a new development process could be triggered which would make use of spare cells placed in the circuit.

Cells contain the genetic code of the complete circuit, however they were not designed to transfer the genetic code to other cells during or after development. As such, circuit self-replication is not yet possible. Modifications to allow self-replication may be explored in the future.

A direct genotype to phenotype mapping has been used and this is known to lead to scalability issues when evolving complex circuits. Indirect genotype to phenotype mappings can be implemented in the mapping layer of the cells. In particular a *morphogenetic coding* has been developed taking inspiration from the way inter-cellular chemical signalling regulate the functionality of cells. It has been designed to remain simple and suited for hardware implementations (Roggen et al., 2003). Further work may explore the combination of evolution and development using such indirect genetic codings.

Acknowledgments

This project is funded by the Future and Emerging Technologies programme (IST-FET) of the European Community, under grant IST-2000-28027 (POETIC). The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication. The Swiss participants to this project are funded by the Swiss government grant 00.0529-1.

References

Bradley, D., Ortega-Sanchez, C., and Tyrrell, A. (2000). Embryonics + immunotronics: A bio-inspired approach to fault tolerance. In Lohn, J. et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 215–223, Los Alamitos, California. IEEE Computer Society Press.

Coello, C. A., Aguirre, A. H., and Buckles, B. P. (2000). Evolutionary multiobjective design of combinational logic circuits. In Lohn, J. et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 161–170, Los Alamitos, California. IEEE Computer Society Press.

Floreano, D. and Mattiussi, C. (2001). Evolution of spiking neural controllers for autonomous vision-based robots. In Gomi, T., editor, *Evolutionary Robotics IV*, pages 38–61. Springer-Verlag, Berlin.

Harvey, I., Husbands, P., and Cliff, D. (1993). Issues in evolutionary robotics. In Meyer, J.-A. et al., editors, *Proc. of the 2nd Int. Conf. on Simulation of Adaptive Behaviour*, pages 364–373. MIT Press/Bradford Books.

Higuchi, T. et al. (1993). Evolving Hardware with Genetic Learning: A First Step Towards Building a Darwin Machine. In Meyer, J.-A. et al., editors, *Proc. of the 2nd Int. Conf. on Simulation of Adaptive Behaviour*, pages 417–424, Cambridge, MA. MIT Press-Bradford Books.

Higuchi, T., Iwata, M., Keymeulen, D., et al. (2000). Real-world applications of analog and digital evolvable hardware. *IEEE Transactions on Evolutionary Computation*, 3(3):220–235.

K-TEAM (1999). *Khepera User Manual.* K-TEAM S.A., Préverenges, Switzerland (http://www.k-team.com).

Kajitani, I. et al. (1998). A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI. In Sipper, M. et al., editors, *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, pages 1–12, Berlin. Springer.

Keymeulen, D., Zebulum, R., Jin, Y., and Stoica, A. (2000). Fault-tolerant evolvable hardware using field-programmable transistor arrays. *IEEE Transactions on Reliability*, 49(3):305–316.

Mange, D., Goeke, M., Madon, D., Stauffer, A., Tempesti, G., and Durand, S. (1996). Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties. In Sanchez, E. and Tomassini, M., editors, *Towards Evolvable Hardware*, pages 197–220, Berlin. Springer.

Mange, D. and Tomassini, M. E. (1998). *Bio-Inspired Computing Machines*. PPUR, Lausanne.

Marchal, P., Piguet, C., Mange, D., Stauffer, A., and Durand, S. (1994). Embryological development on silicon. In Brooks, R. and Maes, P., editors, *Proc. of the 4th Int. Conf. on the Synthesis and Simulation of Living Systems*, pages 365–370. MIT Press.

Miller, J. F. (1999). On the filtering properties of evolved gate arrays. In Stoica, A. et al., editors, *1st NASA/DoD Workshop on Evolvable Hardware*, pages 2–11, Los Alamitos, California. IEEE Computer Society Press.

Moreno Aróstegui, J.-M., Sanchez, E., and Cabestany, J. (2001). An in-system routing strategy for evolvable hardware programmable platforms. In Keymeulen, D. et al., editors, *3rd NASA/DoD Workshop on Evolvable Hardware*, pages 157–166, Los Alamitos, California. IEEE Computer Society Press.

Roggen, D., Floreano, D., and Mattiussi, C. (2003). A Morphogenetic Evolutionary System: Phylogenesis of the POEtic Tissue. In Tyrrell, A. M. et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 153–164, Berlin. Springer.

Stauffer, A., Mange, D., Tempesti, G., and Teuscher, C. (2001). A self-repairing and self-healing electronic watch: The biowatch. In Liu, Y. et al., editors, *Proc. of the 4th Int. Conf. on Evolvable Systems (ICES 2001)*, pages 112–127, Berlin.

Thoma, Y., Sanchez, E., Moreno Arostegui, J.-M., and Tempesti, G. (2003). A dynamic routing alogrithm for a bio-inspired reconfigurable circuit. In *Proc. of the 13th Int. Conf. on Field Programmable Logic and Applications (FPL'03)*, pages 681–690, Berlin. Springer Verlag.

Thompson, A. (1997). An evolved circuit, intrinsic in silicon, entwined with physics. In Higuchi, T. et al., editors, *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, pages 390–405, Berlin. Springer.

Tyrrell, A. M., Sanchez, E., Floreano, D., Tempesti, G., Mange, D., Moreno, J.-M., Rosenberg, J., and Villa, A. (2003). POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware. In Tyrrell, A. M. et al., editors, *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, pages 129–140, Berlin. Springer.

Vassilev, V. K., Job, D., and Miller, J. F. (2000). Towards the automatic design of more efficient digital circuits. In Lohn, J. et al., editors, *2nd NASA/DoD Workshop on Evolvable Hardware*, pages 151–160, Los Alamitos, California. IEEE Computer Society Press.

Yao, X. and Higuchi, T. (1999). Promises and challenges of evolvable hardware. *IEEE Transactions on Systems, Man, and Cybernetics*, 29(1):87–97.

Copyrighted Material

Downloaded from http://direct.mit.edu/books/book/chapter-pdf/268186/9780262291392_caf.pdf by guest on 22 September 2023