MULTI-CELLULAR RECONFIGURABLE CIRCUITS: EVOLUTION, MORPHOGENESIS AND LEARNING

THÈSE N° 3198 (2005)

PRÉSENTÉE À LA FACULTÉ DE SCIENCES DE L'INGÉNIEUR

ECOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Daniel Roggen
Ingénieur Diplomé EPFL en Microtechnique

Composition du jury de thèse:

Prof. Dario Floreano, directeur de thèse
Prof. Xin Yao, rapporteur
Prof. Andy Tyrrell, rapporteur
Prof. Eduardo Sanchez, rapporteur

Lausanne, EPFL
March 3, 2005

Abstract

Bio-inspired electronic circuits have the potential to address some of the shortcomings of conventional electronic circuits, such as lack of applicability to ill-defined problems, of robustness, or of adaptivity to unexpectedly changing environments.

Bio-inspired circuits are designed by taking inspiration from principles observed in biology. The evolution of biological organisms, their development from a fertilized egg, and their learning capabilities are three sources of bio-inspiration that can be used for this purpose.

Until now bio-inspired electronics mostly focused on a single aspect of bio-inspiration: either evolution, development or learning. In this thesis we consider that electronic circuits should encompass all three aspects to fully benefit from bio-inspiration. These circuits capable of evolution, development and learning are called POEtic circuits (POE stands for phylogeny, ontogeny and epigenesis, that mean respectively evolution, development and learning).

Conceptually these POEtic circuits, much like biological organisms, are multi-cellular circuits that evolve following the principles of selection and differential reproduction, they develop from a single cell and differentiate according to inter-cellular and environmental signals, and eventually they learn during their lifetime. These circuits may also dynamically reorganize their structure in order to cope with changes in the environment, or when they are expanded with new cells, sensors or actuators. In comparison to conventional circuits, POEtic circuits are created automatically using evolutionary principles, even if only a partial or high-level specification of the problem is known. Development provides a complex genotype to phenotype mapping, that may lead to fault-tolerance or adaptive development in order to cope with environmental changes. Finally learning allows these circuits to memorize past events or adapt their response over time to improve their behavior.

This thesis deals with the evolutionary mechanisms required to evolve these POEtic circuits. We argue that in order to fully realize the potential of POEtic circuits a novel evolutionary system that takes into account their characteristics and that encompasses both a genetic encoding and a developmental system is required. Indeed, evolutionary algorithms commonly used to evolve electronic circuits do not exploit the complex dynamics of development which is seen in biological organisms. They generally use a direct ge-

netic encoding with a one to one genotype to phenotype mapping. As a consequence the genetic string grows with the size of the circuits and this may limit the scalability of the evolutionary approach to larger circuits. Furthermore these encodings do not allow intercellular or environmental interactions during development, which could lead to adaptive development or fault-tolerant circuits.

In this thesis we develop an evolutionary system suited for multi-cellular POEtic circuits. This evolutionary system is inspired by the mechanisms of gene expression and cellular differentiation seen in biological organisms. It attempts to provide better evolvability and scalability than direct genetic encodings, it allows cellular or environmental interactions during development, and it is computationally simple so that it can be efficiently implemented in hardware. It is furthermore generic, and it makes minimal assumptions on the circuits that are evolved: other than assuming they are multi-cellular, it only requires local communication between neighboring cells.

We demonstrate the proposed evolutionary system by evolving multi-cellular circuits for a wide range of applications. The results that we obtain confirm the generality of our approach and its advantages in comparison to direct genetic encodings.

The proposed evolutionary system is used to evolve structures of differentiated cells, and it shows better scalability to larger structures in terms of fitness than a direct genetic encoding. The dynamics of development within the evolutionary system can recover these structures in case of faults, even at high fault rates. The proposed evolutionary system is used to evolve multi-cellular circuits composed of spiking neurons to recognize patterns and to control the navigation with obstacle avoidance of a mobile robot, and in comparison it outperforms a direct genetic encoding. Finally it is used to evolve circuits capable of learning that control a mobile robot in a vision-based learning and navigation task. This last application demonstrates the three aspects of bio-inspiration of POEtic circuits in a single task: evolution, development and learning.

Version abrégée

Les circuits électroniques bio-inspirés ont le potentiel de remédier à certaines carences des circuits électroniques conventionnels, tels que la difficulté à les utiliser lorsqu'une application est mal définie, leur manque de robustesse, et leur manque d'adaptabilité lorsque l'environnement change de façon imprévue.

Les circuits bio-inspirés sont créés en prenant inspiration de principes observés dans la biologie. L'évolution des organismes biologiques, leur développement d'un oeuf fertilisé, et leur capacité d'apprentissage sont trois sources de bio-inspiration qui peuvent être utilisées pour cela.

Jusqu'à présent l'électronique bio-inspirée s'est grandement focalisée sur un seul aspect de la bio-inspiration: soit l'évolution, le développement ou l'apprentissage. Dans cette thèse nous considérons que les circuits électroniques devraient englober tout les trois aspects pour pleinement bénéficier de la bio-inspiration. Ces circuits capables d'évoluer, de se développer et d'apprendre sont appellés des circuits POEtic (POE signifie phylogenèse, ontogenèse et épigenèse, soit respectivement évolution, développement et apprentissage).

Conceptuellement ces circuits POEtic sont, de même que les organismes biologiques, des circuits multi-cellulaires qui évoluent en suivant le principe de la sélection et de la reproduction différentielle, ils se développent à partir d'une seule cellule et se différentient en fonction des signaux inter-cellulaires et environnementaux, et finalement ils apprennent durant leur vie. Ces circuits peuvent aussi réorganiser dynamiquement leur structure de façon à tolérer des changements dans l'environnement, ou lorsqu'ils sont agrandis avec de nouvelles cellules, senseurs ou actuateurs. En comparaison avec les circuits conventionnels, les circuits POEtic sont créés automatiquement en utilisant les principes évolutionnaires, même lorsque seulement une spécification partielle ou de haut niveau à un problème est connue. Le développement pourvoit une conversion complexe de génotype à phénotype qui peut amener à la tolérance aux pannes ou au développement adaptatif de façon à faire face à des changements environnementaux. Finalement l'apprentissage permet à ces circuits de mémoriser des évènements passés et d'adapter leur réponse au cours du temps de façon à améliorer leur comportement.

Cette thèse traite des mécanismes évolutionnaires requis pour évoluer ces circuits POEtic. Nous soutenons qu'afin de réaliser pleinement le potentiel des circuits POEtic, un

nouveau système évolutionnaire est requis qui prend en compte les caractéristiques de ces circuits et qui englobe à la fois un codage génétique et un système développemental. En effet, les algorithmes évolutionnaires couremment utilisés pour évoluer des circuits électroniques n'exploitent pas la dynamique complexe du développement qui est observée dans les organismes biologiques. Ils emploient généralement un codage génétique direct avec une conversion un à un du genotype vers le phénotype. En conséquence la chaîne génétique croît avec la taille des circuits et cela peut limiter la scalabilité de l'approche évolutionnaire à des circuits plus grands. De plus, ces codages ne permettent pas les interactions inter-cellulaires ou environnementales durant le développement qui pourraient mener au développement adaptatif du circuit ou à de la tolérance aux pannes.

Dans cette thèse nous développons un système évolutionnaire destiné aux circuits multi-cellulaires POEtic. Ce système évolutionnaire est inspiré des mécanismes d'expression génétique et de différentiation cellulaire observés dans les organismes biologiques. Il tente d'amener une meilleure évolvabilité et scalabilité que les codages génétiques directs, il permet les interactions inter-cellulaires et environnementales durant le développement et il est computationnellement simple, ce qui permet une implémentation efficace en hardware. De plus il est générique et fait des assomptions minimales quant aux circuits qui sont évolués: outre l'assomption qu'ils sont multi-cellulaires, il requiert uniquement une communication locale entre cellules voisines.

Nous démontrons le système évolutionnaire proposé en évoluant des circuits multi-cellulaires pour une large gamme d'applications. Les résultats que nous avons obtenus confirment la généralité de notre approche et sa meilleure performance en comparaison à un codage génétique direct.

Le système évolutionnaire proposé est utilisé pour évoluer des structures de cellules différentiées et il montre une meilleure scalabilité lors de l'évolution de structures plus grandes en comparaison à un codage génétique direct. La dynamique du développement au sein du système évolutionnaire permet de récupérer ces structures après qu'elles aient été endommagées, même avec un fort taux d'endommagement. Le système évolutionnaire proposé est utilisé pour évoluer des circuits multi-cellulaires composés de neurones à impulsions pour faire de la reconnaissance de formes et pour contrôler la navigation avec évitement d'obstacles d'un robot mobile. Il atteint une meilleure performance en comparaison à un codage génétique direct. Finalement il est utilisé pour évoluer des circuits capables d'apprentissage qui contrôlent un robot mobile dans une tache d'apprentissage et de navigation basée sur la vision. Cette dernière application démontre les trois aspects de la bio-inspiration des circuits POEtic dans une même tâche: l'évolution, le développement, et l'apprentissage.

Acknowledgements

This thesis did not start nor end by case. Or it might have. Looking back, this thesis is not only the result of hard work: it is also the result of meeting and interacting with many persons, including students, colleagues and friends that I met during these last years. I am grateful to all these persons. Without them this thesis would have been different and these years would have been less enjoyable.

First I am very grateful to my advisor, Prof. Dario Floreano. Dario offered me this Ph.D. opportunity while I was already working. There was thus a difficult period until I decided to take the chance, but my choice was certainly the correct one. Thanks for this opportunity Dario, and thanks for your support throughout my thesis (I especially remember one heated debate during a European project meeting)!

I wish to thank my former colleagues, in particular Julien Reichel, Francesco Ziliani, Sushil Bhattacharjee and Jean-Claude Michelou, that lost me to this Ph.D. I enjoyed working with them, but working with so many doctors had to have some side effects!

I am grateful to Hicham Majd that convinced me over a couscous at a Nomads party to think seriously about doing a Ph.D. That was a worthwhile couscous! He also offered me numerous advice throughout the last few years. Thanks!

Starting a Ph.D. in a new lab was an interesting experience since I knew the lab by three different names and I had the opportunity to see it grow over the years. I am thankful to my colleagues that have made of these last years a very interesting scientific and social experience.

In particular I am grateful to Jesper Blynel, a friend from whom I learned many things about science, the community, and the Viking's perception of the world. In addition we spent very many evenings at Satellite or in the city socializing with lab visitors or newcomers. I might still have to buy him some pizza though.

At some point Mototaka Suzuki was one of those newcomers and I think he will not forget the XIII$^{\text{ème}}$ siècle! It was a great pleasure to meet him. I also learned a lot from him, about the community, the Japanese customs, and more recently about the local Japanese "mafia". In addition he does great sushis, and he has mysterious prediction skills...

Andrés Pérez-Uribe, whose motivation, happiness and curiosity is contagious, was also a core element of the lab. I miss those coffee breaks!

The lab would not have been what it is without many other people. Michael Bonani,

untiring “trottinette” driver, mechatronics expert, and beer connoisseur. Michael even has, unofficially, a bar named after him for the reason that I met him numerous times at that place in the early days. Francesco Mondada and Stéphane Magnenat, the free hardware and software “rebels” of the lab with whom it is always a pleasure to discuss about more technical, political and, especially with Stéphane, miscellaneous issues. I hereby also acknowledge that I failed at the “Stéphane” mission... Markus, who spends a lot of energy (but he has an almost infinite amount of it), to make the lab a better and more social place. In addition to snowball contests and other sports-related activities, he is also the worthy organizer of movie-nights. Markus is a great person to know, and I thank him for explicitly and implicitly helping me complete this thesis. Dominique Etienne, who took care of many of the more administrative aspects of the lab, and with whom it was always a pleasure to have a chat. I shall not forget Claudio Mattiussi, the thinker of the lab; Jean-Christophe Zufferey, the plane builder who entertains social groups by doing demonstration flights, especially late in the evening; Antoine Beyeler who is both good at science and parties, and who is improving his flying skills very fast; and my colleagues at ASL1 and ASL3. Thanks also to Cyrille Dunant for some LaTeX help as well as advice on how to organize some chapters at a critical point in the writing period.

I met some persons during crucial periods of my Ph.D., and I am very indebted to them. Arne Koopman is one of these persons. He came to the lab for his master thesis and we had very interesting discussions about development in multi-cellular circuits. He contributed a lot to my thesis. Thanks Arne! In addition he actively participated to the lab social life and he was the initial instigator of movie-nights. Diego Federici is another of these persons. He came to the lab for some months during his Ph.D. Since his research topic and mine were related we had the opportunity to exchange ideas and we had a fruitful collaboration. Diego’s knowledge and his passion for his research were a great source of motivation to me. I owe him a lot! By the way I finally found out the swan’s noise...

During these years I also met Prof. Mamoru Minami and Prof. Takashi Ikegami during their visit in the lab and I appreciated discussing with them. Takashi, next time we meet we really should find a pizzeria!

I had the chance to meet numerous students during my thesis. I am very grateful to all of them. In particular I wish to thank Stéphane Hofmann, Ludovic Righetti, Louis Villedieu, Thierry Frank and Tiago Bertolote, that directly contributed to this thesis in one way or another.

I did this thesis within the context of the European project POEtic. I am grateful to all the colleagues working on this project for the many ideas we exchanged during the past years. In particular I wish to thank Yann Thoma with whom I closely collaborated, and also Prof. Eduardo Sanchez and Prof. Andy Tyrrell who helped me during my thesis and with whom it was a pleasure to work.

I wish to thank my thesis examiners, Andy, Eduardo, Prof. Gianluca Tempesti and Prof. Xin Yao, for having taken the time to read my thesis and for the interesting discussions that followed.

Thanks also to the early readers of this thesis who gave me a lot of useful feedback, Dario, Mototaka, Claudio, Jean-Christophe, Markus, Danesh Tarapore, Stéphane and An-

toine.

I shall not forget my friends of *sorties* with whom it is always a pleasure to go out, visit various bars and clubs, and talk of many things (generally) not related to Ph.Ds! There are too many to mention them all here. In particular thanks to Yves, Laurent, Feride, François, Alex... and all the others!

My parents and family deserve my gratitude. Even though they did not see me much the last years, they continuously supported me during this time. I would not have succeeded without them!

Last but not least, Teresa deserves big and lovely thanks! She had to share me with my thesis, and during the writing months she did not see me very often. Teresa is a very kind person and I thank her wholeheartedly for her patience, for bearing with me and cheering me up during the difficult periods.

Contents

1 Introduction

1.1 Introduction

Electronic circuits are everywhere, from the simplest electronic toy or digital clock up to the most powerful computer. They are even used in the most hazardous environments such as in space where they may control satellites or exploratory rovers.

Increasingly complex circuits are designed, roughly following Moore's observation that the number of transistors in integrated circuits doubles about every 18 months [121], and nowadays integrated circuits containing more than 100 million transistors are common.

Yet, despite the success of designing increasingly larger circuits, current circuits may fall short in terms of robustness, of applicability to ill-defined problems, and of adaptivity to unexpectedly changing environments.

Electronic circuits are often brittle and offer little redundancy: a single defective transistor may lead to a failure of the entire circuit. Circuits usually cannot be designed unless the problem is fully specified, yet in some cases a specification may be difficult to obtain: the required models may be incomplete, or some highly complex functionalities may not lend themselves to analysis. For instance the controller of an autonomous robot may be difficult to design due to the complex relationship between what the robot perceives in its environment and the consequences of the motor actions with respect to the planned objective. Eventually circuits are designed to operate within precise environmental conditions and may not operate correctly when those change. For instance when the wheel of a planetary exploration rover fails, the electronics controlling the robot may need to be reconfigured.

In comparison, biological organisms are robust. The loss of a cell is generally not lethal, some organs are redundant, and in some cases limbs can even be regenerated through development [15, 183]. Biological organisms can achieve complex tasks in their environments, yet they arise from the relatively simple process of natural selection and differential reproduction also known as evolution [21]. Evolution is a continuous process that lets organisms adapt to changes in their environment, even on short time scales [6, 8, 133]. Furthermore organisms can learn, which allows them to adapt their behaviors in function of past events [30].

One way to compensate the shortcomings of current electronics is to take inspiration from principles observed in biology to design *bio-inspired electronic circuits*. In this way some of the characteristics of biological organisms, such as robustness, adaptivity or learning, may be provided to hardware.

The evolution of organisms over the course of generations (Phylogeny), the development of multi-cellular organisms from a fertilized egg (Ontogeny), and the learning capabilities that biological organisms exhibit during their lifetime (Epigenesis) are three sources of inspiration that may be used in bio-inspired electronics. These three sources of inspiration form the so-called POE model of bio-inspiration [139, 146].

Evolution inspired search and optimization algorithms known as evolutionary algorithms. These algorithms can be used to create or *evolve* electronic circuits; a discipline known as evolvable hardware [64]. Circuits can be evolved even with only a partial or high-level specification of the problem. Evolvable hardware is believed to have a lot of potential [191], for instance in adaptive hardware [78], or in fault-tolerant hardware [83], and it may allow to find novel or more efficient circuits than those obtained with traditional techniques [18, 165, 180].

Development can "grow" multi-cellular electronic circuits from a single cell and this may provide a self-reproducing or self-repairing substrate for the implementation of electronic circuits [109], as illustrated by the self-repairing BioWatch [149].

Learning inspired reconfigurable circuits that perform online categorization [128]. It also allows neural robot controllers to improve their response in function of past events, for instance to adapt to new environments or to new robots [44, 126]. Eventually learning may be used to discriminate correct from incorrect behaviors in electronic circuits [14].

1.2 Motivation and aims of this thesis

Current bio-inspired hardware tends to focus on a single aspect of bio-inspiration: either evolution, development, or learning. Yet the three aspects are complementary.

In this thesis we take the stance that bio-inspired electronic circuits should encompass all three aspects to fully benefit from bio-inspiration. This could for instance lead to circuits that are automatically evolved, that adapt to the characteristics of their environment, that are fault-tolerant, and that learn while operating.

We call these electronic circuits capable of evolution, development and learning *POEtic circuits* [178], referring to the POE model of bio-inspiration.

The integration of evolution, development and learning mechanisms in hardware translates into multi-cellular circuits, whose basic unit of organization is, like in living organisms, the cell.

Cells contain the genetic code of the entire circuit. They have a developmental mechanism that controls their functional differentiation according to the genetic code and to cellular and environmental signals. Cells have a functional part, that depends on the application. It may implement for example elementary logic gates or more complex functions such as neurons. Learning is implemented in the functional part of cells, for instance by using neurons capable of adapting their synaptic weights with a learning rule. Finally

the functionality of POEtic circuits is obtained automatically by evolution of the genetic code stored in the cells.

Conceptually this multi-cellular structure, together with the fact that cells contain the genetic code of the entire circuit and a developmental mechanism, allows the multi-cellular circuit to grow from a single cell and it allows self-repair mechanisms by having spare cells differentiate and replace at run-time the functionality of faulty cells. Furthermore it may allow dynamic reorganization of the structure of these circuits in order to cope with changes in the environment, or when they are expanded with new cells, sensors or actuators.

Since evolutionary, developmental and learning mechanisms may vary from applications to applications, POEtic circuits are not directly implemented in silicon, but instead they are obtained by configuring or programming a custom reconfigurable device known as the *POEtic chip* [163] with the desired mechanisms. This POEtic chip includes specific features required for the implementation of bio-inspired systems in hardware.

In this thesis we consider the mechanisms required for the evolution of these bio-inspired multi-cellular POEtic circuits.

In particular evolutionary algorithms commonly used in evolvable hardware are not well suited for multi-cellular POEtic circuits. We argue that to fully realize the potential of POEtic circuits a novel evolutionary system is required that takes into account their characteristics and that encompasses both a genetic encoding and a developmental system.

Indeed evolvable hardware, despite despite showing promising initial results, does not seem to scale to more complex circuits [59, 79, 82, 182]. We believe that this problem comes partly from the direct genetic encodings with one to one genotype to phenotype mapping that are often used. These encodings lead to genetic strings that grow with the size of the circuits and this may limit the scalability to larger circuits.

Furthermore, direct genetic encodings are not suited for multi-cellular circuits capable of dynamic reorganization such as POEtic circuits, since they assume that the number of elements in the circuit is known in advance and does not change throughout the life of the circuit.

Finally conventional evolutionary algorithms do not exploit the complex dynamics of development mediated by gene regulation that is observed in biological organisms. They use a static mapping from genotype to phenotype that does not allow the inter-cellular or environmental interactions during development which could provide fault-tolerance or dynamic reorganization of the circuit when environmental conditions change.

We therefore develop an evolutionary system suited for multi-cellular POEtic circuits. Notably we focus on developing an evolutionary system that attempts to provide better evolvability and scalability than direct genetic encodings, that allows cellular or environmental interactions during development, and that is computationally simple so that it can be efficiently implemented in hardware. We refer to this evolutionary system as the *morphogenetic system*.

We then use this system to evolve multi-cellular circuits for a wide range of applications that combine evolution, development and learning. We demonstrate the implementation of some of these circuits in hardware, either on field-programmable gate arrays (reconfigurable devices that can be programmed to implement electronic circuits) or on

the POEtic chip (a reconfigurable device designed specifically for bio-inspired applications).

1.3 Achievements

The main achievements of this thesis are detailed below, with references to the relevant chapters.

Evolutionary morphogenesis: We develop an evolutionary system (called morphogenetic system) suited for multi-cellular POEtic circuits that uses a genotype to phenotype mapping that takes the form of a simple developmental process inspired by the mechanisms of gene expression and cellular differentiation in biological organisms (chapter 6). We describe the multi-cellular implementation of the morphogenetic system in hardware. We show that it can be implemented with few resources and that it allows fast execution in constant time, regardless of the size of the circuit.

Applications: In order to demonstrate a multi-cellular architecture that is assumed by the morphogenetic system, we first show the evolution and growth (a simplified type of development with a direct genetic encoding) of a multi-cellular circuit on the POEtic chip. Growth means that cells of the circuit interconnect and differentiate at run-time in the chip. We demonstrate this circuit by evolving it to approximate Boolean functions and control the navigation of a robot (chapter 5). This circuit illustrates mechanisms that may lead to self-repairing circuits.

After introducing the morphogenetic system we use it in several multi-cellular applications. We use it to evolve structures of differentiated cells, which are a prerequisite for evolvability, and we show that it offers better scalability than a direct genetic encoding on the structures that are tested. Furthermore we show that the dynamics of the developmental process within the morphogenetic system provides tolerance to faults to these structures of cells (chapter 6).

We use the morphogenetic system to evolve multi-cellular circuits composed of spiking neurons to perform pattern recognition and to control the navigation of a robot in a task of obstacle avoidance. We demonstrate this multi-cellular controller embedded in hardware on a mobile robot. We find that the morphogenetic system outperforms a direct genetic encoding on these tasks (chapter 7).

The previous circuits are not capable of learning. We consider multi-cellular circuits capable of learning by employing a more complex spiking neural model which has a learning rule known as spike-timing dependent plasticity. We show that the morphogenetic system can evolve these circuits to improve their learning performance in a task that consists in learning and discriminating synthetic moving stimuli (chapter 8).

Finally we evolve a circuit to control the navigation of a mobile robot in a task that requires learning. In this task a robot provided with vision learns a particular moving visual cue and afterwards it takes a predefined action when it encounters the learned cue in the environment (chapter 9). This application demonstrates POEtic circuits comprising evolution, development and learning in a single task.

1.4 Structure of the thesis

Since this thesis deals with evolutionary mechanisms for bio-inspired hardware, we review first evolvable hardware in chapter 2. We introduce evolutionary algorithms, we describe the principles of the evolution of electronic circuits, and we highlight the strengths of evolvable hardware with examples of evolved digital and analog circuits. Finally we discuss the limitations of evolvable hardware, such as the issue of scalability, and we suggest that more complex, indirect genetic encodings may be one way to alleviate these limitations.

Indirect genetic encodings may take the form of developmental systems, that we review in chapter 3. We propose a classification of developmental systems applied to electronic circuits that is based on characteristics of their implementation. This classification evidences one class of developmental systems that we consider for the morphogenetic system.

In chapter 4 we describe a generic multi-cellular hardware architecture that allows the integration of evolution, development and eventually learning mechanisms. This architecture, when implemented in reconfigurable electronic devices, leads to POEtic circuits. A reconfigurable device ideally suited to implement POEtic circuits is the POEtic chip. We explain how the features of this POEtic chip are useful for hardware bio-inspired mechanisms and we show how this chip may be configured or programmed to implement POEtic circuits.

In chapter 5 we implement on the POEtic chip a multi-cellular circuit capable of evolution and growth (a simplified mechanism of development with a direct genotype to phenotype mapping) which demonstrates the architecture introduced previously. We evolve this circuit to approximate Boolean functions and to control a mobile robot in a task of obstacle avoidance.

In chapter 6 we introduce the morphogenetic system, the genetic encoding and developmental system designed for POEtic circuits. We analyze the evolvability, scalability and the tolerance to faults of the morphogenetic system in a synthetic application that consists in evolving structures of differentiated cells.

In chapter 7 we use the morphogenetic system to evolve circuits composed of spiking neurons for pattern recognition and robot control. These circuits are however not yet capable of learning.

The following two chapters deal with multi-cellular circuits capable of learning. In chapter 8 we describe a circuit composed of spiking neurons with a learning mechanism known as spike-timing dependent plasticity. We show that this circuit can be evolved with the morphogenetic system to improve its learning performance in a task that consists in learning and discriminating moving synthetic stimuli.

In chapter 9 we evolve with the morphogenetic system this same circuit to control the navigation of a mobile robot in a task that requires learning. In this task a robot learns moving visual cues and performs a predefined action when the learned cue is perceived in the environment.

In chapter 10 we conclude this thesis, highlight its contributions, and propose further research directions.

2 Evolution of electronic circuits

Abstract

Natural evolution relates to how biological organisms undergo natural selection and differential reproduction and change over the course of generations. It has inspired search and optimization techniques known as evolutionary algorithms. Evolutionary algorithms can be used to create electronic circuits: a field known as evolvable hardware. This chapter reviews evolvable hardware. It gives the principles of circuit evolution, introduces the evolutionary algorithms that are used to evolve circuits, and shows examples of evolved circuits that evidence the strengths of the evolvable hardware approach. In particular evolvable hardware allows to find novel, more efficient or unconventional circuits in comparison to those that are obtained with traditional design techniques. Finally this chapter discusses the limitations of evolvable hardware, notably the issue of scalability that limits the size of evolved circuits. Some ways to deal with this issue are described and the approach that will be followed to design the evolutionary system for multi-cellular POEtic circuits is outlined.

2.1 Introduction

Evolutionary algorithms are search and optimization algorithms that are inspired by the evolution of natural organisms. Evolvable Hardware (EHW) consists in using evolutionary algorithms to create hardware systems such as electronic circuits or mechanical structures (e.g. robot or antenna morphologies). This process works by evolving a set of instructions that describes how to build these systems. The objective of this chapter is to review evolvable hardware used to create electronic circuits.

Turing, in the 1940s, came up with the idea of unorganized machines that were “neural networks” composed of randomly interconnected two-input NAND gates. He proposed to use an evolutionary search method to train these networks [159]. He did not investigate this further, but this may be the first idea of EHW.

Nowadays there are reconfigurable devices that can be programmed to implement electronic circuits. The common definition of EHW was given in 1993 by Higuchi that

reported on the evolution of such a device [64]:

> This paper introduces an idea which the authors hope and believe will create not only a new branch of Artificial Life, but may also serve as the basis for a radically new approach to electronic and computer design. The idea can be expressed in two words, "evolvable hardware". Software configurable hardware, such as programmable logic arrays, are on the market which accept a bit string instruction which is used to configure or "wire up" a hardware circuit to give it a desired architecture. This can be done an indefinite number of times. By treating the bit string instruction as a Genetic Algorithm "chromosome", one has the means to evolve hardware.

In this thesis we consider EHW as the application of evolutionary algorithms to find circuits satisfying a predefined objective function. Circuits implemented in reconfigurable devices or assembled from discrete components, based on low-level building blocks (e.g. logic gates) or higher-level building blocks (e.g. neurons), are all particular instances of EHW as long as the building blocks are implemented in hardware and an evolutionary algorithm is used to find the functionality and interconnections of those building blocks.

EHW allows to create electronic circuits from high-level specifications. For instance an electronic circuit that controls a robot can be evolved by measuring the fitness of the circuit from the behavior of the robot, rather than from the precise relation between the inputs and outputs of the circuit. By using an evolutionary process, the creation of electronic circuits is also automated. Moreover, we will show in this chapter that novel and more efficient circuits can be found with EHW.

Section 2.2 gives the principles of the evolution of electronic circuits. Evolutionary algorithms that are used to evolve these circuits are introduced in section 2.3. In section 2.4 we highlight some of the digital circuits that have been evolved, focusing on those circuits that show the advantages of evolvable hardware. Section 2.5 deals in the same way with the evolution of analog circuits. Section 2.6 discusses the evolutionary approach to create electronic circuits and evidences some of its limitations, notably a scalability issue that tends to limit the size of evolved circuits. Some ways to deal with this issue are described in section 2.7. In section 2.8 we conclude and outline the approach that will be followed later in this thesis to design the evolutionary system for multi-cellular POEtic circuits.

2.2 Principles of circuit evolution

When an evolutionary process is used to create a circuit we say that the circuit is *evolved*, contrarily to the traditional approach where one *designs* a circuit. Evolving a circuit requires first to select the components and the interconnections between these components that are subject to evolution. Afterwards a genetic encoding that describes the configuration of the circuit is defined. Eventually a pool of circuits represented by their genetic strings are "bred" using an evolutionary algorithm (EA) with the objective of maximizing a *fitness* function that describes the adequacy of the circuits at performing a specific task.

All the design techniques can potentially be automated with an evolutionary approach. For this reason we briefly summarize how circuits are designed.

A circuit is usually first specified by an abstract description, and once it is fully specified this description is translated into a physical implementation.

Abstract descriptions include textual representations such as Hardware Description Languages (HDLs) or graphical representations such as schematics or state transition diagrams.

HDLs such as VHDL (Very High Speed Integrated Circuits Hardware Description Language) or Verilog describe a circuit in much the same way that source code defines a computer program. Hierarchical modular systems can be represented and modules can be described using instructions that represent the behavior of elementary logic gates (e.g. AND gates, flip-flops).

A schematic is a graphical representation of an electronic circuit, that depicts the elements of the circuit and the interconnections of these elements. The elements can be logic gates, or higher level blocks composed of several elementary logic gates. State transition diagrams are often used to describe the behavior of Finite State Machines (FSMs). A FSM defines a sequence of actions that depends on the internal state of the machine and on the state of the inputs. For instance a counter is a simple FSM. State transition diagrams graphically describe the transitions between the states of the machine, the conditions that trigger the transitions, and the effect of transitions on the outputs of the machine.

The circuit description can be implemented physically in different ways. Discrete components (e.g. resistors, transistors, capacitors) can be used and connected together by means of wires or on a printed circuit board. Since building circuits from discrete elements takes a lot of space, integrated circuits can be created. An integrated circuit consists of a single piece of silicon (a wafer) where all the discrete elements are implemented. Creating an integrated circuit is a costly and complex process that requires to draw the layout of the lithographic masks used in the manufacturing process. Manufacturing itself takes several months. Therefore integrated circuits are generally only done when a circuit must be produced in very large quantities.

Finally there are now reconfigurable devices that can be used to implement various electronic circuits by simple programming. Programmable Array Logic (PAL) are simple devices that can implement Boolean functions. Field-Programmable Gate Arrays (FPGAs) are devices that can implement any digital circuit, within the space available in the device [16]. These devices can be programmed from a schematic or a HDL description of a circuit using tools provided by the manufacturer of the device. These tools try to minimize the resources used on the device and maximize the operating speed of the circuit. This process is complex and can take from several minutes up to several hours. Eventually these tools produce the device configuration string that is downloaded to the reconfigurable device to program the corresponding circuit. There are also reconfigurable devices for analog circuits. They are known as Field-Programmable Analog Arrays (FPAAs) and are programmed in a similar way to FPGAs.

Circuits can be evolved by encoding in the genetic string the abstract representation of the circuit, for instance its schematic or its textual HDL description. The fitness of the circuit can be measured in a physical implementation of the circuit, for instance in a

reconfigurable device. However, since the conversion of the abstract representation into the device configuration string can take a lot of time, this is often impractical during evolution. Therefore the fitness is often measured in a simulation of the circuit, and only once the desired circuit is found it is physically implemented. Alternatively, the genetic string can consist directly of the configuration string of a reconfigurable device. In this way faster evolution can be achieved with the fitness directly measured in hardware.

Evolvable hardware can be classified depending on how the fitness is measured, where is situated the evolutionary algorithm, and how much constraints are applied on the evolved circuits. When the fitness of a circuit is evaluated from a simulation of the circuit, evolution is *extrinsic*. Instead, when the fitness is evaluated from a physical implementation of the circuit, evolution is *intrinsic* [24]. Evolution is *on-chip* when the evolutionary algorithm is implemented on the same physical medium as the circuits that are are evolved, and *off-chip* when the evolutionary algorithm is elsewhere (e.g. on an external processor such as a desktop computer) [173]. Finally evolution is *constrained* when only a subset of all possible circuit configurations are allowed, for example when a specific circuit topology is imposed. When the evolutionary process is free to explore all possible circuit configurations, evolution is said to be *unconstrained* [167].

2.3 Evolutionary algorithms

Evolutionary algorithms (EAs) are a family of search [26] and optimization algorithms that take inspiration from the process of selection and differential reproduction observed in natural evolution.

Evolutionary algorithms include genetic algorithms, genetic programming, evolution strategies and evolutionary programming. Those algorithms operate on a set of candidate solutions (i.e. points in the search space) that is often referred to as a population of individuals, chromosomes or genetic strings.

EAs require a scalar measure of fitness to operate. The fitness indicates the performance of candidate solutions at optimizing the problem at hand. EAs start with a randomly initialized population (i.e. the first generation). The fitness of each individual in the population is measured. If the best individual satisfies the optimization goal then the process stops. Individuals are then selected for reproduction according to their fitness: individuals performing well have a higher probability of being selected than individuals performing badly. Genetic operators such as mutation (that randomly modifies an individual) and crossover (that combines the genetic material of two individuals) are applied to the selected individuals to create a new population. This population forms a new generation. At this point the fitness of the individuals is again measured, and the process is repeated until an individual that satisfies the fitness criterion is found.

Genetic Algorithms (GAs) are generic algorithms that operate on a coding of the parameters of the problem [50, 70, 118]. This coding is often a simple string of bits or digits (the “genetic string”). This string is divided into genes that each represent one parameter of the problem. For instance an integer value can be represented by its binary coding. The process that decodes the genetic string into the parameters is called the genotype to

phenotype mapping. Since GAs operate on an encoded representation of the parameters, the algorithm needs not be modified when it is applied to other parameters (e.g. real values instead of integer values). This allows GAs to be very general optimization algorithms. When the genetic string is binary, the mutation operator consists in random bit flips and the crossover exchanges randomly selected substrings between two individuals. GAd are often used in evolvable hardware because the configuration of a circuit, such as a reconfigurable device, is easily represented by a string of bits.

Evolution Strategies (ESs) differ from GAs by operating directly on the parameters (i.e. the phenotype) rather than on an encoded representation of them [130, 144]. Individuals are composed of two vectors: the object-parameter vector and the strategy-parameter vector. The object-parameter vector represents the parameters that are optimized (e.g. real values). The strategy-parameter vector is of the same size as the object-parameter vector and its components control the mutation rate of the corresponding object-parameters. Mutation of the object-parameter is performed by adding a normal distributed random number, whose standard deviation is given by the strategy-parameter. Mutation of the strategy-parameter is performed by randomly increasing or decreasing the standard deviation. The crossover operator results in object-parameter and strategy-parameter vectors that inherit each element from one of the two parents with a 50% probability. ESs are sometimes used in EHW when a circuit is represented by a string of numbers, such as when schematics are evolved.

Genetic Programming (GP) allows to automatically create computer programs in the form of a tree structure [89, 92]. Each node of the tree represents a function, and the branches represent the data flow. The output of a node, that represents the return value of the node function, is sent across the branches to connected nodes. The program is interpreted by traveling the tree from the terminal leaves to the root. Terminal leaves represent input data to the program and the root of the tree is the output of the program. Genetic operators specific to the tree structure are used. Crossover exchanges randomly selected subtrees between two individuals. Mutation changes the function of a node or the values at terminal leaves.

Evolutionary Programming (EP) is used to evolve finite state machines [45]. The FSM is represented by two functions. The first one describes the state transitions based on the machine state and the inputs. The second describes the output of the FSM, also based on the machine state and the inputs. Genetic operators can modify these, e.g. by adding or removing states, changing the transition rules, or changing the outputs of the machine.

The key difference between these algorithms lies in the representation of the problem (encoding) and in the search operators they employ. They can be considered as a population-based version of a generate-and-test method. This allows to link these methods with other search methods such as simulated annealing or tabu search [192].

Bäck and Schwefel compared EP, ESs and GAs on a set of function optimization problems [7]. They pointed out that EP and ESs were more oriented towards parameter optimization, whereas GAs were more general purpose algorithms. In their experiments ESs were converging faster than GAs due to the self-adaptation of the strategy-parameter vector of ESs.

In EHW, GAs and ESs are the most commonly used evolutionary algorithms. GP

is rather used to evolve computer programs, although those may implement Boolean or analog functions akin to electronic circuits [89, 90]. EP is rarely used in the EHW community.

2.4 Digital circuit evolution

In this section we illustrate three approaches to the evolution of digital circuits: the extrinsic evolution of schematics, the intrinsic evolution of simple Boolean logic circuits, and the unconstrained evolution of FPGAs. We illustrate each approach by examples that show the strength of EHW and the specificity of evolved circuits.

2.4.1 Schematic evolution

Schematic evolution refers to the evolution of the schematic representation[1] of a circuit. Since implementing a circuit represented by a schematic in a physical device is a process that takes time and requires costly components (e.g. reconfigurable devices), schematic evolution is often extrinsic. For this reason feedback loops that may generate oscillating circuits are avoided because these are difficult to simulate accurately.

Schematic evolution can lead to circuits that implement known functionalities in novel ways. This was evidenced by Miller et al. that evolved a 6 tap digital finite impulse response (FIR) filter using only multiplexers [113] (i.e. the impulse response of the filter lasts 6 sampling periods). To evolve this circuit Miller et al. used a genetic encoding known as Cartesian Genetic Programming (CGP) [115]. CGP encodes the functionalities of the elements in the circuit and their interconnections with a string of numbers. All the signals available in the circuit (i.e. the inputs of the circuit and the outputs of the elements inside the circuit) are numbered sequentially. The functionality and connectivity of each element is encoded by a sequence of numbers (3 numbers for two-input elements, or 4 for three-input elements). The first numbers indicate to which signals the inputs are connected. The last number indicates the functionality of the element. The outputs of the circuit are encoded by additional numbers which indicate to which signals they are connected. The string of numbers is then evolved using an evolution strategy. Figure 2.1 illustrates this genetic encoding.

To obtain the FIR filter a 7x7 array of logic was evolved using CGP. Signal samples were provided as inputs to the evolved circuit, and the output of the circuit had to provide the filtered signal. Circuits that displayed the desired band-pass filtering characteristic were successfully evolved. The authors emphasized that no explicit multiplications or additions were used in the circuit and that there is no known mathematical way to design a filter using only multiplexers.

Schematic evolution can also lead to circuits that require less logic elements than their counterparts created with standard methods. This was demonstrated by Vassilev et al. that

[1] A schematic is a graphical representation of an electronic circuit, that depicts the elements of the circuit and the interconnections of these elements.

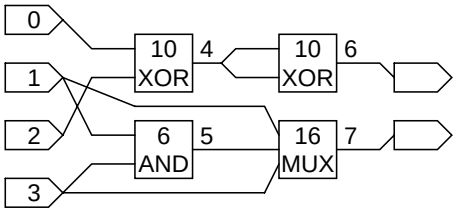

Figure 2.1: Encoding of a circuit with Cartesian genetic programming. The signals available in the circuit are sequentially numbered. Those signals include the inputs of the circuit, and the outputs of the logic elements. In the figure the signals are numbered from 0 to 7. The genetic encoding of the circuit consists of a string of numbers. Three (for two-input elements) or four numbers (for 3-input elements) are used to encode the connectivity and functionality of each element in the circuit. The first numbers indicate to which signals the inputs of the element are connected. The last number (indicated in bold) represents the functionality of the element (e.g. 10 is a XOR, 6 is an AND). In this example, the first element is encoded with the numbers "0 2 10". This means that the first input is connected to signal 0, the second is connected to signal 2 and the functionality is a XOR. The outputs of the circuit are also encoded by numbers that indicates to which signals they are connected. Here the outputs are connected to signals 6 and 7 (from [113]).

evolved 2x3, 3x3, 3x4 and 4x4 bit multipliers using CGP. They found circuits using less two-input gates than the best known conventional circuits.

CGP is not the only way to evolve schematics of circuits (see also [20, 166]). Since the authors did not compare the results obtained by CGP with other search methods (e.g. conventional genetic algorithm, simulated annealing), there is no indication that CGP is the best algorithm for this type of problems.

For example, using a different but related encoding, Coello et al. evolved several Boolean circuits from elementary logic gates that proved to be more efficient than those obtained using traditional methods [18]. These circuits were evolved from an array of logic gates where gates, in contrast to CGP, were only allowed to connect to others in the immediately preceding column. The fitness function was a multi-objective fitness function that rewarded circuits that had valid behaviors and that minimized the number of gates. Therefore there was explicit pressure to find circuits with a minimum number of logic gates. Kalganova et al. similarly used multi-objective fitness functions to evolve arithmetic circuits while minimizing their size [80].

Multi-objective fitness functions have also been used to evolve circuits that must satisfy several objectives, such as filters where constraints on signal phase and amplitude, as well as on the stability of the filter, must be satisfied [142].

2.4.2 Intrinsic evolution of Boolean logic circuits

Extrinsic schematic evolution requires a computer that simulates the circuits to measure their fitness, but in some applications high speed and compact implementations are required. In this case evolution is best performed intrinsically, by directly evolving the

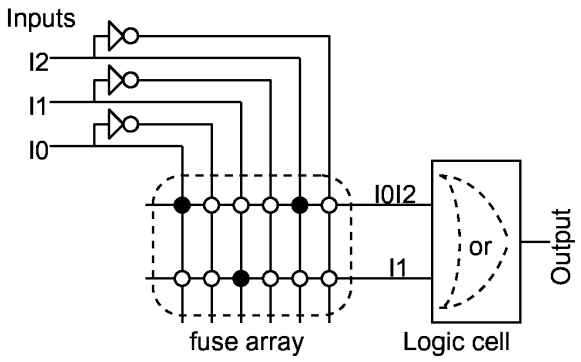

Figure 2.2: Simplified structure of a PAL. The output of the PAL is equal to the OR of several AND gates that are represented by the horizontal lines. The AND is done on the signals that are connected to the horizontal lines (illustrated by a black dot). In this example the first AND gate does the product between I0 and I2 and the output of the second AND gate is equal to I1. The output of the circuit is the sum (OR) of those two terms (from [66]).

configuration string of a reconfigurable device and measuring the fitness on the physical circuit. Reconfigurable electronic devices such as Programmable Array Logic (PAL), which are designed to implement Boolean logic function, can be used for this purpose.

Since any Boolean logic function can be represented by a sum of product terms, the PAL architecture is designed to allow an efficient implementation of those terms. The architecture of a PAL is illustrated in figure 2.2. Each output signal of the PAL comes from an OR gate that does the sum operation. The inputs of the OR gate come from programmable AND gates that do product operations. Each AND gate is represented by a distinct horizontal wire in the figure. The inputs of the AND gates are selected among the input signals and their inverse, or they can remain unconnected. Signals connected to the AND gates are represented by a black dot. In the figure, the upper input of the OR gate receives the product of $I0$ and $I2$. The lower input receives $I1$. Therefore the resulting Boolean function is $Output = I0 \cdot I2 + I1$. The connections to the AND gates are implemented with one-time programmable fuses, or reprogrammable memory elements. These circuits can be evolved by considering the state of all the connections of the AND gates as the genetic string of the circuit.

Higuchi was the first to show digital circuit evolution on a GAL16V8 chip [95] that is a particular type of PAL. In particular he evolved 4-bit multiplexers [64] and 3-bit counters [65].

Adaptive electronic controllers can be evolved in these reconfigurable devices. This was demonstrated by Kajitani et al. that designed a custom integrated circuit containing a large PAL with an embedded genetic algorithm. This circuit was used to control an artificial hand from myoelectric signals [78]. However, since myoelectric signals differ from one person to another, the electronic controller had to adapt to the user. To achieve this the PAL was evolved to select the appropriate actions of the hand according to the input signals, that were the frequency spectrum of the myoelectric signals. The fitness was proportional to the number of correct associations of input signals to actions. The authors compared the evolvable hardware controller to an artificial neural network (ANN) and

showed that classification of input signals was more precise with EHW (85%) than with an ANN (80%).

In this application, the key difference between EHW and ANN lies in the complexity (and therefore size in a hardware implementation) of the elementary functional blocks. Artificial neurons are more complex and take more space than the simple logic gates used in PALs. The authors pointed out that the EHW chip was better suited than ANN for small and embedded controllers because of its comparatively small size and quick adaptability.

2.4.3 Unconstrained evolution

FPGAs are reconfigurable devices that allow to implement more complex circuits than PALs. Unconstrained evolution refers to the evolution of FPGAs by using the configuration string of the FPGA as the genetic string of the evolutionary algorithm. Evolution is "unconstrained" because the user does not impose a predefined topology on the circuit that is evolved, nor does he limit evolution to a subset of the available functional blocks.

Since the evolutionary algorithm operates directly on the configuration string of the FPGA, no such string should damage it (e.g. cause short circuits). Only some FPGA architectures support unconstrained evolution. The XC6200 family of FPGAs from Xilinx is one of those [189]. The XC6200 is organized as an array of programmable cells with resources for routing between cells. Figure 2.3 represents a simplified view of a cell. Cells have one output and one input on each side (north, east, south, west). Inside each cell there is a function unit that can perform a logic function or serve as a memory element. The inputs of the function unit and the outputs of the cell are selected by multiplexers. The configuration of all these multiplexers forms the configuration of the cell. The configuration of an array of cells is obtained by concatenating the configuration bits of all the cells in the array. The FPGA is organized such that outputs of a cell are always wired to inputs of adjacent cells. In this way short circuits are impossible, irrespective of the FPGA configuration.

Unconstrained evolution may allow to find circuits that operate using different principles than those that are used when designing circuits. This may lead to circuits that are more efficient than circuits that are designed (e.g. using less components). This was first demonstrated by Thompson that evolved a circuit capable of discerning between an input signal oscillating at two different frequencies [165, 166]. This circuit was evolved so that its output was low when the input signal was oscillating at 1 kHz and high when the input signal was oscillating at 10 kHz. An array of 10x10 cells in the XC6200 FPGA was used for this purpose, resulting in a genetic string that was 1800 bits long. The challenge of this experiment was that no reference clock signal was given to the circuit. Nevertheless, after 3500 generations the desired circuit was obtained. Figure 2.4 illustrates the output of the circuit at different generations for the two input signals. Analysis revealed that the evolved circuit might have used physical characteristics of the chip such as gate propagation time to discriminate between the two frequencies. In comparison, designing a circuit for this task would have required an external time reference (e.g. an oscillator).

Thompson showed that circuits obtained by unconstrained evolution adapted to the physical characteristics of the chip on which they were evolved, and on environmental

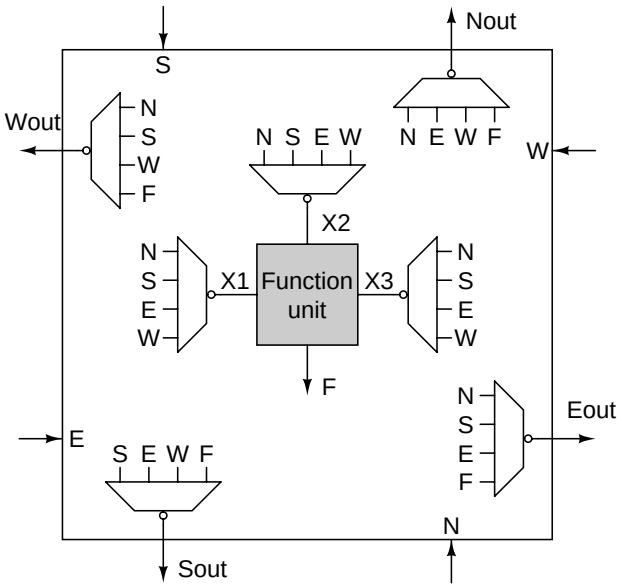

Figure 2.3: The simplified view of a XC6200 cell. The cell has one input and one output on each side. The function unit implements a logic or a memory element. The inputs of the function unit and the outputs of the cell are selected by multiplexers.

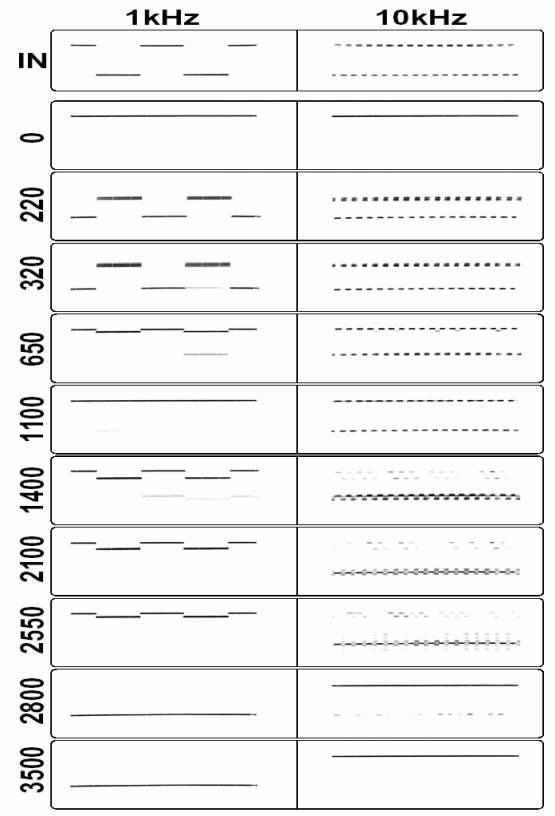

Figure 2.4: Output of the frequency discriminator at different generations for the 1 kHz and 10 kHz signals. After 3500 generations the input signal is correctly detected and the output is low with the 1 kHz signal, and high with the 10 kHz signal (from [165]).

conditions such as the temperature. When the same evolved circuit was implemented on another chip, or tested at another temperature, the resulting fitness dropped. These variations are caused by the fact that different chips have slightly different characteristics, within the tolerance of the manufacturing process. Furthermore many characteristics of electronic components show a dependency with temperature. Thompson showed how to cope with these problems by evolving simultaneously several circuits on different chips and at different temperatures [164].

Unconventional circuits were also obtained by Huelsbergen et al. that successfully evolved circuits that oscillated at various low frequencies (from 20 kHz to 80 kHz) on a XC6200 FPGA in a setup similar to that of Thompson [73]. In particular the circuits did not receive any external time reference. The authors observed that the temperature influenced the oscillation frequency of the circuits. They suggested that this might be exploited by evolution to create a temperature feedback loop that might lead to circuits operating independently of the temperature.

2.5 Analog circuit evolution

Some applications are well suited to analog processing, e.g. filters, oscillators, amplifiers. For these applications, evolving analog circuits may thus be more appropriate than evolving digital circuits.

Extrinsic evolution can be done using analog circuit simulators such as SPICE (Simulation Program with Integrated Circuit Emphasis) [91]. In this section we focus on the intrinsic evolution of analog circuits that is a recent development in EHW. This may come come from the fact that commercial reconfigurable analog devices were introduced after their digital counterpart. In the following subsections we will describe the three most common categories of intrinsic analog evolution, that are the evolution of FPAAs, the evolution of custom generic reconfigurable boards, and the evolution of custom integrated circuits.

2.5.1 Field programmable analog arrays

FPAAs are reconfigurable analog devices that can be programmed to implement many analog functions such as filters, oscillators, comparators, etc. They have the advantage of integrating several discrete components in one chip. In addition they offer reconfiguration, that can be exploited to change the characteristics of a circuit without having to replace its components. FPAAs are generally based on operational amplifiers. Operational amplifiers are components that amplify analog signals with a high gain and they are the elementary building blocks of many analog functions.

The configuration string of FPAAs can be evolved in the same way as the configuration string of digital reconfigurable devices, therefore allowing the evolution of analog circuits.

The Zetex TRAC (Totally Reconfigurable Analog Circuit) is a FPAA composed of 20 operational amplifiers [195]. The inputs of the operational amplifiers can come from an external pin, or from another operational amplifier in the device. According to the config-

uration string, passive components of pre-defined values integrated in the device (resistors, capacitors and diodes) are connected to the operational amplifiers. This enables operational amplifiers to take different functionalities: integrator, differentiator, pass-through, adder, etc.

Flockton et al. demonstrated the feasibility of intrinsic evolution of analog circuits on the TRAC. They successfully used a GA to evolve a circuit that matched closely a predefined input-output transfer function [38]. They also evolved analog bandpass filters using a set of external resistors [39].

The MPAA020 is another FPAA that was developed originally by Motorola [123] but that is now owned by Anadigm and sold as the AN10E40 [3]. This FPAA is composed of 20 Configurable Analog Blocks (CAB). Each CAB is composed of an operational amplifier, passive components, and programmable switches that interconnect the components with the operational amplifiers. Compared to the TRAC, this device offers programmable passive components (e.g. the value of the integrated resistors is specified by the configuration bits). This is done using a switched capacitor technology: by switching a capacitor between two terminals at various frequencies, the quantity of charges transferred between the two terminals can be modulated, thereby simulating resistors of different values.

Zebulum et al. demonstrated that evolution could handle circuits based on switched capacitor technology by evolving intrinsically an oscillator on the MPAA020 [193]. This is an important point since switched-capacitor technologies tend to introduce noise in the signals that may perturb the measurement of the fitness.

2.5.2 Custom generic reconfigurable boards

Commercially available FPAAs impose constraints on the circuits that can be evolved since these circuits must use the functional blocks provided in the FPAA (e.g. operational amplifiers), the available passive components, and the interconnections predefined by the architecture of the FPAA. It is therefore not possible to evolve circuits using other functional blocks, possibly more adapted to a particular application (e.g. transistors instead of operational amplifiers). Furthermore, in order to understand an evolved circuit it is often necessary to perform measurements on the internal nodes of the device (i.e. the output of internal function blocks). This is often not possible in FPAAs.

For these reasons, custom generic reconfigurable boards were developed for the evolution of analog circuits. These boards allow to evolve circuits based on components provided by the user. Furthermore all the internal nodes are accessible for measurements.

The Evolvable Motherboard (EM) is one of those reconfigurable boards that was developed for intrinsic hardware evolution (figure 2.5) [96, 97]. The EM is composed of 6 programmable crosspoint switch arrays. Each of these arrays is composed of horizontal and vertical wires with a programmable switch at each intersection. By programming the switches, the horizontal and vertical lines are connected. In each array a daughterboards can be plugged in. Daughterboards allow to connect external components (e.g. resistors, capacitors, transistors, operational amplifiers) on the signal lines of the array. In the fig-

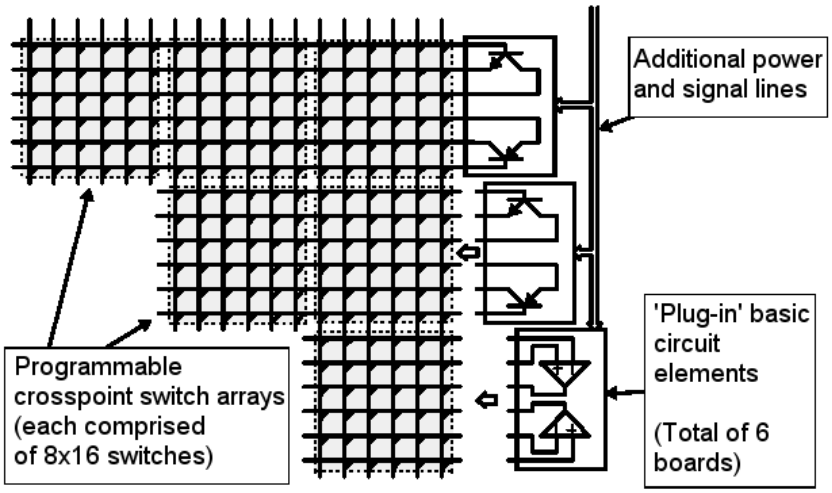

Figure 2.5: Simplified representation of the EM. The six programmable crosspoint switch arrays are visible on the left. On the right three daughterboards are connected to the evolvable motherboards to provide external components. In this figure the top two daughterboards contain transistors while the third one contains operational amplifiers. By evolving the connections between horizontal and vertical lines circuits composed of transistors and operational amplifiers can be evolved (from [96]).

ure, the top two daughterboards contain NPN and PNP bipolar transistors[2], while the third contains two operational amplifiers.

The genetic string of the circuit is the configuration of all the interconnection switches (on or off). Hence circuits containing any of the components connected to the evolvable motherboard via the daughterboards can be evolved. The EM was used to evolve digital inverters [96], and inverting amplifiers and oscillators [97] from PNP and NPN transistors.

The Programmable Analog Multiplexer Array (PAMA) is another generic reconfigurable board [140, 194]. The core of the PAMA is an analog bus of 8 lines. Some of those lines are used for external signals (input and output signals, ground and power supply) and the others are used for internal signals. External components are connected to the analog bus using programmable analog multiplexers. The configuration of the multiplexers indicates to which bus line the pins of external components must be connected. In this way external components sharing the same bus line are interconnected, or they are connected to the inputs, outputs or power supply via the corresponding bus lines. The genetic string of the PAMA is the configuration of all the multiplexers.

Zebulum et al. evolved an inverter by connecting NPN transistors and resistors to the PAMA [194]. Santini et al. presented an improved version of the PAMA, with an analog bus of 16 lines [140]. They evolved a XOR gate and a 2 input analog multiplexer from PNP and NPN transistors and resistors.

In many experiments, both with the EM and the PAMA, several uncommon circuits

[2]In a very simplified way NPN or PNP indicate whether the transistors conduct current when the input is low or high.

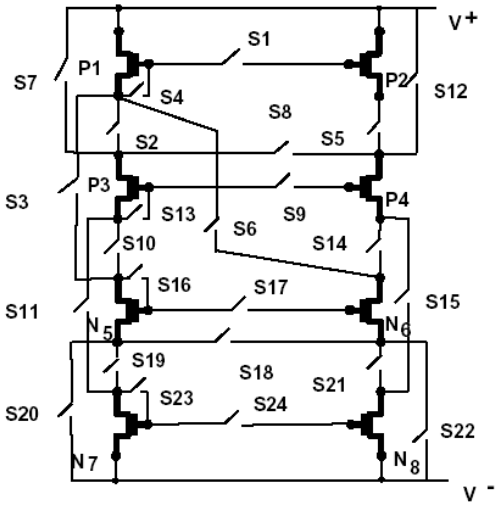

Figure 2.6: Illustration of the transistor array of the FPTA. It consists of 8 transistors and 24 programmable switches (from [151]).

were evolved. For instance components sometimes had floating pins. Evolution also exploited physical characteristics of components in several experiments. This was evidenced after replacing a component with another one of the same type and noticing a decrease in the fitness value.

These reconfigurable boards can be easily and cheaply built using commercially available components and they offer a lot of flexibility. They do however take more space than FPAAs since discrete components are used.

2.5.3 Custom reconfigurable integrated circuits

Generic reconfigurable boards lack the integration seen in FPAAs but they allow the evolution of circuits with any type of user-defined components. An alternative is to develop custom reconfigurable integrated circuits that are specifically designed for evolvable hardware. By developing a custom circuit, a high degree of integration can be achieved, and the circuit can be designed to include all the components required for a particular application.

This approach was followed at the NASA Jet Propulsion Laboratory (JPL) where a reconfigurable integrated circuit was developed for evolutionary applications. The resulting device is referred to as the Field-Programmable Transistor Array (FPTA) [150, 151, 154].

The FPTA is a transistor-level reconfigurable device. Transistors are used to allow both analog and digital circuit evolution in the same device.

The first generation FPTA consists of an array of 8 transistors (4 NMOS and 4 PMOS) interconnected with 24 programmable switches [150], as illustrated in figure 2.6. The switches are in sufficient number to allow many topologies commonly used in circuits. Larger circuits are obtained by connecting several FPTAs together. Higher level building blocks can be obtained by freezing the connections between the transistors in a FPTA.

Therefore, once configured, a single FPTA can serve as a custom function block.

The genetic string of the circuit consists of the configuration of all the switches. The FPTA was used to evolve circuits having a Gaussian current-voltage response, inverters [151], NAND gates [153], and T-norm and S-norm gates for fuzzy logic [152]. In some runs the evolutionary process even rediscovered equivalent human designed circuits.

The second generation FPTA (FPTA2) was developed to handle more complex circuits. The FPTA2 is composed of 64 arrays of transistors and a transistor array now contains 14 transistors and 44 programmable switches and it includes programmable resistors and capacitors. The FPTA2 also includes a vision sensor, which makes it an evolvable sensor system [154].

An approach similar to that of the NASA JPL was followed by Langeheine and others, using a different FPTA architecture [94].

Evolvable hardware can lead to fault-tolerant circuits. This was demonstrated by Keymeulen et al. on the FPTA [83]. Two approaches to fault tolerance were considered. In the first case, when a fault occurred, the FPTA was reconfigured with each of the circuits obtained in the last generation, until one that happened to perform well with that fault was found. In the second case faults known to occur were introduced during the evolutionary process, leading to circuits that were robust to several faults. In both cases significant fault-tolerance was observed.

Another approach to analog processing consists in using a digital representation of analog signals. This is the objective of the Palmo system that uses a custom chip architecture that processes analog data with a digital pulse-based representation [60]. The magnitude of a signal is encoded by the width of the digital pulse, and the sign of the signal is determined by the occurrence of the pulse with respect to a global sign clock. This digital encoding allows to transmit signals on long distances and it is less sensitive to noise than analog encodings. The basic computational blocks used in the Palmo system are cells acting as differential integrators with a programmable scaler. A chip containing an array of Palmo cells was manufactured and tested with simple circuits (e.g. filters, Σ-Δ modulators) but evolution was not yet performed.

2.6 Discussion

In the previous sections we showed several examples of novel circuits obtained with EHW. Some of these circuits are unconventional circuits. They perform the desired functionality in another way than a circuit which is designed would (e.g. some evolved circuits operate with components that have floating pins). Evolved circuits may exploit physical characteristics of the substrate on which they are implemented. They may provide adaptive hardware, by letting the circuit evolve until it satisfies a desired degree of adaptivity. Evolved circuits may use less logic gates than their counterparts which are designed. Eventually evolved circuits may provide tolerance to faults, either by letting the circuit re-evolve in case of faults, or by subjecting the circuit to faults during evolution.

Miller et al. summarized the results obtained with EHW by suggesting that this approach could explore a larger set of circuits than what a human designer could. This

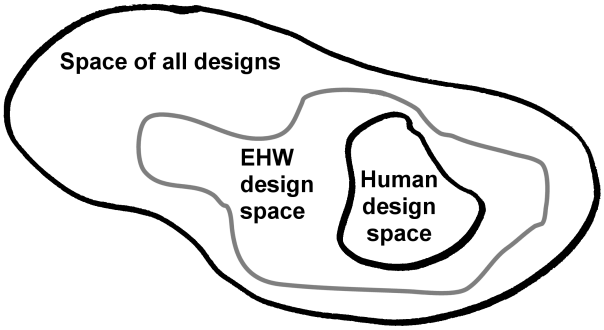

Figure 2.7: The EHW design space may grasp a larger part of the space of all designs than the human design space, thereby leading to novel or unconventional circuits (from [114]).

potentially allows EHW to discover circuits unknown to engineers or out of reach of traditional design techniques [114] (figure 2.7).

Indeed evolved circuits need not follow the hierarchical decomposition commonly found in circuits that are designed. Therefore evolution may find more compact circuits by eliminating the redundancies that may exist between building blocks used in traditional designs. Mathematical models of circuits are also used in the design process, and design rules are followed to stay within the validity range of these models. For instance digital sequential circuits are designed so that all the signals change synchronized with a global clock. Unconstrained evolution shows that new circuits can be found when these rules are removed. In particular FPGAs that are designed for digital synchronous logic may be used as continuous time recurrent circuits by evolution.

Evolvable hardware however also has some drawbacks. Evolved circuits may lack understandability and their operational envelopes may be difficult to determine (e.g. the temperature range in which circuits behave correctly). Therefore evolved circuits may require extensive testing to ensure correct behavior in all the operating conditions before they can be used in critical systems.

Furthermore evolution must sometimes proceed for several thousand generations before the desired circuits are obtained. This is especially the case with unconstrained evolution that tends to generate huge search spaces (genetic strings of several thousand bits). Such long evolutionary runs may be impractical in applications that require fast evolution, for instance to recover from faults in real-time.

More importantly EHW seems to encounter an issue of scalability that limits the size of the circuits that are evolved [191, 182, 82, 79, 59]. The scalability issue is that the larger circuits become, the increasingly harder it is to evolve them. One of the causes of this problem may be the direct genetic encodings that are generally used in evolvable hardware. These encodings assign one gene to each element of the circuit. Therefore the size of the genetic string grows with the size of the circuit. The size of the search space of larger circuits thus explodes, and evolutionary search may not find circuits satisfying the objective function. The following section describes some ways to deal with this aspect.

Another limitation of direct genetic encodings in evolvable hardware is that the genotype to phenotype mapping is static. Environmental interactions are not taken into account

when the genetic string is decoded. Therefore these genetic encodings do not fully take advantage of the complex dynamics of gene regulation observed in biological organisms, that might be exploited to provide fault tolerance to the circuit or adaptivity to the environment.

2.7 Towards more complex evolved circuits

Different approaches have been proposed to tackle the problem of scalability in EHW and evolve more complex circuits.

Indirect mappings between the genotype and the phenotype that allow for gene reuse may be used [190]. Such a mapping may take the form of a developmental system where the genetic string contains instructions that indicate how to build a larger circuit. This approach is reviewed in chapter 3. Variable length genetic encodings may also be used. For instance Higuchi et al. used variable length chromosomes to evolve a PAL and they observed that they could evolve larger circuits with variable length chromosomes than with a simpler GA [66].

Incremental evolution may be used to evolve simpler subsets of the target problem. For instance, instead of evolving a single digital circuit with several outputs, several subcircuits with a single output can be evolved [18, 79, 174].

High-level building blocks may also be introduced. These building blocks may be found by the evolutionary process, such as automatically defined functions in GP [91], or they can be defined from previously evolved circuits [182]. Predefined high-level blocks may also be used, such as artificial neurons [47].

In the EHW community simple evolutionary algorithm are often used. However, optimized algorithms may be devised to better handle some problems. For instance diploid genes may improve the adaptation of electronic circuits to changing environments [68], and algorithms combining genetic algorithms and simulated annealing show promising results [99].

Since the structure of the fitness landscape may influence the difficulty of evolutionary search [107], selecting an appropriate genetic encoding and appropriate building blocks may lead to fitness landscapes that are better suited for evolutionary search. This is not a trivial issue since characteristics of the fitness landscape sometimes fail to distinguish between easy and hard problems [147]. However neutrality[3] seems to favor the evolution of digital circuits [181] and genetic encodings that allow for neutrality may be designed to improve the evolvability of circuits [180]. Analog circuits are also believed to induce smoother fitness landscapes that may improve evolutionary search [151, 194].

[3]Neutrality is a characteristic of the fitness landscape that indicates that it is possible to go from one point of the search space to another one of identical fitness by application of the genetic operators, by following a path of constant fitness.

2.8 Summary

In this chapter we reviewed some of the applications of EHW in digital and analog circuits. We showed that EHW could find more efficient circuits than those obtained by design, that it could find unconventional circuits by exploiting physical properties of the hardware, and that it could lead to adaptive or fault-tolerant hardware. Nowadays EHW also gains support in the industry. For instance it is used to optimize the clock distribution in high-speed systems, it can be used to compress data in printers, and it allows to adapt analog filters in cellular phones after manufacturing [67].

Finally we discussed the problem of scalability that limits the size of evolved circuits and we highlighted some of the approaches that may be followed to tackle this problem. In this thesis we will follow three of these approaches to develop the evolutionary system for multi-cellular POEtic circuits in chapter 6. We will use an indirect genotype to phenotype mapping that allows for gene reuse, we will use predefined high-level building blocks, and eventually we will use building blocks that can process analog values, since these may induce smoother fitness landscapes that may improve evolutionary search. In particular we will work with spiking neurons that transmit information with binary spikes, but that can encode analog values in the temporal patterns of spikes. Spiking neurons have another advantage since they may be used as compact neural controllers for robots [43]

3 Developmental systems for electronic circuits

Abstract

In chapter 2 we reviewed evolvable hardware and we evidenced a problem of scalability that we linked to the direct genetic encodings that are often used in this field. Since direct genetic encodings assign one gene to each element of a circuit, the genetic string grows with the size of the circuit and leads to large search spaces. Developmental systems may alleviate this problem by providing more efficient indirect genotype to phenotype mappings. Furthermore the dynamics of development may allow inter-cellular or environmental interactions during development that might lead to adaptive or fault-tolerant electronic circuits. The purpose of this chapter is to review developmental systems used for evolvable hardware and classify those according to characteristics of their hardware implementation. Finally we conclude highlighting two points that we will consider to design the developmental system for multi-cellular POEtic circuits in chapter 6.

3.1 Introduction

Chapter 2 evidenced the problem of scalability that is faced in evolvable hardware, and linked it to direct genetic encodings that are often used in this field. These encodings assign one gene to each element of a circuit. Therefore the genetic string grows with the size of the circuit and leads to large search spaces. Kitano, that reported on the early evolution of neural networks with direct genetic encodings, also noted the issue of scalability, together with the lack of biological plausibility of these encodings [84]:

> There are two major problems in these approaches. First, none of these studies carried out systematic experiments in terms of scalability and the speed of convergence, leaving applicability of the schemes for designing larger networks open to questions. [...] Second, these methods are biologically unfeasible, because they assume that connectivity information is encoded in the DNA in almost one-to-one correspondence. This leads to two major problems: (1) it cannot capture morphogenesis of neural systems, and (2) sufficient information cannot be encoded in DNA of the given length. Thus,

existing methods do not take full advantage of using genetic algorithms for neural network designing.

A possible way to ensure better scalability is to use indirect genetic encodings with a genotype to phenotype mapping that takes the form of a developmental process [85, 190]. Therefore a small number of "instructions" in the genotype may describe how to "grow" a larger phenotype, thus reducing the size of the search space.

In addition direct genetic encodings do not capture the complex dynamics of development mediated by gene expression which is seen in biological organisms. Developmental systems may allow inter-cellular or environmental interactions during development which could provide circuits with fault-tolerance or dynamic reorganization capabilities to cope with environmental changes.

A developmental system may consists of abstract instructions encoding the phenotype, it may be biologically inspired, or seek biological plausibility. Such indirect genetic encodings are referred to as embryogenics [85], embryogeny [9], artificial ontogeny [12, 131], morphogenic evolutionary computation [4], or artificial embryology [23]. We refer to them simply as *developmental systems* to avoid the biological connotation of the above terms, as several genotype to phenotype mappings may be unrelated with biology.

The purpose of this chapter is to review developmental systems applied to evolvable hardware and the applications that are the result of such a combination. We also propose a classification of these developmental systems which is based on characteristics related to their hardware implementation. This classification evidences one category of developmental systems that we believe is promising for evolvable hardware.

Since biological development is often a source of inspiration to design developmental systems, the principles of biological development are briefly introduced in section 3.2, together with some common mathematical abstractions of development in section 3.3. Developmental systems used for evolvable hardware are reviewed in section 3.4. Developmental systems may also be used to grow neural networks or morphologies, and an overview is given in section 3.5. A classification of developmental systems used for evolvable hardware is proposed in section 3.6. Finally section 3.7 concludes this chapter by highlighting two points that will be considered to design the evolutionary system for multi-cellular POEtic circuits in chapter 6.

3.2 Biological development

The main concepts of biological development are summarized here. A full treatment of this topic may be found in [19, 188].

The development of a multi-cellular organism follows the following stages:

1. Fertilization is the process by which the genetic material of a sperm cell and an egg cell are merged to create the embryo. Fertilization initiates the development process.

2. Cleavage consists in rapid cell division starting from the fertilized egg. During cleavage the total size of all the cells remains the same (figure 3.1). After about

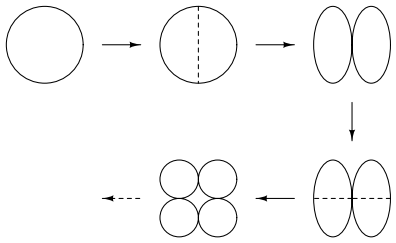

Figure 3.1: Illustration of the process of cleavage by which cells undergo division but the total size of the cells remains identical.

10 divisions some changes start to appear and three regions can be identified: the mesoderm which will give the muscles and bones, the ectoderm which will form the nervous system and the endoderm which will become the gut.

3. During pattern formation the major body axes are defined, such as the anterior and posterior ends and the dorsal and ventral sides. Initial asymmetries in the egg may be used to establish those body axes.

4. Gastrulation consists in the movement of sheets of cells. The mesoderm, ectoderm and endoderm are rearranged; this leads to a change in form of the organism, also called morphogenesis.

5. Cell differentiation then occurs: cells become structurally and functionally different, e.g. muscle cells, skin cells.

6. Growth occurs after the form of the embryo is defined in a small scale. The size of the organism increases either by cell multiplication, cell growth, or deposition of external material (e.g. bones).

The development process is controlled by the genes which are encoded in the DNA of every cell. Genes contain all the "instructions" necessary to build proteins. Some of those proteins are necessary for the life of the cell (e.g. to break down molecules to produce energy). Some are used to implement cell-specific functionalities (e.g. transport of oxygen by blood cells). Eventually some control development.

However genes are not always interpreted or expressed, and the corresponding protein is not always produced. When a gene is interpreted it is said to be activated, and when it is not the gene is said to be repressed. The degree of expression of a gene is regulated by *transcription factors* that are specific proteins regulating the expression of genes. Those transcription factors are themselves the product of other genes. Therefore complex dynamic patterns of gene activation and repression arise in cells. This is referred to as *gene regulatory networks* (GRNs).

The expression of a gene is controlled by a part of it which is called the *control region*, whereas the instructions necessary to build proteins are in the *coding region*, as illustrated in figure 3.2. The control region, also called the regulatory region, contains *enhancer regions* to which the transcription factors can bind to activate or repress the transcription

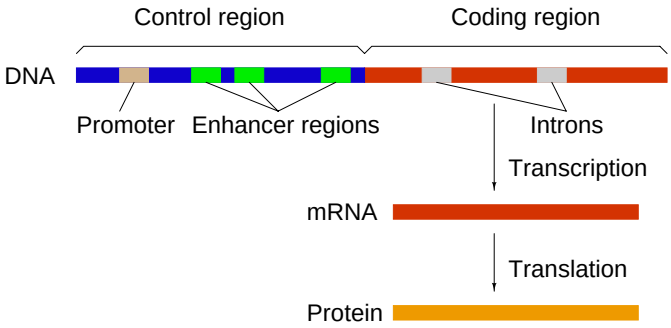

Figure 3.2: Structure of a gene and the process of production of proteins. Genes are composed of a control region and a coding region. The coding region contains the instructions necessary to build a protein. The control region contains several docking sites, called the enhancer regions, where transcription factors (proteins) can attach to activate or repress the transcription of the gene. Gene expression starts at the promoter region if the correct transcription factors are present. During the process of transcription the non-coding parts called introns are removed and messenger RNA is synthesized which is then translated into proteins.

of the gene. There can be several enhancer region and therefore complex arrangement of transcription factors may be necessary to activate the gene.

Gene expression starts from a region called the promoter region. A special molecule called RNA polymerase docks on this region. If the correct transcription factors are present it unwinds the DNA, interprets the gene and synthesizes the corresponding mRNA (messenger RNA). Therefore when a gene is activated production of mRNA follows. This phase is called transcription. During transcription, the non-coding parts of the gene called introns are removed. The mRNA is then translated into a protein that eventually folds in a 3D shape.

Genes control the development of organisms through the proteins they produce. Different patterns of gene expression lead to cell differentiation, to changes in cell shape or to cell movements which cause morphogenesis. Cell division and programmed cell death (apoptosis) are also under the control of genes.

Gene expression can be affected by external factors, for example environmental changes, but also by signals coming from other cells. The process by which a signal from a cell influences the development of another one is called *induction*. Inductive signals may be chemicals diffusing among cells on long distances or they may be molecules exchanged locally on the surface of cells which are in contact. Only cells which are in an appropriate state respond to inductive signals.

Positional information can also lead to specific patterns of gene expression. This is an important step during pattern formation. Positional information can be conveyed by the intensity of a chemical which varies in space. Such a chemical may come from the initially asymmetric distribution of yolk in the egg. When a chemical is conveying such a positional information it is called a *morphogen*.

T		T+1
$I1$	$I0$	O
0	0	0
0	1	0
1	0	1
1	1	0

Table 3.1: This truth table represents the new state of a gene (O) at the next time step (time T+1) in function of the current (time T) state of the genes that control its expression ($I0$ and $I1$).

3.3 Mathematical models of biological development

Even though the biological mechanisms of development are rather complex, there are simple mathematical abstractions that can describe some aspects of development.

3.3.1 Random boolean networks

Random boolean networks (RBNs) were proposed by Kauffman to study the hypothesis that organisms may be randomly constructed molecular automata [81]. To this end he developed a mathematical simplification of the dynamics of gene regulation where genes are considered as binary: either fully activated or repressed.

A RBN is a collection of genes which are updated (activated or repressed) in discrete time steps. The state of a gene is updated in function of the state of other genes that control its expression. This may be represented by a truth table, as in table 3.1, that represents the new state of the gene (i.e. whether the gene will be activated or repressed in the next time step) in function of the state of the genes that control its expression.

Kauffman considered genes that have an identical number of inputs K. RBN genes are "random" in the sense that the inputs of genes are randomly selected among N genes in the network. Also, the truth table of every gene is different and randomly selected among the 2^{2K} possible truth tables. Kauffman showed that RBNs can exhibit stable cycles (point attractors or cycle attractors) that may be identified to a cell type, that the number of distinguishable cycles can predict the number of cell types in an organism with a similarly sized genetic network, and that RBN can change operating mode (cycles) like cells do differentiate.

Since their inception other types of RBNs have been proposed, e.g. with asynchronous or non-deterministic update. See [48] for a recent classification of RBNs. More realistic models of gene regulations consider genes that have different degree of expression. These models are referred to as gene regulatory networks (GRNs).

3.3.2 L-Systems

L-Systems are mathematical models proposed by Lindenmayer to describe the development of multi-cellular organisms [100]. Cells have a state, an input and an output. The cell

is considered as a sequential machine which changes state in function of its inputs and its current state. The outputs also depend on the current cell state and inputs. Lindenmayer describes the next state of a cell as a generating function δ:

$$\delta(p, q) = r.$$

In the notation p and q represent the current state and input of the cell and r is the new state. The output of the cell is represented by:

$$\lambda(p, q) = u$$

with the same notation as above, u representing the output of the cell. Until now this description is similar to a Mealy machine in electrical engineering. The difference comes from the fact that cell-division can be represented by having a sequence of two or more states as the output of the generating function δ. For example the generating function

$$\delta(a, 1) = ab$$

replaces the state a by states a and b when the input is 1, thereby increasing the length of the organism.

L-Systems can also be seen as a formal grammar that consists of symbols, rewriting or production rules and a start symbol. Rewriting rules have a left-hand side and a right-hand side consisting of a string of such symbols (or words):

$$LHS \rightarrow RHS.$$

Rewriting rule are applied to strings or words by replacing the left-hand side by the right-hand side. This process begins from the starting symbol.

For example consider the system composed of the following rewriting rules:

$$S \rightarrow AB$$

$$A \rightarrow AB$$

$$B \rightarrow b$$

Starting from the symbol S, it gives the following developmental steps:

Step 0: S
Step 1: AB
Step 2: ABb
Step 3: $ABbb$

L-Systems have been used to model plant growth. Consider the following rewriting rules:

$$S \rightarrow F$$

$$F \rightarrow F[LF][RF]$$

where F draws a segment of line in the current direction, L and R change the drawing direction to the left or right and [and] memorize and restore the current position and

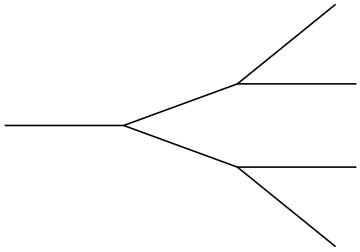

Figure 3.3: Illustration of plant-like structures which can be developed using L-Systems.

drawing direction. Figure 3.3 illustrates the structure which is obtained after three applications of this rewriting rules.

Variants of L-Systems include parametric L-Systems where parameters can be associated with symbols and used in the rewriting rules. L-Systems are context-free when the rewriting rules depend on a single symbol. When they depend on several symbols the L-System is context-sensitive.

3.3.3 Other models of development

Turing proposed a model based on reaction-diffusion equations that generate patterns similar to some that are observed in nature. This model is known as Turing's morphogenesis [177]. For instance reaction-diffusion equations can generate patterns similar to those observed on animal coat [124]. Since Turing's morphogenesis, more complex systems of differential equations have been proposed to model the process of gene regulation [17]. Until now reaction-diffusion models have not been used in evolvable hardware and for this reason this approach is not described in more details here.

Real interactions among genes are more complex than the idealization of RBNs or L-Systems or simple reaction-diffusion equations. For this reason a number of researchers focused on more realistic models. For instance Kitano et al. developed a virtual biology laboratory that consists of a detailed computer simulation model of major biological systems [86]. The recent models of gene regulation have been reviewed by Reil [132].

3.4 Developmental systems in evolvable hardware

Developmental systems implemented in hardware often mimic the multi-cellular nature of biological organisms, resulting in multi-cellular circuits that differentiate under the control of the developmental system.

The fitness of the circuit is evaluated after its development from the genetic string. When the developmental process is deterministic a single circuit evaluation is required to determine its fitness. If development is not deterministic, for instance because the environment can influence development, the circuit needs to be developed and evaluated several times, possibly in different environmental conditions, in order to have an accurate measure of the fitness.

Developmental systems used for evolvable hardware can be loosely grouped in four categories: those using gene regulatory networks to model circuit growth, those using more general cell programs, those using L-Systems, and eventually those using abstract representations of circuits. In an additional fifth category we describe a particular developmental system that uses a direct genotype to phenotype mapping and provides multicellular "Embryonics" circuits with fault-tolerance.

3.4.1 Gene regulatory networks

Gordon et al. explored a model of biological development for the evolution of electronic circuits, which is akin to a minimalistic gene regulatory network with binary protein concentrations in a locally interconnected 2D array of cells [53]. Cells are composed of 4 inputs, a functional part implemented by a 4-input look-up table (LUT) and one output. Development is based on rules that have preconditions and postconditions. Preconditions indicate which proteins must be present or absent in the cell for the postcondition to occur. Postconditions can either generate proteins or change the functionality of cells. The functionality of a cell is modified by changing its input and output connectivity, or by modifying its LUT. Since proteins have binary concentrations, protein diffusion and decay is not modeled in this system. Development rules are executed in software but the functional part of the cells is implemented in a Xilinx Virtex FPGA. Unconstrained evolution was performed to evolve a two bit adder with carry. While the evolvability of the system was worse than a direct encoding (lower maximum fitness at the end of the runs), more regular and repeatable structures appeared in the content of the look-up tables. The authors argued that this is a key point of developmental systems and they suggested that evolving larger adders would become easier since they can be implemented with very regular structures such as the ripple carry adder. The authors afterwards reduced the size of the search space to try to improve the evolvability by reducing the number of rules used to modify the content of the LUTs. Although the results were less good in terms of fitness than with the previous encoding, the authors noticed that the content of the evolved LUTs corresponded to simpler circuits that translated into more gate-efficient solutions when implemented using discrete components. Eventually the model was enriched by introducing diffusion and a finer detection of protein concentrations in neighboring cells. The 2-bit adder could be evolved with this new developmental model when the genetic algorithm was replaced by a hill-climbing algorithm [52].

Gene regulatory networks that have continuous protein concentrations may allow for more complex dynamics in the developmental process. This approach was followed by Koopman et al. that developed a simplified gene regulatory system with continuous protein concentrations that is suited for hardware implementation [88]. Each cell contains a gene regulatory network that interprets an artificial genome containing the rules of activation or repression of genes, in function of the existing proteins in the cell. The model includes protein decay (modeling the half-life of proteins) and protein diffusion across neighboring cells. The authors used fixed-point bit-serial arithmetic and encoded the parameters with a minimal number of bits to achieve compact hardware implementation. The system was partially implemented in the POEtic chip with 200 molecules per cells

(see chapter 4 for a description of this chip). The system was used to evolve circular multi-cellular structures of predefined sizes. In particular the authors showed that the dynamics of the developmental system provided tolerance to diverse kind of faults to the multi-cellular structures.

Simulations of gene regulatory networks may also be used to understand the complex regulatory pathways in organisms, for example to design drugs. Tagkopoulos et al. exploited the analogies between CMOS circuits and GRNs to implement efficiently these networks in a custom analog integrated circuit [156]. While the authors focused on biological applications, the circuit they developed is a reconfigurable device that could be used in evolvable hardware, with the advantages of compactness and high speed of the analog implementation.

3.4.2 Cell programs

Gene regulatory networks act as programs within the cells. However these cell programs need not necessarily mimic genes and proteins; they can be more general programs.

Developmental Cartesian Genetic Programming (DCGP) is a developmental system proposed by Miller where cells implement both a functional part and a developmental part in the form of a cell program [116]. The functional part corresponds e.g. to logic gates. The developmental program of the cell takes as inputs the connectivity, functionality and position of the cell in the circuit, and it controls the new connectivity and functionality of the cell, and whether it duplicates in the next developmental step. Figure 3.4 illustrates the program of a DCGP cell that has two functional inputs (A and B, e.g. a two-input logic gate). Development starts from a single initial cell and all the cells of an organism share the same developmental program. The developmental program is encoded with Cartesian genetic programming (see chapter 2) and is evolved. Miller used DCGP to evolve binary adders and even-parity functions. He showed that development could sometimes provide a moderate degree of generalization: by increasing the number of developmental steps he obtained a logic function of one additional input. He however remarked that the DCGP genotypes were less evolvable than direct encodings, and that improving the scalability of evolvable hardware may be a more complex issue than simply looking for a way of reducing the genotype length.

Miller modified DCGP to use internal and external variables, akin to chemical concentrations, instead of the absolute position of cells in the circuit. The cell program now maps input conditions (chemical concentration and type of the cell and of its immediate neighbors) to output behaviors: production of chemicals, change of the cell type, cell death or growth of a new cell (duplication) [117]. The evolved cell program is also encoded using Cartesian Genetic Programming. Miller used this system to evolve organisms with specific patterns of colored cells (e.g. a French flag). He showed that the system could adapt to the environment: external environmental signals could control the color of cells obtained after development. He also demonstrated that the system was capable of self-repair: after removing cells, the developmental process could recover the initial pattern of differentiated cells. This system was implemented in software, however hardware implementation and its use in applications requiring self-repair and adaptation are under-way

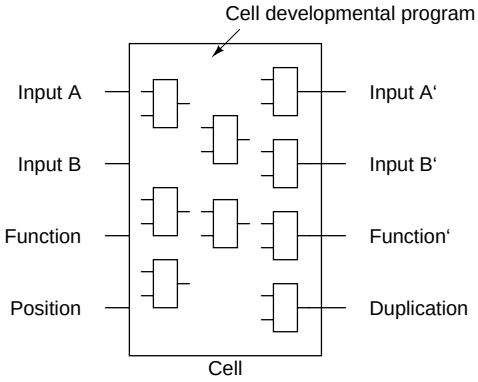

Figure 3.4: Illustration of the cell program in Developmental Cartesian Genetic Programming. Inputs of the developmental program are, on the left, the connectivity of the inputs, the current function of the cell and its position in the circuit. The developmental program outputs, on the right, the new connectivity and function of the cell for the next developmental step, and whether the cell duplicates (i.e. when Duplication is 1 the cell duplicates). Boxes inside the DCGP cell represent the building blocks composing the developmental program which governs the cell behavior.

[101].

De Garis considered the evolution of large-scale neural networks in a cellular-automaton (CA) executed in a custom hardware architecture designed for high-speed CA simulation [24]. The neural connections are grown according to an evolved program that is executed in a fully distributed way by the CA. During development, growth signals are sent along synaptic connections. When they reach the extremity of connections they induce growth, possibly altering the synaptic direction or creating branchings. Whenever a synapse reaches another one or a neuron, a connection is established. The neural architecture is evolved by genetically encoding the sequence of growth signals. De Garis used this system to produce waveforms of arbitrary shapes, to halve the frequency of an input signal, to solve the XOR problem, to detect a moving line and to detect a frequency or signal strength [25].

3.4.3 L-Systems

Since simulating genetic regulatory networks or cell programs may be computationally intensive, or take a lot of space in hardware, more abstract developmental systems such as L-Systems may be used.

Haddow and Tufte explored an indirect genetic encoding based on L-Systems to address the scalability issue of evolvable hardware. This system is used to evolve electronic circuits on a custom “virtual” FPGA [57]. The virtual FPGA has the features deemed necessary for unconstrained evolution [56]. It does not exist as a custom chip, but is implemented over a Xilinx Virtex FPGA. Two types of L-System rules are used. Change rules replace a part of the configuration string with another one of equivalent length. Growth rules are used to allocate new logic elements (called *s-blocks* in the virtual FPGA terminology) on free space around the s-block that triggered the rule. The starting axioms and

development rules are evolved using a standard genetic algorithm. The system was used to evolve circuits with specific distribution of s-blocks on a 16x16 cell array with moderate success [58]. Subsequent work considered the restriction of s-block configuration to specific s-block types to help evolution find specific kinds of circuits, and L-System rules were extended to be contextual and to control cell death [176]. Applications included the growth of structures from a single starting cell (achieving a specific circuit size with a limited number of developmental steps), the differentiation of cells (a number of different cell types had to be present after the developmental process) and the formation of patterns. In the last case, a symmetrical pattern of cells had to be found within a limited number of developmental steps. These circuits however did not implement a real functionality and were of limited size: 3x3 s-blocks for the growth and differentiation task, and 4x4 s-blocks for the pattern formation. The developmental mechanism is implemented in hardware on a dedicated processor [175] and the genetic algorithm is executed in software on a standard desktop computer.

3.4.4 Abstract representations

Even simpler models of development can be used, that do not necessarily use a multicellular approach to development.

Circuits represented in a Hardware Description Language (HDL) can be evolved by a developmental process consisting of rewriting rules following a HDL grammar, evolved by "production" genetic algorithms [119]. Rewriting is controlled by a tree-structured chromosome where each node contains a production rule that is recursively applied to a starting symbol. In addition to selection, mutation and crossover, specific operators were designed for evolution: duplication copies a functional block within an individual, insertion copies a functional block from another individual, and deletion removes functional blocks. Since HDLs can represent high level building blocks (registers, arithmetic operators), this genetic encoding is usually more compact than a direct representation of all the logic gates of a circuit. Evolved HDL circuits are simulated to evaluate their fitness. This approach was used to generate controllers for artificial ants that had to follow a possibly interrupted food trail. The authors suggested that this approach might exploit regularities in the phenotype and therefore that it might be scalable to more complex problems [63].

Koza et al. showed that genetic programming could be used to evolve electronic circuits [89]. The process by which the tree-representation of the circuit that is used in genetic programming is decoded into a circuit may be seen as a simple type of developmental system. Koza et al. showed that genetic programming could be enhanced with automatically defined functions to provide modularity and reuse of structures [90, 91]. This method was successfully applied to design low-pass filters, two-band crossover filters, amplifiers, etc. [92].

Lohn et al. explored a linear representation of analog circuits that is decoded with a "circuit construction robot" [102, 103]. The genetic string consists of a sequence of bytecodes that are executed sequentially. Each bytecode indicates which new element to add to the circuit (e.g. resistor, capacitor, transistor), its value, and how to interconnect it. The authors noted that this encoding generated many topologies which are seen

in hand-designed circuits, even though not all the topologies could be represented. The authors successfully evolved analog filters and transistor-based amplifiers using this representation. This approach does use a direct genetic encoding (the number of bytecodes is equal to the number of elements in the circuit), but since it relies on an encoded growth program, it may be considered as a relatively simple developmental system.

Mattiussi et al. proposed a genetic encoding that can be decoded even after major reorganization by genetic operators, and at the same time that allows for gradual changes of the phenotype under certain genetic operators [111]. The genetic encoding consists of a string out of which the components, component values (e.g. capacitor values) and terminal interconnections are decoded by a process of string matching. The decoding process looks for predefined substrings that identify components. Once a substring is found, the genetic string is scanned from this point onwards to extract the substring corresponding to the component value (if any) and substrings that identify the terminals of the component (terminal labels). Those substrings are delimited by predefined tags. If the necessary information (component value and terminal labels) are not found before encountering a substring representing a new component, then the component is not instantiated. Interconnections among component terminals are implemented by resistors whose values are inversely proportional to the degree of similarity among the strings identifying the terminals. The authors used this system to evolve analog circuits such as voltage references, but they noted that this genetic encoding is also suited for the evolution of other "analog networks", such as neural networks or gene regulatory networks.

3.4.5 Embryonics

Embryonics (Embryological Electronics) is a custom reconfigurable device that provides self-repair and self-replication capabilities to multi-cellular electronic circuits [108, 109, 110]. This is achieved with a developmental system that, contrarily to most of those shown earlier, uses a direct genotype to phenotype mapping.

Circuits (or organisms) are implemented in Embryonics in the form of a rectangular array of cells. These cells are structurally identical and execute the same program, which is considered as the genetic code of the organism. The cell program is hand-coded. This program is however parameterized: depending on the coordinates of the cell in the organism different execution paths are followed. The developmental process of Embryonics organisms is thus a coordinate-based differentiation. Initially all the cells are undifferentiated. Logic within the cells establishes a X;Y coordinate system. As the program in the cells executes, the cells take the functionality corresponding to their coordinates. As the cells all contain the complete genetic description of the organism (the cell program), mechanisms such as self-reproduction and self-repair are possible.

Self-repair is implemented by detecting faults in cells by functional redundency and deactivating the faulty cells. At this stage the developmental process is restarted, but the coordinate system is modified so that faulty cells are not numbered (i.e. faulty cells are not included in the coordinate system). As a consequence, faulty cells are not used, and spare cells are used instead to implement the circuit. In case of faults, Embryonics development always leads to circuits functionning exactly like the orgininal circuits, as long as spare

cells are available in the circuit.

3.5 Other applications of developmental systems

Developmental systems are also used to evolve neural networks. Kitano proposed a graph generation system inspired upon L-Systems to generate the connection matrix of neural networks [84]. L-Systems were also used by Vaario [179], and by Boers and Kuiper to develop modular neural networks [10]. Gruau developed a language controlling the growth of neural networks called cellular encoding that allows to evolve modular neural networks [54]. He showed that cellular encoding could automatically find small neural architectures that solve the pole-balancing problem, whereas with a direct encoding several architecture had to be tried manually before finding a proper one [55]. Luke and Spector proposed an alternative to Gruau's cellular encoding which they call edge encoding [104]. Eggenberger proposed a biologically inspired model of development that uses a gene regulatory network to create neural networks in 3D space [31, 32]. Jakobi presented another biologically inspired genetic encoding for the evolution of neural controllers for robots [76]. Nolfi et al. evolved neural networks whose connections resulted from an axonal growth process [125]. Boshy and Ruppin developed a self-organized compact encoding that they used to find compact neural controllers without any selective pressure towards small size [13]. Astor and Adami used an artificial chemistry to grow neural networks. Genomes were hand designed to exhibit some biological behaviors [5]. Federici proposed a developmental model to evolve neural controllers for robotic applications where each cell executes a program encoded in the form of a neural network that controls the growth and differentiation of cells [35]. Quick did not evolve neural networks but exploited the dynamic nature of GRNs to control robot behaviors by coupling sensory information and motor commands to proteins within cells [129].

Developmental systems can also be used to evolve structures or robot morphologies. Sims evolved the morphology and controller of 3D creatures in a simulated physical world. A directed graph represents how to build the morphology and control system: nodes correspond to rigid parts of the creature and they encompass functional blocks that control the forces applied on the joints based on sensory readings [145]. Dellaert and Beer proposed a biologically defensible model of development based on random Boolean networks, and they used it to co-evolve the body and control system of autonomous agents [27, 28]. Agarwal developed a cell programming language that mimics the life of biological cells [1]. Eggenberger used differential gene expression to grow the morphology of 3D organisms [33]. This work was carried on by Bongard et al. [11, 12]. Hornby and Pollack used L-Systems to generate moving virtual creatures [71]. Symbols of the L-System correspond to construction commands and the controller consists of oscillators placed at the joints of the creature. They used the same developmental system to evolve structures (e.g. tables) and they obtained higher fitness and faster than with a direct genetic encoding [72]. Kumar and Bentley developed an evolutionary developmental system based on gene regulatory networks that they used to evolve simple spherical 3D shapes [93]. They also compared different type of developmental systems to evolve patterns and showed that

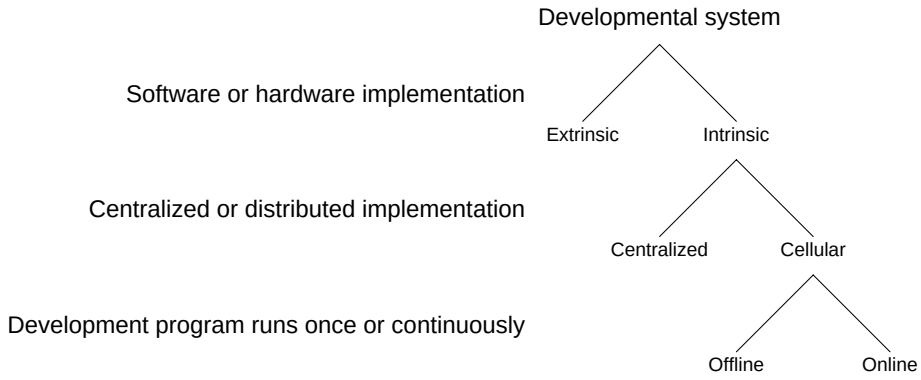

Figure 3.5: Classification of developmental systems used in evolvable hardware. We distinguish between developmental systems executed in software or hardware. For those executed in hardware, their implementation can be either centralized (e.g. in a dedicated coprocessor) or distributed or cellular. In the last case environmental or cellular interactions may continuously influence the developmental process, in which case development is online. On the other hand development may be executed only once to obtain the phenotype and development is offline.

they could provide significant advantages compared to direct genetic encodings [9].

Developmental systems are further reviewed in [4, 87, 148].

3.6 Classification of developmental systems

To classify developmental systems that are employed in evolvable hardware, we consider key characteristics of their hardware implementation (figure 3.5).

Inspired from the difference between intrinsic and extrinsic evolution in evolvable hardware [24], that distinguishes the physical implementation of an evolved circuit from its simulation, we want to distinguish similarly between the execution of the developmental system in software or in hardware. By *extrinsic developmental system* we mean that the developmental mechanism is executed in software on a desktop computer. The resulting circuit is then implemented physically or simulated. By *intrinsic developmental system* we mean that the developmental system is implemented in the same hardware as the circuit that is evolved (e.g. the same chip).

In the case of an intrinsic developmental system we wish to distinguish between a *centralized implementation* or a *distributed or cellular implementation*. In a centralized implementation a single hardware unit is in charge of running the developmental mechanism in the same hardware as the evolved circuit. This can be a CPU or a dedicated coprocessor located on the same device as the circuit that is grown. In a distributed approach the developmental system is executed by many independent but communicating units. Each of these units may be seen like a cell, that implements the developmental process in addition to its normal functionality. Cellular implementations may be faster, more scalable, more biologically plausible, and possibly more robust than centralized implementations, at the expense of more space.

Finally in *online development* the developmental process is running continuously to decode the genotype into the phenotype. Development may thus react to inter-cellular or environmental signals while the circuit operates. This may allow for characteristics that are seen in living organisms such as self-repair or adaptation. In *offline development* the developmental process decodes the genotype into the phenotype in one step. Once the phenotype is obtained the developmental process stops. Although this is biologically less plausible, this may save hardware resources when implementing the developmental system.

In this review most developmental models are extrinsic [52, 63, 89, 92, 88, 111, 116, 117] and few are intrinsic [25, 109, 156, 175].

Among the intrinsic developmental models, one is modeling biological gene regulatory networks in hardware but does not have the objective to develop circuits [156]. Embryonics is also an intrinsic developmental system, however circuits are hand-coded and not evolved [109]. To the author's knowledge, at the time of writing the only remaining intrinsic evolutionary developmental systems are the CA-based growth of neural networks of De Garis et al. [25] and the L-System-based encoding of Haddow and Tufte [175]. The former uses a cellular implementation of the developmental system, while the latter uses a centralized implementation.

All the developmental systems in this review operate offline, with the exception of Miller's cellular program which showed that online development could lead to fault tolerant and adaptive development [117], and Embryonics development that allows self-repairing electronic circuits [109].

3.7 Summary

In this chapter we reviewed developmental systems used in evolvable hardware. These systems provide an indirect genotype to phenotype mapping that may improve the scalability of evolvable hardware to larger circuits. The dynamics of the developmental process may also provide adaptivity and fault-tolerance that is often seen in biological organism [109, 117].

There seem to be two approaches to developmental systems. On the one hand supporters of biologically plausible models of development claim that scalability and evolvability can only be improved in this way [93]. However mimicking biology tends to lead to complex developmental systems [31, 32, 93]. Biological realism is further limited by our still partial understanding of biological development [187] and the trade-offs between biological plausibility and implementation constraints (e.g. limited silicon resources) and evolutionary needs (e.g. fast evaluation of many candidate solutions). Furthermore, this review shows that circuits evolved with the more biologically plausible developmental systems (e.g. gene regulatory networks, cell programs or L-Systems) are still rather simple (e.g. simple arithmetic functions or predefined non-functional structures of cells). On the other hand there are developmental systems that are abstract decoding of the genotype. These may be inspired by biology, but do not seek biological plausibility. This second approach is a more pragmatic view of development [158], yet it seems to work

for several problems. For instance genetic programming allows to evolve complex analog circuits with modularity and reuse of structures [90, 91], even though it is far from being biologically plausible.

We classified developmental systems according to characteristics of their hardware implementations. We have seen that there are few intrinsic, online and cellular developmental systems, even though this is where we believe most of the benefits of developmental systems lie. Intrinsic development means fast genotype to phenotype mapping and close interaction of the developing circuit with its environment. Together with online development this may allow adaptation to the environment and fault-tolerance. Finally cellular implementations may be more robust than centralized ones and are more scalable.

In summary we draw two points from this review, that we will consider in the developmental system for multi-cellular POEtic circuits. The first one is that we should consider intrinsic, cellular and online developmental systems. The second it that we should not overly focus on biologically plausible developmental systems, since these may lead to complex systems that take a lot of space in hardware and furthermore they do not yet seem to allow the evolution of more complex circuits than simpler developmental systems. This does not preclude biology as a source of inspiration, but we will consider simpler models of development that may better accomodate implementation constraints such as silicon area and computational time.

4 Multi-cellular architecture for bio-inspired hardware

Abstract

In the previous chapters we described how evolution may be used to create electronic circuits, and how a genotype to phenotype mapping based on a developmental system may provide better scalability to evolvable hardware, or lead to adaptive or fault-tolerant circuits. In this chapter we describe a multi-cellular architecture that allows a flexible integration of evolution, development and, as will be shown in later chapters, learning mechanisms. We explain how this architecture is translated in hardware to give POEtic circuits capable of evolution, development and learning. Finally we describe a reconfigurable device called the POEtic chip, that is ideally suited to implement this architecture in hardware. The mechanisms of evolution, development or learning are not considered here but in subsequent chapters.

4.1 Introduction

In introduction we took the stance that bio-inspired hardware should encompass evolution, development and learning to fully benefit from bio-inspiration. Circuits capable of evolution, development and learning are referred to as *POEtic circuits*. POE stands for Phylogeny, Ontogeny and Epigenesis, which are respectively evolution, development and learning [178].

We argued that to fully benefit from the potential of these circuits a novel evolutionary system that combines a genetic encoding and a developmental system needs to be designed. Such an evolutionary system may allow to capture the complex mechanisms of development mediated by gene regulation that are seen in biological organisms, and this may improve the scalability of evolvable hardware, or lead to adaptive development or fault-tolerant circuits.

In order to design this evolutionary system, an architecture which allows the integration of mechanisms of evolution, development and also learning must be defined.

Therefore, the first objective of this chapter is to show an architecture that allows a flexible combination of evolution, development, and also learning. The architecture does

not specify which are the evolutionary, developmental and learning mechanisms (this is done in later chapters), but it specifies the overall structure of the POEtic circuits and how these mechanisms interact with each other.

This architecture describes only the "concept" of a circuit. It needs to be physically implemented to translate in a POEtic circuit. The second objective of this chapter is to explain how this architecture can be implemented in hardware. Since POEtic circuits may be used in various applications (in this thesis we consider different applications in mobile robotics, but also pattern recognition), the architecture is best implemented in a reconfigurable device (i.e. a chip that can be programmed or configured to implement electronic circuits). This chapter thus explains how reconfigurable devices can be programmed with this architecture.

Finally this chapter describes one particular reconfigurable device which is ideally suited to implement POEtic circuits because it has specific features to support the implementation of bio-inspired mechanisms in hardware. This device is called the *POEtic chip*.

This chapter is organized as follows. In section 4.2 we describe the architecture that allows to integrate evolution, development and learning mechanisms. In section 4.3 we describe how this architecture is translated in hardware. In section 4.4 we describe the reconfigurable POEtic chip than can be configured to implement POEtic circuits following the architecture introduced earlier. Section 4.5 concludes this chapter.

4.2 Multi-cellular architecture

Combining evolution, development and learning in electronic circuits requires an architecture that can accommodate these mechanisms, yet provides flexibility to allow different mechanisms to be used depending on the applications of the circuits (e.g. different learning mechanisms may be envisaged depending on the application).

In this thesis we want to evolve electronic circuits with an evolutionary system that combines a genetic encoding and a developmental system in order to allow complex indirect genotype to phenotype mappings. The developmental system that we consider in this thesis (described later in chapter 6) is inspired by the mechanisms of growth and differentiation of multi-cellular biological organisms.

For this reason, to accommodate development, the architecture that we consider to integrate evolution, development and learning is a *multi-cellular* architecture composed of a regular 2D array of *cells*.[1]

In order for a cell to perform a function in the circuit, it must contain a *functional* part. The function can be a simple logic gate or a more complex function like a neuron.

The function that a cell takes in the circuit depends of the genetic code of the circuit, and of the genotype to phenotype mapping mechanism (i.e. the developmental system)

[1] In principle POEtic circuits could exploit other cell topologies (e.g. irregular structures of cells). However, irregular cell structures may require more complex evolution, development and learning mechanisms that take more space in hardware. We assume regular 2D structures of cells in order to minimize the hardware resources required for these mechanisms.

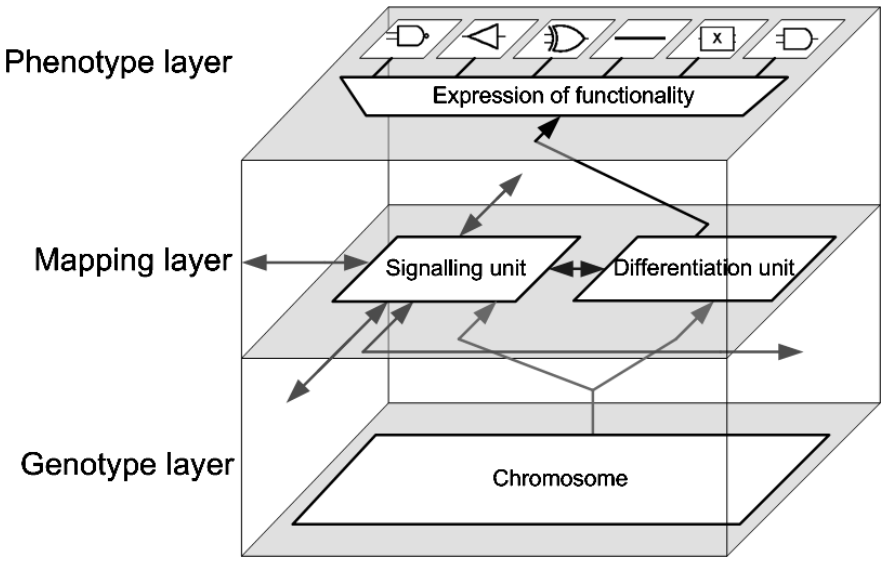

Figure 4.1: Three-layered architecture of a cell. The genotype layer is a memory storing the genetic code of the entire circuit. The mapping layer reads and decodes the relevant part of the genetic string, for instance using inter-cellular signaling and differentiation rules. The mapping layer then indicates the functionality that must be expressed in the phenotype layer. The phenotype layer can implement any of the functionalities that may be present in the circuit. Inter-cellular communication occurs on the phenotype level (e.g. the interconnections of a neural network) but it can also occur on the mapping layer, for instance to exchange signals controlling the differentiation of cells.

that controls the differentiation of cells. The cell must therefore contain the *genetic code* of the circuit. The genetic code may also be transferred from a cell to another one during the operation of the circuit to allow the growth of the multi-cellular circuit from a single initial cell.

Finally, the *genotype to phenotype mapping* mechanism, or developmental system, that interprets the genetic code of the circuit and provides the functionality of cells must also be included in the cell.

These three parts of the cell may be represented by three different layers within the cell: a genotype, a mapping, and a phenotype layer (figure 4.1) [178]. The genotype layer is a memory that contains the genetic code of the entire circuit. The mapping layer implements the developmental system. It maps the genotype into the phenotype by interpreting the relevant parts of the genetic string, and it indicates the functionality that the phenotype layer must take. For this purpose the mapping layer can access the genetic string in the genotype layer. The phenotype layer implements the functional part of the circuit, for instance a logic gate or a neuron, according to the result of the development mechanism.

From the communication point of view there is bidirectional communication between cells, and inter-cellular communication is possible in all three layers. The phenotype layer may use it to exchange functional signals between neighboring cells (e.g. between neurons), and the genotype to phenotype mapping may depend on signals exchanged on the mapping layer. Cells also have input or output connections to sensors or actuators.

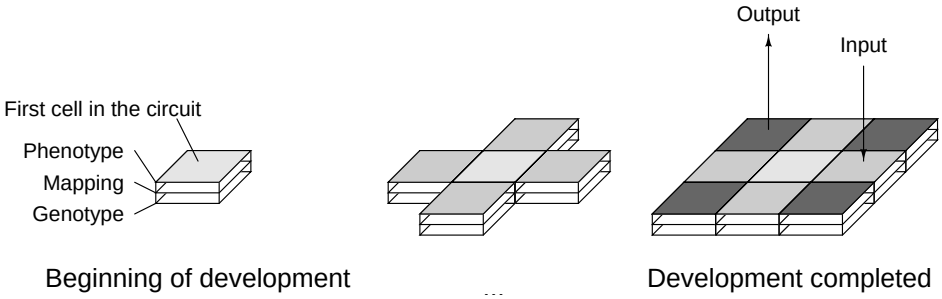

Figure 4.2: Development of a multi-cellular circuit from an initial cell. Each square represents a cell. The grayscale in the top layer of the cell represents the functionality of the cell in the circuit. Cells can have inputs and outputs to exchange signals with the environment (e.g. sensors or actuators).

Thereby the multi-cellular circuit can be seen like an organic tissue such as skin or muscles, where each cell is capable of sensing its environment and acting on it.

The mechanisms of evolution, development and learning each operate on one of the layers. Evolution manipulates the genetic material of the circuit in the genotype layer. The growth and differentiation of the cells is controlled by the developmental system in the mapping layer. Learning mechanisms are implemented in the phenotype layer of the cells. This architecture is flexible: it allows mechanisms in the different layers to be selected according to the application.

Conceptually, this architecture allows the growth and differentiation of multi-cellular electronic circuits starting from an initial cell (figure 4.2).

This architecture describes only the “concept” of circuits capable of evolution, development and learning. This architecture must be physically implemented in hardware to obtain a POEtic circuit. This aspect is described in the next section.

4.3 Translating this architecture in a circuit

The architecture that we described in the previous section needs to be physically implemented in hardware in order to obtain POEtic circuits.

Depending on the applications, different mechanisms of evolution, development or learning may be used. Implementing the architecture shown above directly in silicon does not allow to change these mechanisms depending on the application.

To achieve a higher degree of flexibility, the architecture is therefore best implemented in a reconfigurable device (i.e. a chip that can be programmed to implement circuits) such as a FPGA (field-programmable gate array) or the POEtic chip, which is described in section 4.4.

Reconfigurable devices are composed of reconfigurable logic elements whose functionalities and interconnections can be *configured* or programmed to take a desired functionality (e.g. a logic element can be configured as a logic gate or as a flip-flop).

Implementing the architecture described in section 4.2 in such a reconfigurable device means that cells, together with their phenotype, mapping and genotype layers, are imple-

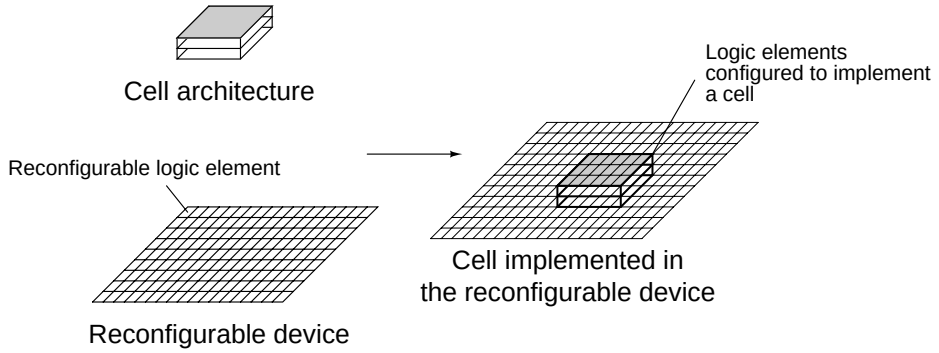

Figure 4.3: The architecture of the cell, with its phenotype, mapping and genotype layer (top left), is implemented in hardware in a reconfigurable device (bottom left). The reconfigurable device consists of an array of logic elements that can be programmed to implement a desired functionality, such as logic gates or flip-flops. The cell with its phenotype, mapping and genotype layers is therefore implemented in the reconfigurable device by configuring several logic elements (right).

mented by configuring several logic elements of the reconfigurable device appropriately. Figure 4.3 illustrates this process.[2]

Conceptually we intend to have a multi-cellular circuit that grows and develops according to the genetic string of the circuit and to the developmental system. Hardware however does not allow new cells to be physically created (i.e. no new silicon can be created to implement cells). Therefore, instead of "growing" new cells, the reconfigurable device is initially configured with undifferentiated cells. The developmental system then controls the differentiation of those pre-existing cells according to the genetic string and to the development rules.

In summary, the reconfigurable device is first programmed with an array of undifferentiated cells. Afterwards development starts, and the cells of the multi-cellular circuit differentiate according to the developmental system. Once development is completed, the circuit can be interfaced to sensors or actuators with the input and output of the cells (figure 4.4).

The developmental system controls the differentiation of the cells. This implies that cells must be able to take the functionalities required by the development process. To simplify the hardware implementation, we assume that the functionalities which can be expressed by the development process are predefined by the user before evolution. These predefined functionalities can be neurons (e.g. to evolve circuits that implement neural networks), logic gates, signal processing elements, etc.

The functional part of the cells (i.e. the phenotype layer) is designed so that the cell can implement any of these predefined functionalities. In other words cells are *totipotent*: an undifferentiated cell can take any of the functionalities which may occur in the circuit.

[2]The three layers of a cell do not exist as a 3D structure in the reconfigurable device. This three-layered representation is only used for visualization purposes. The cell which is implemented in the reconfigurable device consists of a planar array of reconfigurable logic elements. In practice the three-layers are simply adjacent blocks of reconfigurable logic elements.

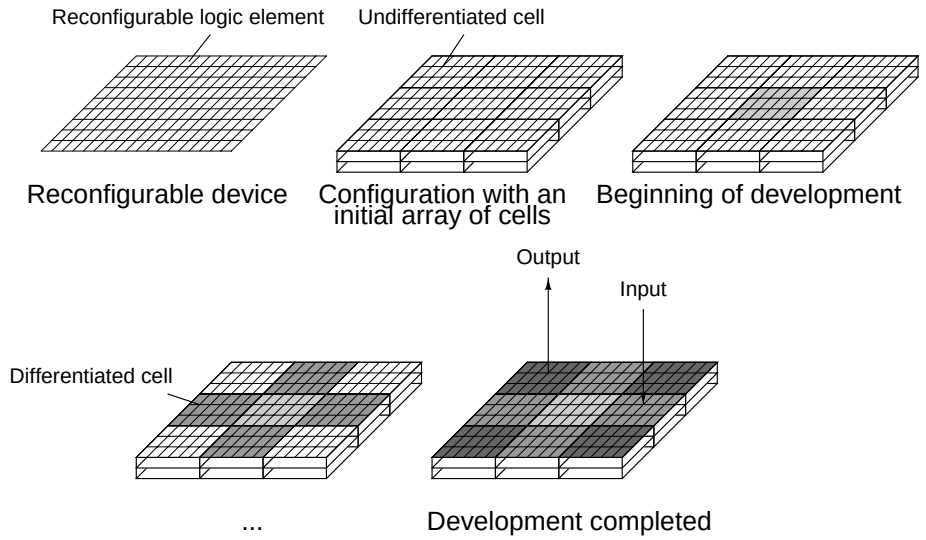

Figure 4.4: The development of the multi-cellular circuit starts by programming the reconfigurable device with an array of initially undifferentiated cells. Afterwards the developmental system, according to the genetic string of the circuit and to the inter-cellular or environmental signals, controls the differentiation of the cells at run-time (gray cells indicate cells that underwent differentiation). Once the circuit is completely developed, inputs can be applied to cells and outputs can be read.

The hardware implementation of totipotent cells can be done in two ways. A mechanism of self-reconfiguration[3] can be employed, by which the mapping layer of the cell reconfigures the logic elements that form the functional part of the cell to implement the required functionality. Alternatively the cells can be designed so that they contain all the predefined functionalities that can appear in the circuit, and the functionality that the cell effectively implements is selected at run-time according to the development process.

Until now we considered the architecture of the cells, but not the mechanism by which the reconfigurable device is initially programmed with cells, neither the mechanism of evolution (the genotype layer is only a memory holding the genetic code of the circuit but does not include any evolutionary mechanism). The initial configuration of the reconfigurable device and the evolutionary mechanism are done by a processor outside of the multi-cellular array. This processor may however be physically present in the same chip as the reconfigurable logic elements.

Some reconfigurable devices, and in particular the POEtic chip described in section 4.4, contain in a single chip both reconfigurable logic elements and a processor. These devices are thus ideally suited to implement the mechanisms described here since both the multi-cellular circuit and the processor taking care of running the evolutionary mechanism and programming the reconfigurable logic are integrated in a single chip.

The complete implementation of POEtic circuits in hardware thus consists of two parts. One part is the multi-cellular circuit itself, implemented in reconfigurable logic,

[3]Self-reconfiguration is the capacity of one logic element in a reconfigurable device to change the functionality of another logic element, therefore altering its functionality.

and the other part is a processor that executes the evolutionary mechanism and programs the reconfigurable logic with the array of cells (figure 4.5).

We refer to these two parts respectively as the *organic subsystem* and the *environment subsystem*. The organic subsystem is the part of the reconfigurable device where multi-cellular circuits are implemented (i.e. the organic subsystem consists of reconfigurable logic elements). These multi-cellular circuits develop like "organisms", hence the name of the subsystem.

The environment subsystem is the part of the reconfigurable device that executes the evolutionary mechanism and that takes care of programming the reconfigurable logic with the array of undifferentiated cells. In addition the environment subsystem can be used to interface the multi-cellular circuits with their environment (hence the name of the subsystem). In other words, input and output signals of the multi-cellular circuit can go through the environment subsystem, which may for instance preprocess them, before sending them to sensors or motors. In addition the environment subsystem also measures the fitness of the circuit.

In summary, in order to use the multi-cellular circuit the following sequence of operations is followed:

1. Initialization of the organic subsystem with the cells, complete with the genetic code of the entire circuit. These cells are undifferentiated yet.

2. The multi-cellular circuit develops according to the genetic string of the circuit and the developmental system implemented in the mapping layer of the cells, and the cells differentiate accordingly.

3. Once development is complete, the circuit is ready for operation. The processor can apply inputs to the circuit, or read outputs. The fitness of the circuit is measured by the processor that monitors the behavior of the circuit.

4. When a new circuit is required (e.g. when the fitness of another genetic string needs to be measured) the process restarts at the first point.

In the next section we describe a particular reconfigurable device known as the POEtic chip. This POEtic chip is ideally suited to implement POEtic circuits because it contains an environment subsystem and an organic subsystem such as the one described here.

4.4 The POEtic chip

The POEtic chip is a custom reconfigurable device similar to a FPGA (field-programmable gate array), but it is developed specifically to implement bio-inspired systems [122, 160, 161, 163].

In particular the POEtic chip is ideally suited to implement POEtic circuits since it is composed of an organic subsystem (i.e. reconfigurable logic) which can be used to implement multi-cellular circuits, and an environment subsystem which can be used to configure the organic subsystem, run the evolutionary algorithms, measure the fitness, and

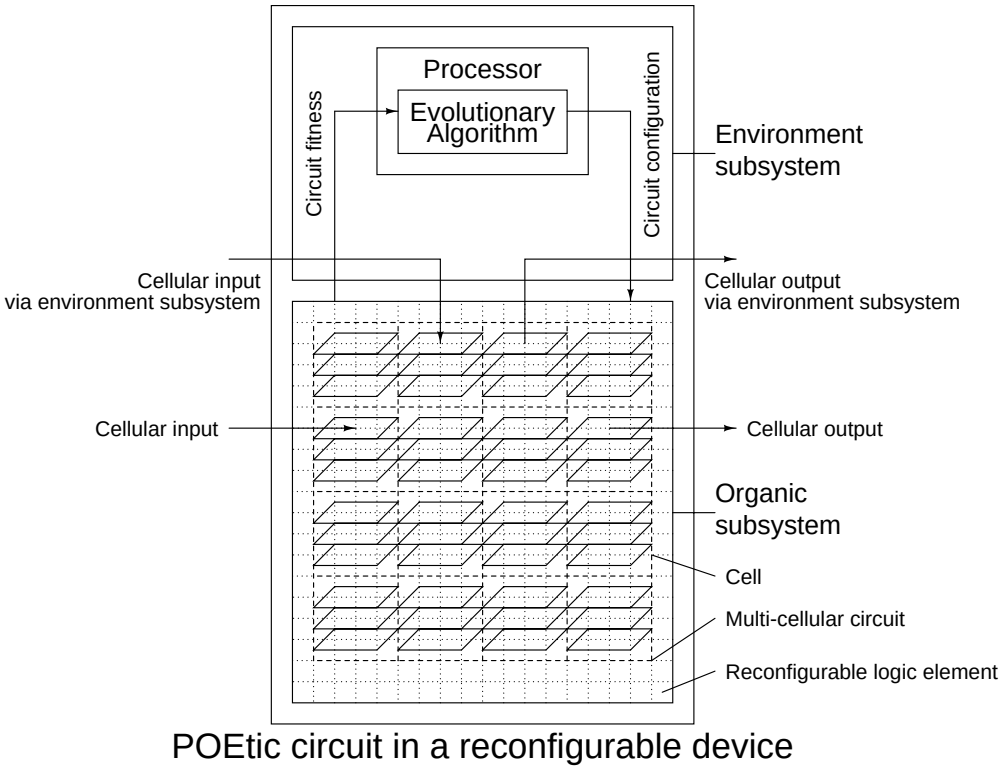

Figure 4.5: The reconfigurable device which implements POEtic circuits is partitioned in two subsystems: the organic and the environment subsystem. The organic subsystem consists of reconfigurable logic elements (bottom) which are used to implement the cells which make up the multi-cellular circuit capable of evolution, development and learning. The environment subsystem (top) executes the evolutionary algorithm, configures the organic subsystem, and measures the fitness of the circuit, typically by monitoring the inputs and outputs of the circuit. Interconnections of the multi-cellular circuit with the outside world can occur directly at the level of the organic subsystem, or inputs and outputs can pass through the environment subsystem that may preprocess them or use them to measure the fitness of the circuit.

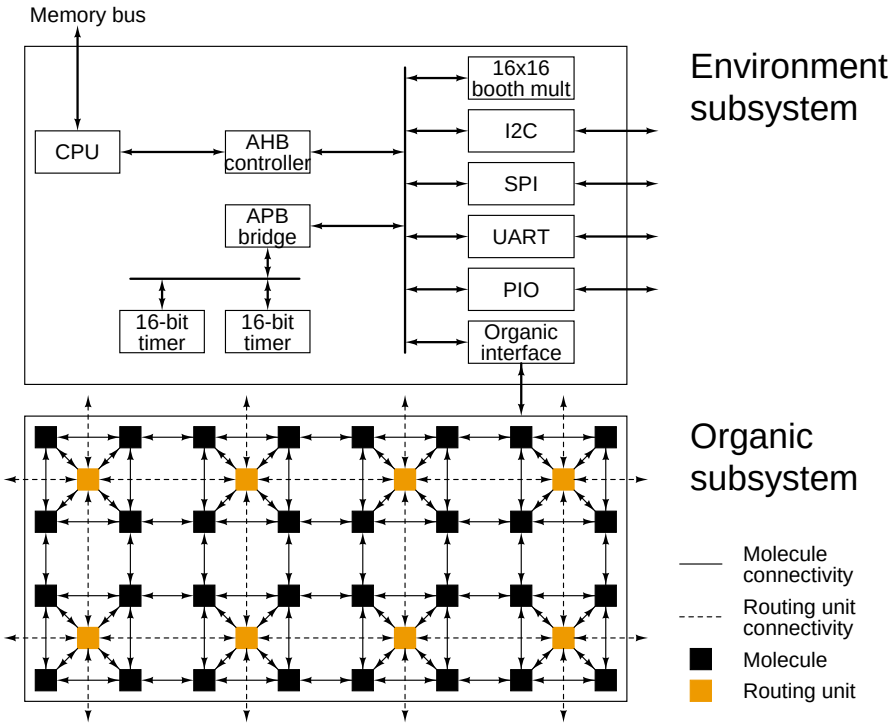

Figure 4.6: The POEtic chip is composed of two subsystems. The environment subsystem is composed of a processor (CPU) that is interfaced with several peripherals (e.g. for communication), external memories to store data, and the organic subsystem. The organic subsystem is composed of locally interconnected reconfigurable logic elements called molecules, and of routing units that are used for long-distance connections with dynamic routing. AHB and APB refer to bus interfaces that are used to connect the CPU to its peripherals.

interface the multi-cellular circuit with its environment. The POEtic chip is configured to implement POEtic circuits as explained in the previous section (figure 4.5).

The architecture of the POEtic chip is illustrated in figure 4.6 and its main features are summarized below. More informations are provided in appendix A.

Environment and organic subsystem: The POEtic chip contains an organic subsystem and an environment subsystem. The organic subsystem is composed of reconfigurable logic and is used to implement the cells of multi-cellular circuits. The environment subsystem contains a processor or Central Processing Unit (the POEtic CPU) and communication peripherals. The processor can be used to implement evolutionary algorithms, measure the fitness of the circuit, communicate with external devices (e.g. sensors, actuators) and configure the organic subsystem.

Fast access to the configuration bits: Evolvable hardware requires to test a lot of circuit configurations before one which has the desired fitness is obtained. To minimize the reconfiguration time and maximize the speed of evolution the processor has direct and fast access to the configuration bits of the reconfigurable logic in the organic subsystem. In comparison commercial FPGAs often require an external interface which slows down the configuration.

Documented configuration string: Since the POEtic chip is a custom chip, the meaning of all of its configuration bits is known. This may be used to perform unconstrained evolution in the reconfigurable logic and analyze the evolved circuits from their configuration string, or to develop design tools suited for bio-inspired applications. FPGA manufacturers nowadays do not disclose this information.

Hardware self-reconfiguration: Hardware self-reconfiguration is the reconfiguration of one logic element by another one. Self-reconfiguration may allow adaptive hardware (e.g. logic elements can be reprogrammed to act differently depending on environmental stimuli), and self-repairing or self-reproducing hardware (e.g. logic elements can reconfigure spare logic elements with a copy of their configuration bits to recover a functionality or instantiate a new functional element). Self-reconfiguration in commercial FPGAs often involves the reprogramming of an entire area of the chip. With the POEtic chip the exact desired elements can be reconfigured.

Optimized CPU instruction set: Evolutionary algorithms are stochastic algorithms and therefore they make extensive use of pseudo-random numbers. They also manipulate the genetic code of the circuit (a bit string) at the bit level. The POEtic processor has hardware instructions to perform that kind of operations that may be used to speed up the evolutionary process.

Dynamic routing: In conventional FPGAs physical connections between logic elements must be planned at design-time and compilation tools take care of placing and routing the components on the FPGA using the available resources. Afterwards, there is no possibility to change the placement of components or their routing without a new compilation. This approach is therefore relatively static. In the POEtic chip *dynamic routing* is introduced in the reconfigurable logic. Dynamic routing builds connections between logic elements automatically and at run-time in hardware in a distributed way. This may be used for dynamic reorganization within the chip, for instance in case of self-repair, self-reproduction, or to react to environmental changes (e.g. changes in the locations of inputs or outputs).

Expandable: Several POEtic chips can be connected together pin-to-pin. These interconnected chips act like a single expanded chip with a larger organic subsystem (i.e. more reconfigurable logic elements). This allows to implement circuits that do not fit in a single chip. When several POEtic chips are interconnected, only a single CPU in one of the environment subsystem is active. All the other CPUs are virtually disconnected from the expanded chip. The active CPU can however access the communication peripherals in all of the POEtic chips [122].

4.4.1 Cells in the POEtic chip

Cells of multi-cellular circuits are implemented in the organic subsystem of the POEtic chip. The organic subsystem of the POEtic chip is composed of two type of elements: *molecules* and *routing units*.

Molecules are reconfigurable logic elements that can be configured or programmed to implement cells. A cell is typically composed of several molecules (the number of molecules depends on the functionality of the cell). Routing units are used to implement inter-cellular communication.

Molecules are locally connected to their immediate neighbors. They can be configured to implement different functionalities such as memory elements, logic gates, etc.

The most common functionalities of the molecules are briefly explained here. In the 4-LUT[4] mode the molecule implements a programmable Boolean function of 4 inputs. In the 3-LUT mode the molecule implements two programmable Boolean functions of 3 inputs. This can be used to implement efficiently arithmetic functions (e.g. both the result of a sum and the carry can be computed by the same molecule). In the shift memory mode the molecule can store a 16-bit value. The configure mode is used to reconfigure a neighboring molecule, thus changing its functionality. This may be used to implement cellular differentiation according to the developmental system. Eventually two other modes, called input and output, are used for inter-cellular communication in conjunction with the routing units.

Routing units are used for connections among cells. Routing units are locally connected between them, and in addition each routing unit is connected to a group of four neighboring molecules, as illustrated in figure 4.6. Routing units support a mechanism known as *dynamic routing* that can build connections automatically and at run-time between input and output molecules. Dynamic routing relies on addresses or identifiers stored in input and output molecules. A breadth first search algorithm implemented in the routing units creates connections between input and output molecules that have the same identifier. Connections can also be changed, added or removed at run-time by locally reconfiguring the address in the input or output molecules. Therefore new connections can be created at run-time, even if they are not initially planned.

Figure 4.7 illustrates the organic subsystem of the POEtic chip configured to implement 4 fictive cells which are interconnected with the dynamic routing mechanism. The input and output molecules are indicated in the figure, together with their respective addresses, and the path created by the dynamic routing mechanism.

4.4.2 Chip manufacturing

The POEtic chip is manufactured in a 0.35 μm CMOS AMI process with 5 metal and 1 polysilicon layers.

Figure 4.8 shows the layout of a test chip that was manufactured before the final chip. The left figure shows the chip in the development tools, and the corresponding micrograph of the manufactured chip is shown on the right. The size of this test chip is 13 sq. mm. It includes the organic subsystem with three groups composed of 4 molecules and one routing unit, and the environment subsystem with the POEtic CPU and a timer and multiplier as sole peripherals.

[4]LUT means look-up table. That is the 4-LUT function is implemented by a look-up table with 4 inputs.

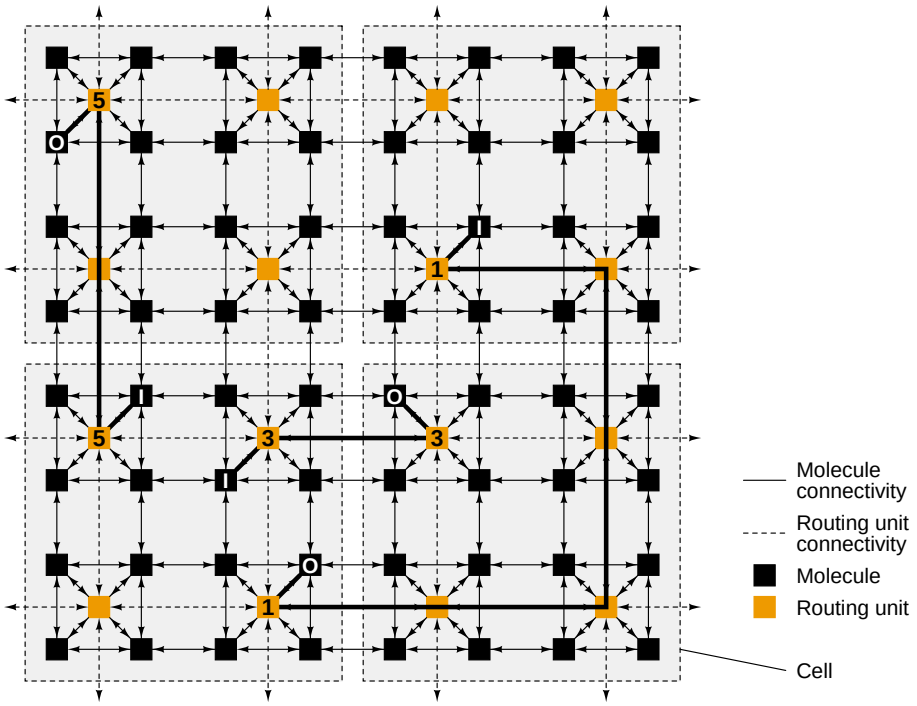

Figure 4.7: Cells of POEtic circuits are implemented in the organic subsystem of the POEtic chip. The figure depicts the organic subsystem which implements 4 cells. The three-layered structure of cells is not represented here to simplify the figure. The cells are interconnected via the routing units using dynamic routing. There is one routing unit for each group of 4 molecules. Input or output molecules (shown by 'I' and 'O' in the figure) have addresses which are used by the routing units to perform dynamic routing. These addresses are represented by a number in the routing unit bordering input or output molecules. Once dynamic routing is started, the dynamic routing units create connections between the routing units that have the same address. These connections are indicated by the thick lines over the routing unit connectivity lines.

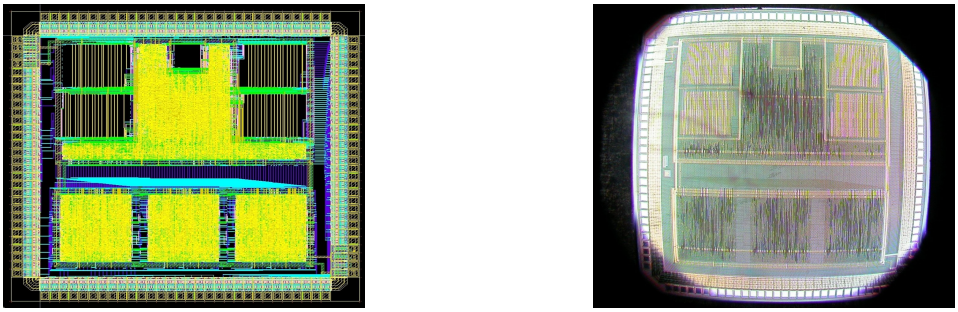

Figure 4.8: Layout (left) and micrograph (right) of the POEtic test chip. The upper half of the chip consists of the CPU, with the timer and multiplier peripherals. The lower half consists of 3 easily distinguishable groups which are each composed of 4 molecules and one routing unit.

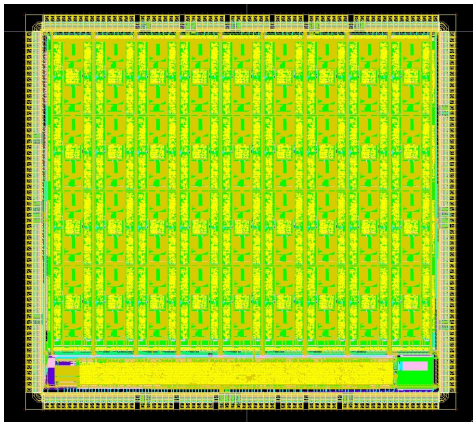

Figure 4.9: Layout of the final POEtic chip. The CPU and its peripherals are located at the bottom of the chip. The rest of the space is occupied by 144 molecules and 36 routing units organized as a 9 by 4 array.

The final POEtic chip contains all the peripherals described in section A.1. Its organic subsystem is composed of 144 molecules organized as an 18 by 8 array of molecules. Since each group of four molecules shares a routing unit, these routing units are organized in a 9 by 4 array. Figure 4.9 illustrates the layout of the final chip that is 35 sq. mm.

4.5 Summary

In this chapter we described an architecture that is suited to implement evolution, development and learning mechanisms in hardware. The architecture consists of a 2D array of cells with each cell containing the entire genetic description of the circuit, a mapping mechanism that implements the developmental system, and finally a functional part.

This architecture describes the “concept” of a circuit. The physical realization of POEtic circuits is done in a reconfigurable device such as the POEtic chip. The use of a reconfigurable device allows to change the evolutionary, developmental or learning mechanisms depending on the application of the circuit.

The POEtic chip contains an organic subsystem where the multi-cellular circuits (or organisms) are implemented. It contains also an environment subsystem (composed of a processor and peripherals) that is used to evolve the genetic string of the circuit, and to interface the circuit with the outside of the chip (i.e. the environment).

The organic subsystem of the POEtic chip contains two important features for the implementation of POEtic circuits. Dynamic routing allows to create or change connections between cells at run-time. This may be useful to implement self-repair or self-reproduction mechanisms. Self-reconfiguration allows a logic element to change the functionality of another one, and this may be used to change the functionality of cells at run-time when they differentiate.

In this chapter we did not describe any of the evolution, development or learning mechanisms. This is the object of the following chapters. In particular in the next chapter

we demonstrate how the architecture introduced here is implemented in the POEtic chip to create functional multi-cellular circuits.

5 Evolution and growth of a multi-cellular circuit

Abstract[1]

In the previous chapter we described a multi-cellular architecture suited to integrate mechanisms of evolution, development and also learning. We explained how this multi-cellular architecture could be implemented in a reconfigurable device to obtain POEtic circuits. We however did not implement a functional circuit following this architecture. The objective of this chapter is to show the implementation of this multi-cellular architecture on the POEtic chip in a functional circuit (without learning mechanisms yet). Cells of the multi-cellular circuit contain the complete genetic code of the circuit, a simple mapping layer with a direct genotype to phenotype mapping that allows the circuit to grow from an initial cell, and a functional part that computes a logic function. We demonstrate that this multi-cellular circuit can be evolved to approximate Boolean functions and to control the navigation of a mobile robot in a task of obstacle avoidance. Finally we discuss how this circuit may be extended to handle circuit self-repair and self-replication.

5.1 Introduction

In chapter 4 we described a multi-cellular architecture that allows to combine mechanisms of evolution, development, and eventually also learning. Cells of this architecture are composed of three different layers: a genotype layer containing the genetic code of the circuit, a mapping layer that controls the differentiation of the cell, and a phenotype layer that is the operative part of the cell. In that chapter we explained that this architecture could be implemented in the POEtic chip. We did not however describe any functional

[1]This work is a joint project with Yann Thoma of the Logic System Laboratory, EPFL, Switzerland. It was published in [137]. Yann Thoma designed the cell, prototyped the system on an FPGA board, modified the VHDL code of the molecules to allow simulations even in case of combinational loops, and modified to the POEticMol software (low-level molecule editor) to allow co-simulation of the molecules with the CPU. I did the high-level architecture of the system in VHDL including the configuration and I/O interface. I developed the tools to emulate the CPU and co-simulate the CPU with the POEticMol software. Finally I wrote the software running on the CPU and did all the evolutionary experiments.

circuit.

The objective of this chapter is to describe how this architecture is implemented in the POEtic chip[2] to provide functional circuits. For this purpose a simple mapping mechanism and a simple cell functionality are considered. Cells of the circuit contain the complete genetic description of the final circuit. The mapping mechanism is a direct genotype to phenotype mapping that allows the multi-cellular circuit to "grow" from an initial cell. We refer to circuit "growth" and not "development" because of the direct genotype to phenotype mapping. We reserve the term development when an indirect genotype to phenotype mapping is used. Cells differentiate according to the genetic string and to the mapping mechanism, and they take a specific connectivity and functionality in the circuit. The functional part of the cell implements a simple Boolean function (the cell is not capable of learning, learning is considered in chapters 8 and 9).

The functionality of the multi-cellular circuit is found by evolving the circuit genetic string using a genetic algorithm. The genetic string encodes the functionality and the interconnections of the cells. The multi-cellular circuit is demonstrated in two applications: the approximation of Boolean functions (adder and multiplexer), and the control of the navigation of a mobile Khepera robot in a task of obstacle avoidance.

We also discuss how the mechanism of growth may be extended to implement circuit self-repair and self-replication mechanisms.

This chapter is organized as follows. The circuit and cell structure are described in section 5.2. Section 5.3 shows how the circuit is evolved. Finally the results are discussed in section 5.4 before concluding in section 5.5.

5.2 Multi-cellular circuit and cell

The overall circuit is composed of two cells (figure 5.1). From the functional point of view, each cell implements a Boolean function of three inputs. The circuit has 6 external inputs (In 0 to In 5 in the figure), e.g. to connect to sensors, and 2 outputs (Out 0 and Out 1 in the figure), e.g. to connect to actuators.

Evolution is applied to the cell functionality (i.e. the Boolean function can be modified) and to the cell input connectivity. The input of a cell can come from one of the external inputs, or from the output of one of the two cells.

Cells are composed of the three layers (genotype, mapping and phenotype layers) described in chapter 4. The processor of the POEtic chip first configures the molecules of the organic subsystem (i.e. the reconfigurable logic within the POEtic chip) with the two cells complete with the genetic code of the entire circuit. These cells are initially undifferentiated and unconnected. Growth then starts, and according to the genetic string of the circuit the cells interconnect and differentiate by expressing a corresponding part of the genetic code. Afterwards the processor applies inputs to the circuit, reads the outputs of the circuit, and it measures the fitness of the circuit from its behavior.

[2]The architecture is actually implemented in a prototype of the POEtic chip, that consists of a FPGA configured to emulate the POEtic chip [137].

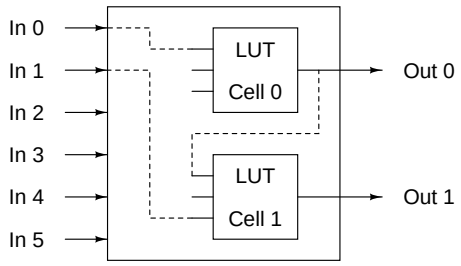

Figure 5.1: Functional view of the multi-cellular circuit with two cells that can compute logic functions of three inputs with a look-up table (LUT). The circuit has two outputs (Out 0 and Out 1) and six external inputs (In 0 to In 5). The outputs of the circuit are connected by design to the outputs of the cells. The function of the cell (i.e. the content of the LUT) and the connections of the inputs of the cells are evolved. The dashed lines symbolize these evolved connections.

The mechanisms implemented in the three layers of the cell are illustrated in figure 5.2 and described below.

5.2.1 Phenotype and genotype layers

Functionally, the cell implements a Boolean function of 3 inputs. This is implemented by a 3-LUT molecule (i.e. a molecule that implements a 3-input look-up table). Three input molecules are used to receive signals from other cells or external inputs, and one output molecule provides the result of the Boolean function for other cells and the output of the circuit.

The Boolean function and the interconnections between the cells and the inputs of the circuit are encoded in the genetic string. The genetic string contains 4 genes for each cell, each stored in one molecule.

The first gene encodes the functionality of the cell, which is simply the content of the look-up table of the 3-LUT molecule. The three other genes encode the connectivity of the three inputs of the cell. The inputs of the cell can be connected to any of the 6 external inputs of the circuit and to the output of any of the two cells. The connections of these inputs are implemented with the dynamic routing mechanism of the POEtic chip. For this purpose the 6 external inputs and the outputs of the two cells have predefined addresses. The genes coding the interconnections of the cell simply contain the address of the signal to which the input must be connected. Dynamic routing then takes care of creating the connections.

When the cell differentiates, the content of the corresponding 4 genes is transferred into the 3-LUT molecule and into the 3 input molecules. Once the content of these molecules is programmed, the cell takes the corresponding functionality, and the inputs of the cells are connected with the dynamic routing mechanism according to the addresses stored in the input molecules.

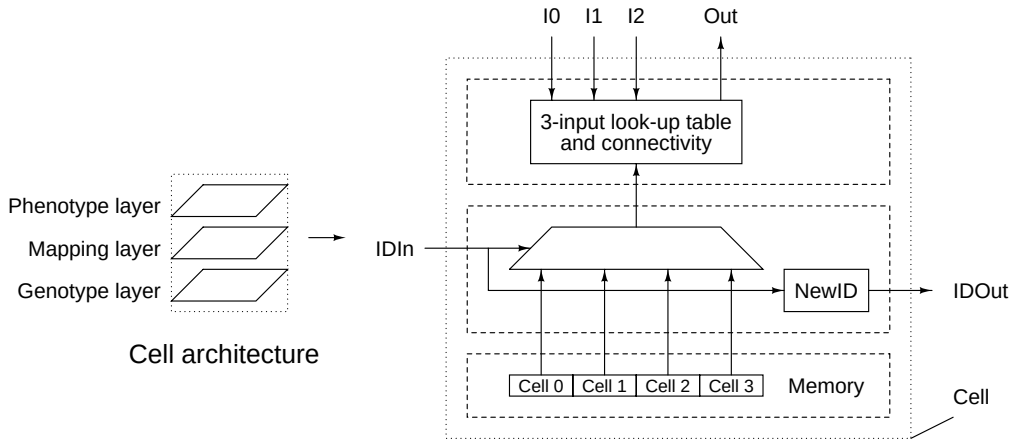

Figure 5.2: Cells of POEtic circuits are composed of three layers: phenotype, mapping and genotype (left). The mechanisms implemented in these layers are represented on the right. The genotype layer is a memory storing the genetic code of the entire circuit (here depicted storing the configuration of 4 cells). The mechanism in the mapping layer allows the multi-cellular circuit to grow from a single cell and controls the functional differentiation of the cell. This mechanism selects the appropriate part of the genetic code according to the identifier of the cell IDIn (i.e. the position of the cell in the circuit) and configures the phenotype layer of the cell with the appropriate functionality. At the same time the mapping mechanism transmits to another cell the identifier IDOut which becomes the identifier of the next cell in the circuit. That cell uses this identifier in the same way to differentiate. The phenotype layer implements a 3-input look-up table. This means that the cell can implement a Boolean function of 3 inputs (I0 to I2). The content of the look-up table and the connectivity of the inputs of the cell are encoded in the genetic string and provided by the mapping mechanism.

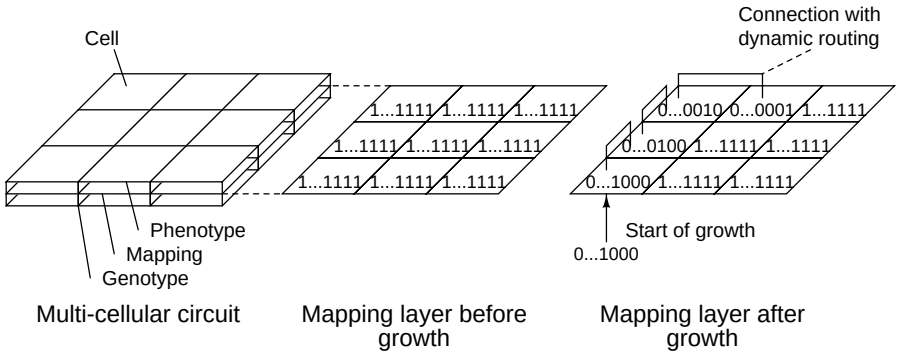

Figure 5.3: The growth phase occurs at the level of the mapping layer of the multi-cellular circuit (left of figure). Growth assigns to each cell a unique identifier (ID). Here the growth of a 4-cell circuit in a 3x3 array of cells is illustrated. Before growth, all the cells have an ID of 1...1111 (middle of figure). The growth process starts from the lower-left cell that is initialized by the POEtic processor with the ID 0...1000 (right of figure). The cell then connects to another cell in the array using the dynamic routing mechanism and transfers the ID shifted by one. Hence the second cell (middle left cell) receives the ID 0...01000. This process continues until a cell receives the ID 0...0001, at which point growth process is completed.

5.2.2 Mapping layer: growth and differentiation

The mapping mechanism builds the multi-cellular circuit from the genetic code of the circuit. It consists of two phases, growth and differentiation. The growth phase is used to assign to each cell a unique 16-bit identifier or ID (i.e. each cell receives a sequential number). This identifier is a number that indicates the position of the cell in the circuit. It is used during the differentiation phase to select the appropriate part of the genetic string to express.

For a circuit composed of n cells, the identifier of the first cell in binary is a 1 followed by n-1 0's, with n the number of cells in the circuit. Here two cells are implemented (n=2) hence the identifier of the first cell is 0...00010.

The POEtic processor first configures the entire organic subsystem with the cells and the genetic code of circuit. These cells are yet unconnected and undifferentiated: all the cells have an identifier of 1...11111. This identifier is also used as the address of the cell. It is used to connect to undifferentiated cells with the dynamic routing and trigger their differentiation. All the cells are initially inactive until they get connected with the dynamic routing.

The POEtic processor initiates the growth by connecting to an undifferentiated cell with identifier 1...11111 and transmitting the identifier of the first cell in the circuit over this connection. This identifier is IDIn in figure 5.2. At this point, the cell which got connected activates (the others stay inactive waiting for a connection). Logic within the mapping layer of the activated cell computes the identifier IDOut of the next cell (see figure 5.2): IDOut is equal to IDIn shifted by one (e.g. if IDIn is 0...01000 then IDout is 0...00100). Afterwards this cell connects with the dynamic routing to another undifferentiated cell with identifier 1...11111 and transmits IDOut, which becomes the identifier of that cell. The newly connected cell repeats the same process, which continues until a cell

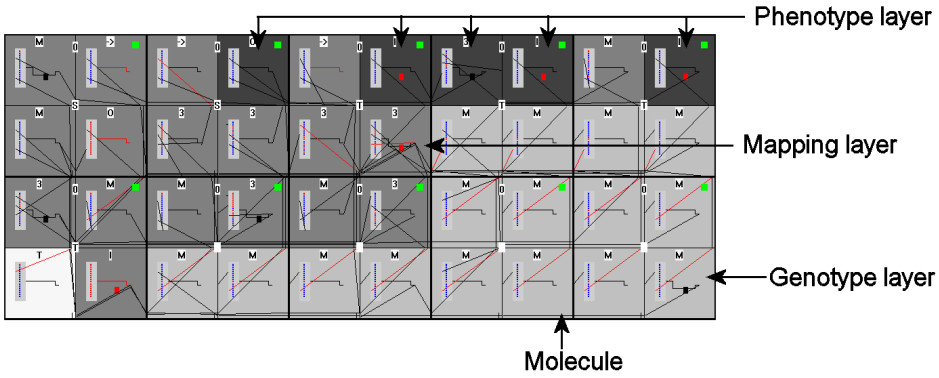

Figure 5.4: The cell is composed of 10 by 4 molecules. Five molecules form the phenotype, 16 molecules store the genotype and the remaining cells are used for the growth and differentiation (mapping layer).

receives the identifier 0...00001. This indicates that it is the last cell in the multi-cellular circuit, and the cell does not attempt to connect. This process is illustrated for a 4-cell circuit in a 3x3 array of cells in figure 5.3.

Afterwards the cells differentiate according to their identifier (see figure 5.2). The identifier indicates which genes in the genetic string are used to program the functional part of the cell. The configuration of each cell is stored in 4 genes. The first cell thus expresses the first 4 genes, the second cell the following 4 genes, etc. The 4 genes are used to program the 4 molecules implementing the functional part of the circuit, as described in section 5.2.1. This is done using the self-reconfiguration capabilities of the molecules of the POEtic chip.

5.2.3 Implementation

The cell is composed of 40 molecules, as illustrated in figure 5.4. The genotype layer is implemented with 16 memory molecules that store the genetic code of the entire circuit. Each cell of the multi-cellular circuit needs 4 molecules to store its configuration, therefore the cell developed here can be used for systems of up to 4 cells. The mapping from genotype to phenotype is done by 18 molecules: 8 molecules are responsible for the growth process, and 10 molecules used for differentiation. The phenotype layer is implemented by 5 molecules. Three of them serve as inputs (input molecules). One is the functional part of the cell, that implements the Boolean function of 3 inputs (a 3-LUT molecule). The last one is the output (output molecule).

5.3 Circuit evolution

The genetic string of the circuit is evolved with a genetic algorithm (GA). The circuit is evolved to approximate Boolean functions (adder and multiplexer) and to control the navigation of a Khepera robot in a task of obstacle avoidance.

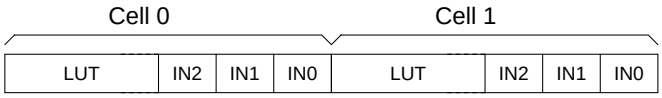

Figure 5.5: The genetic string of the circuit is composed of 34 bits (17 bits per cell). The connectivity of the inputs of the cells is coded on 3 bits, whereas the lookup table is coded on 8 bits.

Inputs			Multiplexer	Full adder	
A	B	Sel or Cin	Out	Cout	Sum
I_2	I_1	I_0	O_1	O_1	O_0
0	0	0	0	0	0
0	0	1	0	0	1
0	1	0	0	0	1
0	1	1	1	1	0
1	0	0	1	0	1
1	0	1	0	1	0
1	1	0	1	1	0
1	1	1	1	1	1

Table 5.1: Truth table of the multiplexer and of the full adder. Input I_0 is the select of the multiplexer or the carry-in of the full adder.

The genetic code is illustrated in figure 5.5. Eight bits are used to encode the content of the LUT and three bits are used for each cell input to encode the connectivity (i.e. the input of each cell can come from 8 possible sources, the 6 external inputs or the output of the two cells, therefore 3 bits are necessary to encode the connectivity). The genetic code is thus 17 bits per cell and the complete genetic code takes 34 bits. Inputs and outputs of the circuit are set and read by the POEtic processor, that also runs the GA and computes the circuit fitness.

5.3.1 Evolution of logic functions

The multi-cellular circuit is evolved to implement a multiplexer and a full adder. Three inputs are used (inputs In 0 to In 2) while inputs 3, 4 and 5 are set at all time to constant values 0, 1 and 0 respectively (see figure 5.1 for the inputs and outputs of the circuit). The multiplexer uses one output whereas the adder uses two, that are the sum and the carry. Table 5.1 shows the truth table corresponding to the two functions. Circuits are evolved by a GA with the following parameters: population size of 200, rank selection of the 20 best individuals (each selected individual is reproduced 10 times), 5% of mutation rate, one-point crossover rate of 30%, and elitism that copies the best individual unchanged in the new generation.

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 thus 8 and 16 for the multiplexer respectively the full adder. Evolution is performed with a simulation of the multi-cellular circuit to speed up experiments (the software simulation tools are described in appendix B). Figure 5.6 shows

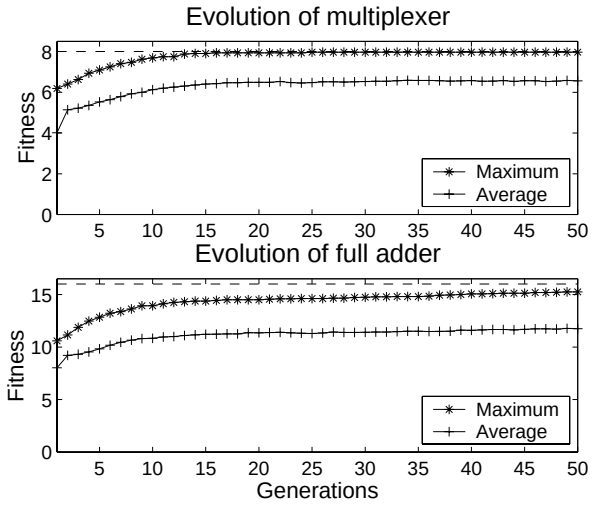

Figure 5.6: Maximum and average fitness over the generations when evolving logic functions (average of 32 runs). The horizontal dashed line represents the maximum fitness.

the fitness over the generations averaged on 32 runs for the two functions. Evolution manages to implement the multiplexer in 31 of the 32 runs (in one run the maximum obtained fitness 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 evolve than the full adder because it is a simpler circuit that can be implemented with only one cell whereas the full adder needs two.

5.3.2 Evolution of a robot controller

The circuit is evolved to control the navigation of the two-wheel differential drive Khepera mobile robot [120] in a task of obstacle avoidance. In other words, the circuit maps sensory inputs to motor commands. The Khepera robot has 8 proximity sensors. The proximity sensors are thresholded and connected to the inputs of the circuit. When the robot is closer than about 3 cm from a wall, the sensor value is 1, otherwise it is 0. The motors of the Khepera are controlled by the output of the cells. An output of 1 corresponds to a wheel speed of +80 mm/s, while an output of 0 corresponds to a speed of -80 mm/s. Figure 5.7 shows the mapping of the sensor inputs and motor outputs on the circuit. Some sensors are grouped by taking the value of the most active sensor.

The robot runs in a rectangular 40x65 cm arena. The robot has a sensory motor period of 100 ms during which the speeds of the wheels remain constant. At the end of the period, the outputs of the circuit are read and the speed of the wheels is updated. Afterwards the proximity sensors are read and the inputs of the circuit are set accordingly.

The fitness of the circuit is determined from the behavior of the robot. The fitness function rewards straight motion, which implicitly leads to obstacle avoidance. The fitness function is the sum of the normalized speeds of the wheels at each sensory-motor step, when both wheels have positive speeds (spin forward) [40].

The circuit is evolved using the same GA parameters as in section 5.3.1. Evolution is

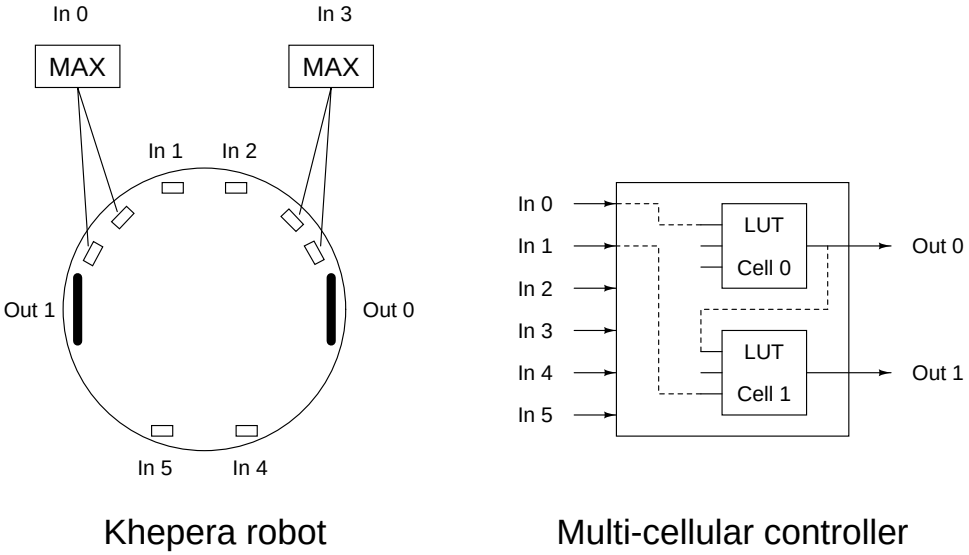

Figure 5.7: Mapping of the Khepera proximity sensors and actuators (shown on the left) on the multi-cellular circuit (right). Sensor readings are mapped to inputs In 0 to In 5 of the circuit, while the outputs Out 0 and Out 1 control the speed of the wheels. Some sensors are grouped by taking the value of the most active sensor.

performed with a simulation of the robot and a simulation of the multi-cellular circuit, to speed up experiments. Figure 5.8 shows the evolution of the fitness over the generations averaged on 32 runs. The best controller is already capable of obstacle avoidance after about 10 generations.

5.4 Discussion

The implementation of a cell requires 40 molecules. Comparatively, the functional part of the cell requires only 5 molecules. There is thus a significant resource overhead needed to implement the developmental mechanism and the memory storing the circuit genetic string. However the developmental system can potentially offer benefits in terms of self-repair and self-replication, as discussed below. In addition, the cell functionality used here is very simple. If more complex cell functionalities were used (e.g. neurons as in chapters 7 and 9) the overhead would proportionally be reduced.

The mapping mechanism that we described in this chapter can be applied to cells of any shape, which can be physically placed anywhere in the organic subsystem, even at irregular intervals. This flexibility is provided by the dynamic routing that takes care of interconnecting automatically the cells. Placing cells at irregular intervals may be necessary if parts of the organic subsystem are damaged: cells can be placed on the functional molecules (an off-line test may reveal the damaged locations), and dynamic routing (assuming it is not damaged) takes care of connecting the cells in a transparent way.

Alternatively, cells could be interconnected by the processor in the POEtic chip when it configures the organic subsystem with the cells for the first time. This solution however

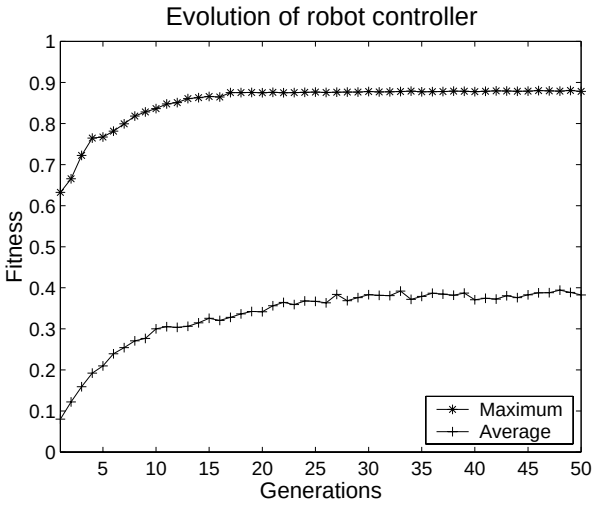

Figure 5.8: Evolution of the maximum and average fitnes of the robot controller (average of 32 runs). Maximum fitness is 1.

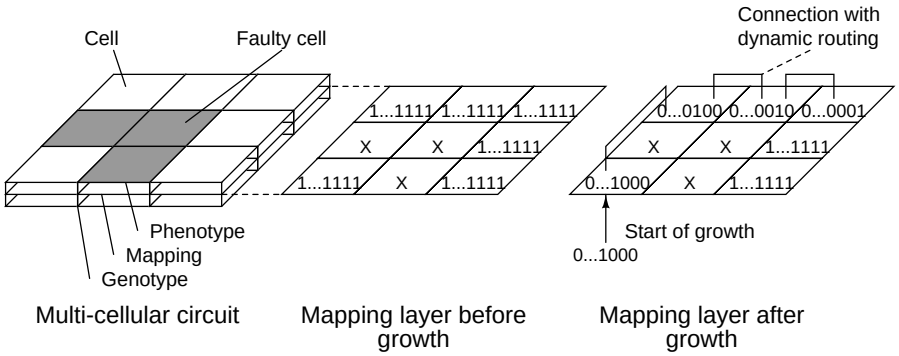

Figure 5.9: Possible extension of the growth mechanism to provide self-repair. A 4-cell circuit develops on a 3x3 array of cells with three faulty cells that are indicated in gray. Since faulty cells deactivate their dynamic routing inputs, the growth process automatically makes use of spare cells available in the circuit.

does not provide the same flexibility as the mechanism implemented here, especially if the cell locations in the circuit vary (i.e. in case of faults).

The mapping mechanism described here may be used as a foundation to implement self-repair and self-replication mechanisms. Self-repair requires a mean to detect faults in the circuit. This can be done for instance by functional redundancy within the cells or following the Immunotronics approach [14]. Upon detection of a fault, the defective cell would go off-line by deactivating the input molecule used for dynamic routing during growth. This can be done with local self-reconfiguration. At this stage a new development process would be triggered. Since growth relies on dynamic routing to connect to a cell with the appropriate identifier, the faulty cell would be avoided, and a spare cell available in the circuit would be used instead. This process is illustrated in figure 5.9. As long as enough functional cells are available in the system, the development process

would automatically connect to the required number of cells to implement the desired circuit. Dynamic routing and self-reconfiguration would thus be exploited to implement self-repair mechanisms in the POEtic chip in a transparent way. Implementing this system however requires that the dynamic routing units are not damaged. It also requires that faulty cells can still deactivate their dynamic routing input. Eventually the state of faulty cells needs to be transferred to spare cells, so that the state of the circuit is not lost when a fault occurs. All these aspects remain to be investigated.

Another extension of this circuit consists in transferring the genetic code from one cell to another during growth. This modification may allow to implement self-replicating circuits in the POEtic chip. This also remains the object of future work.

Compared to classic unconstrained evolution which manipulates directly the configuration bits of a FPGA [165], evolution is performed at a higher level thanks to dynamic routing. Indeed the approach could be classified as *intrinsic schematic* evolution. Connections are evolved by encoding the address of the inputs rather than by encoding the configurations of many switch boxes. Consequently the genetic code is more compact and evolution may be faster. Note that the genetic coding resembles Cartesian genetic programming [113] that also encodes the functionality and connectivity of every cell. Circuits evolved in simulation with Cartesian genetic programming could be intrinsically evolved in this multi-cellular circuit.

5.5 Summary

In this chapter we demonstrated how the multi-cellular architecture introduced in chapter 4 can be implemented in the POEtic chip to obtain functional multi-cellular circuits. We successfully evolved these circuits to approximate Boolean functions, and to control the navigation of a robot.

We implemented a mapping mechanism that relies on dynamic routing to let the cells interconnect and differentiate at run-time. We discussed how this mechanisms may be improved to provide self-repair and self-replication capabilities to multi-cellular circuits.

The mapping mechanism implemented here uses a direct genotype to phenotype mapping for demonstration purposes. In the next chapter we introduce the morphogenetic system, a genetic encoding and developmental system that allows complex indirect genotype to phenotype mappings and that is designed especially for multi-cellular circuits.

6 Evolutionary morphogenesis for multi-cellular systems

Abstract[1]

In this chapter we introduce a minimalist genetic encoding and developmental system suited for multi-cellular systems such as POEtic circuits. We call this system the *morphogenetic system*. This morphogenetic system is inspired by gene expression and cellular differentiation. Yet it is a computationally inexpensive system that allows fast simulation and efficient hardware implementation. In particular it allows to evolve multi-cellular systems composed of any predefined high-level or low-level cell functionalities, such as the one we may wish to have in POEtic circuits. The morphogenetic system shows better scalability compared to a direct genetic encoding in the evolution of structures of differentiated cells, and its dynamics provides fault-tolerance even at high fault rates. We analyze the morphogenetic system in function of its parameters, and we describe its hardware implementation in the POEtic chip.

6.1 Introduction

In introduction we described bio-inspired POEtic circuits that encompass mechanisms of evolution, development and learning. We argued that to fully benefit from these POEtic circuits, an evolutionary system is required that takes into account their characteristics, and that combines a genetic encoding and a developmental system.

The need for a specific evolutionary system stems primarily from the issue of scalability that we evidenced in evolvable hardware, where genetic algorithms with a direct genotype to phenotype mapping are generally used (chapter 2). In addition, conventional genetic algorithms do not exploit the complex mechanisms of development which are seen in biological organisms. The mapping between the genotype and the phenotype is static and it does not allow inter-cellular or environmental interactions during development that may lead to adaptive development or fault-tolerant circuits.

[1] Part of this work was published in [135, 134].

Indirect genetic encodings based on developmental systems, that we reviewed in chapter 3, are one possibility to address these points. We remarked that developmental systems need not necessarily be biologically plausible to perform well. Simpler developmental systems may perform as well and use less computational resources. This is particularly important in evolutionary computation, that requires the evaluation of a lot of candidate solutions, and in hardware applications, that require compact implementations.

In this chapter we introduce a minimalist genetic encoding and developmental system for multi-cellular systems which we call the *morphogenetic system*. It is inspired by gene expression and cellular differentiation and attempts to achieve low computational complexity in order to allow fast simulation and efficient hardware implementation.

This morphogenetic system takes into account the requirements of multi-cellular POEtic circuits. It assumes that circuits consist of a regular 2D array of cells, as introduced in chapter 4. It assumes local communication between cells. This allows to apply the morphogenetic system to circuits regardless of their size, and this allows cells to be added or removed from the circuit during development. The morphogenetic system can be implemented in a fully distributed way (i.e. cellular implementation). This allows fast development, close interaction between the development mechanism and the environment, and this is potentially a more robust solution than a centralized implementation. Finally the morphogenetic allows inter-cellular interactions that may provide fault-tolerance.

The functionality of POEtic circuits is defined by the functionality of each cell and by their interconnections. Cell functionalities must be "compatible": they must be able to interconnect with each other and exchange information in a meaningful way, and interconnections between cells should not lead to electrical problems (e.g. short circuits). For this reason the morphogenetic system relies on predefined *cell functionalities* that are electrically compatible with each other. These functionalities may be high-level functions such as neurons, or low-level functions such as elementary logic gates. Interconnections among cells are considered as part of the cell functionalities. This can be efficiently implemented with dynamic routing on the POEtic chip.

After describing the morphogenetic system, we investigate its evolvability by evolving structures of differentiated cells, and we investigate its scalability by evolving phenotypes of different sizes. Afterwards we consider the potential benefits of the dynamics of the developmental system on fault-tolerance, and we analyze extensively the morphogenetic system. All these experiments are done with a software model of the morphogenetic system and of the multi-cellular circuit. Finally we describe the implementation of the morphogenetic system in hardware on the POEtic chip.

This chapter is organized as follows. Section 6.2 introduces the morphogenetic system. Section 6.3 investigates its capacity to evolve 2D structures of various complexity, and section 6.4 tests its scalability by evolving phenotypes of different size. The capacity of the dynamics of the morphogenetic system to withstand faults is investigated in section 6.5. The morphogenetic system is analyzed in section 6.6. In particular the effect of the parameters of the system on the fitness is studied, as well as the morphology of phenotypes obtained by the developmental process. In section 6.7 we implement the morphogenetic system in hardware on the POEtic chip. Results are discussed in section 6.8 before concluding in section 6.9.

6.2 Morphogenetic system

The morphogenetic system is inspired by the mechanisms of gene expression and cellular differentiation of biological organisms, notably by the fact that concentrations of proteins and inter-cellular chemical signaling regulate the functionality of cells [19, 188]. In particular it takes inspiration from the early stage of development, when cells differentiate according to their position in the organism, as specified by gradient of chemicals known as morphogens. Quoting Wolpert [188]:

> Cells could have their position specified by a variety of mechanisms. The simplest is based on a gradient of some substance. If the concentration of some chemical decreases from one end of a line of cells to the other, then the concentration of that chemical in any cell along the line effectively specifies the position of the cell with respect to the boundary. A chemical whose concentration varies, and which is involved in pattern formation, is called a morphogen.

The morphogenetic system assumes that the cells of the circuit can implement a function from a set of predefined functionalities (something akin to skin, muscle, neuron cells, etc. in biological organisms). The morphogenetic system allows to "develop" a multicellular circuit from its genetic code by executing two simple phases that operate in parallel. A *signaling* phase relies on local communication in the cellular circuit to exchange signals among adjacent cells to implement a diffusion process. In parallel, the *expression* phase finds the functionality to be expressed at each cell by matching the signal intensities in each cell with a corresponding functionality stored in an expression table. The genetic code contains the position of diffusing cells (*diffusers*) and the signal-function matching of the expression table. It is evolved with a genetic algorithm.

6.2.1 Family of functions

The morphogenetic system relies on a set of predefined functionalities which we refer to as a family of functions. The family of functions must include a sufficiently rich repertoire of functionalities to realize the desired circuit. However the morphogenetic system is essentially independent of the phenotype: any functionalities can be used as long as they fit in cells. For instance pixels of different colors may be used to evolve patterns, or logic gates may be used to evolve circuits.

6.2.2 Signaling phase

Inter-cellular communication allows the exchange of signals between adjacent cells. A signal is a simple numerical value (the signal intensity) that the cell owns, and that adjacent cells are able to read, akin to a chemical concentration. Signals may be of different types (i.e. of a different chemical nature). Signaling starts from *diffusers* placed in cells. There are diffusers for the different type of signals. When a cell contains a diffuser for a

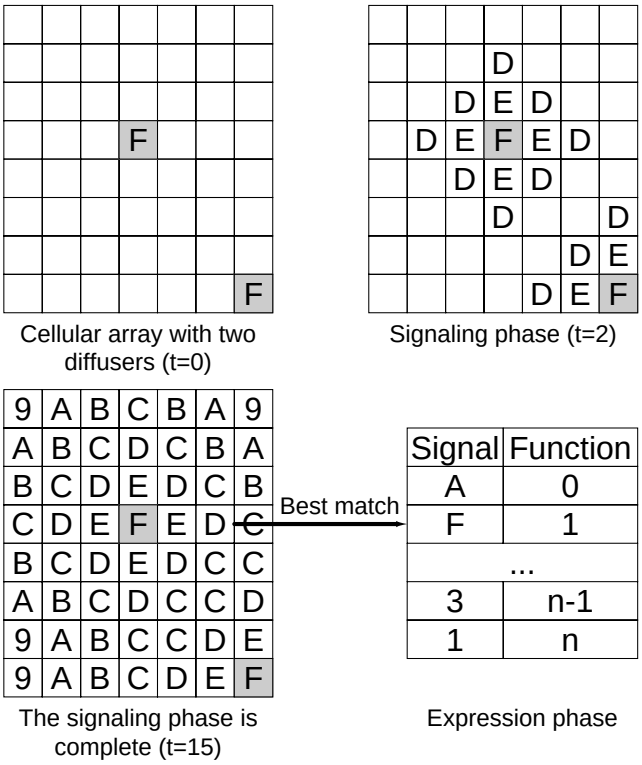

Figure 6.1: The three first arrays are snapshots of the signaling phase with one type of signal and two diffusers (gray cells) at the start of the signaling phase (top left), after two time steps (top right) and when the signaling is complete (bottom left). The numbers inside the cells indicate the intensity of the signals in hexadecimal. The expression phase (bottom right) maps the signal intensities in the cells into functionalities with an expression table that contains the signal-function matching rules. In this example the signal D matches the second entry of the table with signal F (smallest Hamming distance), thus expressing function 1.

1	Decode the chromosome to find the location of the diffusers. For all the diffusers of type t in cells j set $V_t^j = 1$ and $C_t^j = 15$, for all the other signals of type s in cells i set $V_s^i = 0$ and $C_s^i = 15$.
2	For each signal s and each cell i do steps 3 to 5:
3	If $V_s^i = 1$ then skip this cell or signal.
4	Compute the intensity of signal s. It is $C_s^i = max(C_s^j - 1, 0)$. j is any of the four neighbors of cell i for which $V_s^j = 1$. Set $V_s^i = 1$. If not such cell j exists, then $V_s^i = 0$.
5	Perform the expression phase to obtain the cell functionalities according to the signal intensities in the cells
6	Repeat step 2 to 5 15 times to complete the signaling phase. Each iteration corresponds to a *developmental step*.

Table 6.1: Algorithm of the signaling phase of the morphogenetic evolutionary system.

particular signal type, the intensity of this signal in the cell is always set to the maximum intensity.

We refer to the intensity of a signal of type s in cell i as C_s^i. The signaling algorithm (see below) only sets signals which have not yet been initialized (i.e. which have not yet been set by the signaling algorithm). For this reason each each signal in each cell has a flag which indicates if it is initialized (or valid). When $V_s^i = 1$ the signal of type s in cell i is initialized, otherwise $V_s^i = 0$.

Signals of each type are processed independently, without interactions among them, as if they were in different chemical layers. Initially, except for diffusers, all the signals are uninitialized. Signals are then set up by the signaling algorithm. The signaling algorithm ensures that signal intensities decrease linearly with the Manhattan distance to the diffusers.

The signaling algorithm is illustrated in table 6.1. All the signals are updated synchronously at the end of step 4 in the table. Step 2 to 5 correspond to one *developmental step*. Each additional developmental step expands the signals around the diffusers. Figure 6.1 illustrates this in the case of a single type of signal, with two diffusers placed on the cellular circuit. After each developmental step the expression phase (described below) is executed to obtain the functionality of the cells according to the signal intensities.

In the current implementation the signal intensities are represented by 4-bit numbers. Therefore, after $2^4 = 16$ steps the development of the multi-cellular system is completed.

6.2.3 Expression phase

The expression phase assigns a function to each cell by matching the signal intensities inside that cell with the entries of an expression table T stored in the genetic string (figure 6.1, bottom right). Figure 6.2 shows the expression table with n entries (i.e. there are n functionalities in the function family) and $S = 4$ types of signals. Each entry of the table contains the intensities of the signals and the function to express in case of match. The intensity of signal s in the entry j of the table is noted by T_s^j. A cell i is said to match an entry j of the expression table when the distance $d = \sum_{s=1}^{S} DOp(C_s^i, T_s^j)$ is minimum.

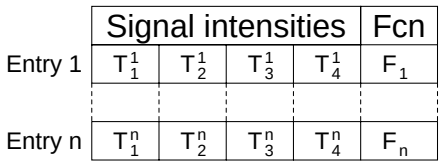

Figure 6.2: The expression table T of the morphogenetic system contains one entry for each functionality in the family of functionalities. Here the expression table is shown with n entries and therefore there are a maximum of n functionalities. For each of these functionalities the expression table contains the intensities of all the signals (here there are 4 type of signals) to which the signal intensities in the cells are compared to find the functionality to express in the cell.

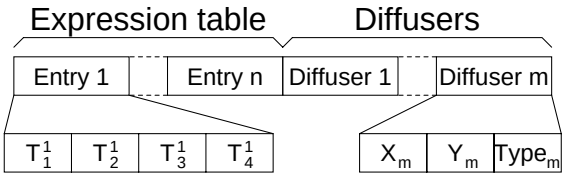

Figure 6.3: The genetic code contains two parts. The first is the expression table T, here with n entries. In this example there are four types of signals therefore each entry is 16 bits long. The second part contains the locations and types of the diffusers. The number of bits for the X and Y coordinates depends on the size of the circuit. The number of bits for the type of the diffuser depends on the number of signal types (e.g. 2 bits when 4 signal types are used).

The distance operator DOp is the bitwise Hamming distance.

6.2.4 Genetic encoding and evolution

The genetic code contains the expression table T, and the location of the diffusers (figure 6.3). The genetic code therefore affects the pattern of diffusion and the expression rules of the cells in the circuit. However it does not encode the functionalities that a cell can express, which are predefined.

In most of the experiments described in this chapter we use 4 types of signals. In chapters 7, 8, 9 we always use 4 types of signals. In this case each entry of the expression table is 16 bits (4 signals coded on 4 bits each). The functions are not encoded and evolved in these experiments. The locations of the diffusers are stored as pairs of X,Y Gray-coded coordinates, plus two bits (in the case of 4 signal types) indicating the type of the diffuser (i.e. $2^2 = 4$ type of signals). A population of genetic strings is randomly initialized and evolved using a standard genetic algorithm [50].

6.2.5 Computational requirements

The morphogenetic system is implemented using only additions, subtractions, comparisons and logic operations. There are no floating point operations and none of the costly operations of multiplication or division are required. The time required for complete development in a sequential implementation is of order $O(X \cdot Y \cdot S \cdot n)$ for an X by Y

multi-cellular system with S types of signals and n entries in the expression table. In a cellular (distributed) implementation, where each cell implements the signaling and expression mechanisms, the time is of order $O(S \cdot n)$. The number of diffusers influences only the initialization time (i.e. setting the signal intensities to the maximum value after decoding the genetic string).

6.3 Evolvability

A prerequisite for evolvability is the ability of a genetic encoding to generate phenotypes of various structure and complexity. This capacity is assessed by evolving phenotypes to resemble various 2D patterns. In this case cell functionalities correspond to pixel colors. The six 8x8 target patterns illustrated in figure 6.4 are considered. The first four patterns (*uniform*, *checkerboard*, *mixed1* and *mixed2*) are black and white (family of functions: black, white) and test whether the morphogenetic system is capable of generating uniform structures and different kinds of diversified structures. Those patterns were initially proposed in [135]. The remaining two patterns (*Norwegian flag* and *CA-generated* pattern) are composed of four colors (family of functions: black, white, blue, red). They were initially proposed in [37]. The *Norwegian flag* has symmetries which may be exploited by a developmental system as proposed in [148]. The *CA-generated* pattern is generated with a cellular automata (CA) using Wolfram's rule 90 starting from a random initial line. This rule tends to generates patterns of high complexity that may be difficult for a developmental system to evolve.

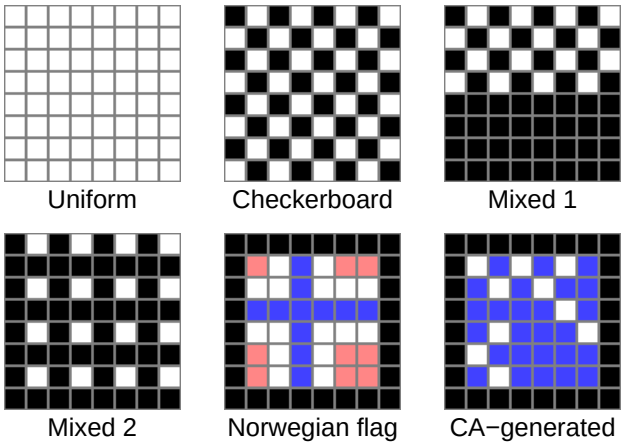

Figure 6.4: The pattern coverage task consists in evolving an array of 8x8 cells with a specific target pattern. There are six target pattern. The first four are binary patterns (*uniform*, *checkerboard*, *mixed1* and *mixed2*). The last two patterns use 4 colors and are the *Norwegian flag* and the *CA-generated* pattern.

The parameters of the morphogenetic system are the following: 2 or 4 entries in the expression table (depending on the size of the function family), 16 diffusers and 4 types of signals. This number of diffusers has been selected empirically from preliminary tests. The chromosome size is 160 and 192 bits for the 2- and 4-color patterns respectively: 2 or

4 entries in the expression table * 16 bits + 16 diffusers * 8 bits (6 bits for the coordinates and 2 bits for the type of the diffuser).

The population is composed of 400 individuals, selection is rank selection of the 300 best individuals (the first 100 best individuals are reproduced twice, the following 200 are reproduced once), the mutation rate is 0.5% per bit, one-point crossover rate is 20% and elitism is used by copying the 5 best individuals without modifications into the new generation.

The fitness is proportional to the resemblance of the phenotype to the target. In order to maintain diversity, phenotypic traits that occur often in the population decrease the fitness of the individuals that own them [37].

The morphogenetic system is compared to a direct genetic encoding where each pixel is encoded by 1 or 2 bits for the 2- and 4-color patterns, leading to a chromosome of 64 and 128 bits respectively. The parameters of the genetic algorithm are the same as those used with the morphogenetic system.

Evolution is done for 2000 generations and figures 6.5 and 6.6 show the evolution of the maximum fitness (average of 10 runs) for the first 500 generations for all the target patterns.

The morphogenetic system always reaches the maximum fitness with the *uniform*, *checkerboard* and *mixed 1* patterns, on average in 1, 8 and 49 generations respectively. The morphogenetic system cannot fully cover the *mixed2* pattern, but it comes very close (fitness higher than 0.95).

The *Norwegian flag* and *CA-generated* patterns, which display more complex structures and four instead of two colors, cannot be covered completely within 2000 generations by the morphogenetic system. However the fitness is still relatively high (average maximum fitness of 0.84 and 0.88 respectively).

The direct genetic encoding manages to reach the maximum fitness with all the patterns: the small search space and the simple structure of the fitness landscape favor that. It takes on average 51 generations to evolve the patterns of two colors, and 136 generations for the pattern of 4 colors.

In the problem considered here, an indirect encoding tends to have a more complex fitness landscape than a direct encoding because of epistatic interactions among genes, which may generate more rugged fitness landscapes (see section 6.6.4 for a preliminary analysis). Furthermore some phenotypes may not be expressible because the genotype to phenotype mapping may not allow some regions of the phenotypic space to be encoded. However this does not mean that direct genetic encodings should be favored, since direct encodings are not well suited for multi-cellular circuits capable of dynamic reorganization for which the morphogenetic system is designed.

Still, the morphogenetic system is capable of generating patterns of various complexity with regular and irregular structures with a relatively high fitness. The fine details of the phenotype may be left to an epigenetic mechanism, such as local hill climbing or Hebbian learning for neural networks, while the phylogenetic mechanism deals with structures at a coarser level. In this case the fact that specific phenotypes cannot be exactly evolved may not be an issue.

The process of development is illustrated in figure 6.7 for a larger 64x64 *Norwegian*

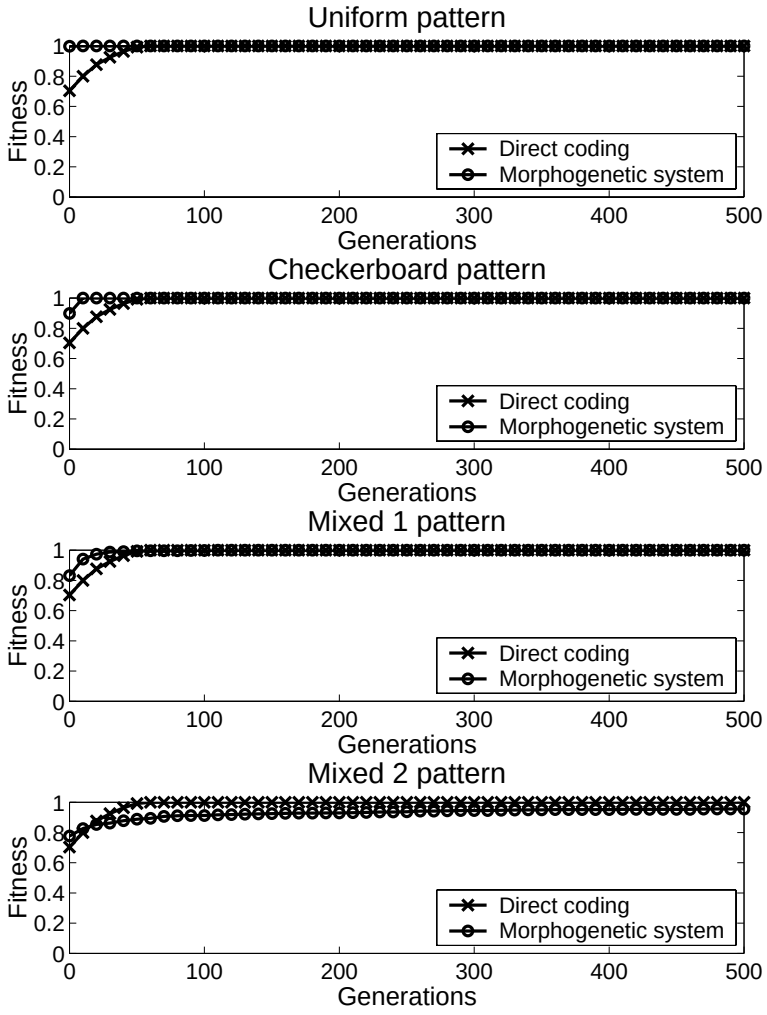

Figure 6.5: Evolution of the maximum fitness for the black and white patterns (average of 10 runs).

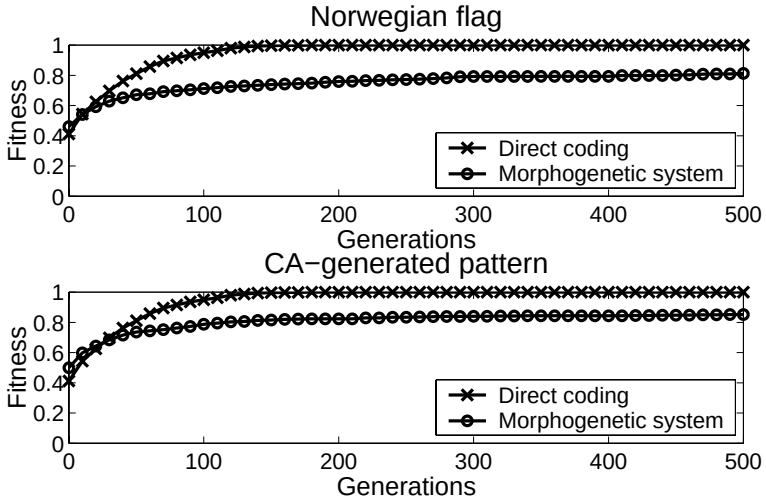

Figure 6.6: Evolution of the maximum fitness for the *Norwegian flag* and *CA-generated* patterns (average of 10 runs).

flag (the 8x8 flag develops in very few steps, larger phenotypes make a better illustration of development). Each picture represents the functionality of the multi-cellular circuit after one developmental step, that is each picture is taken after the signaling phase updates the signal intensities and the expression phase maps these signal intensities into cell functionalities (i.e. at step 5 of the algorithm shown in table 6.1).

6.4 Scalability

The scalability of the morphogenetic system is assessed and compared to a direct genetic encoding by evolving phenotypes of increasing size. Two phenotypes are considered, the *Norwegian flag* and the *CA-generated* pattern. Phenotype sizes are: 8x8, 16x16, 32x32, 64x64, 96x96, 128x128 and 256x256. The *Norwegian flag* is a scaled version of the 8x8 flag. The *CA-generated* pattern is computed using the CA rules from a wider initial line.[2] Therefore its complexity increases. Figure 6.8 illustrates the target patterns for phenotypes of size 64x64.

The size of genetic string with the direct encoding scales with the size of the array. The size of the genetic string of the morphogenetic system scales only with the logarithm (base 2) of the size of the array. The lengths of the genetic strings for the direct coding

[2]With the exception of the patterns smaller than 32x32 for which the border has a fixed width of one pixel, the *Norwegian flag* scales in length with the size of the pattern: the width of the border and the widths of the crosses inside the flag are proportional to the width of the pattern. The pattern width, the width of the border, the width of the outer cross (white) and the width of the inner cross (blue) are listed in this order in the following tuples: (8,1,1,1), (16,1,1,3), (32,1,2,5), (64,2,5,11), (96,3,7,15), (128,4,10,21), (256,8,20,41). The border of the *CA-generated* pattern is always one pixel wide. The first line of this pattern is randomly generated (each pixel takes randomly one of two colors), and the following lines are computed from the line above them using cellular automata rule 90.

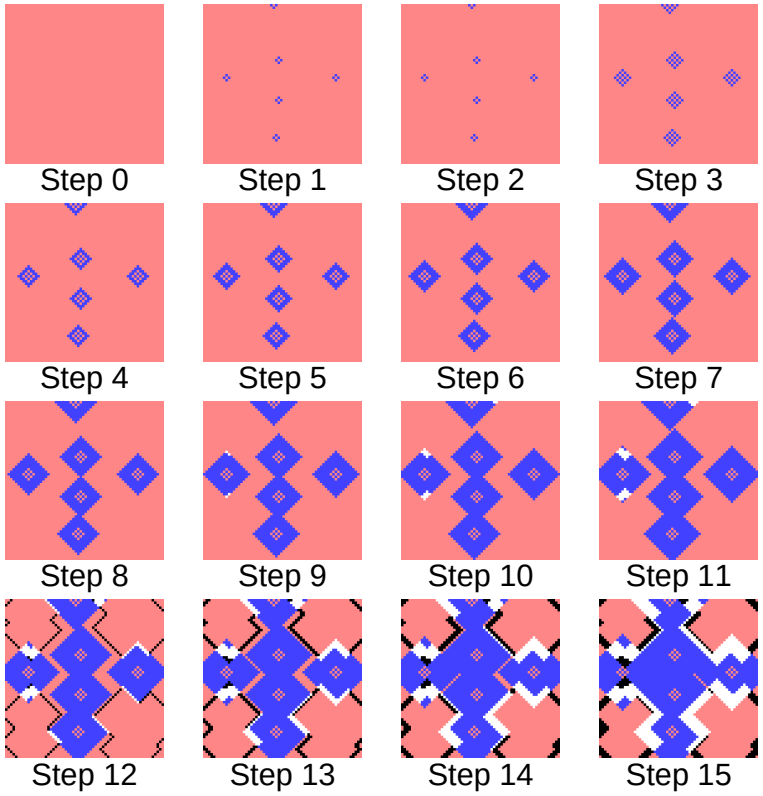

Figure 6.7: Development of the 64x64 *Norwegian flag*. Each picture represents with gray levels the type of all the cells expressed in the multi-cellular circuit. Each picture is a taken after a development step. After 16 steps the development is completed. The top-left picture shows the type of all the cells of the multi-cellular circuit at the beginning of the developmental process. At this stage most cells (with the exception of those having diffusers) have signals whose intensities are uninitialized (i.e. the signal intensities correspond to the reset state of the cell). The evolutionary process yet adapts the expression table so that these cells express the background color of the flag, which happens to be the most common color in the target pattern.

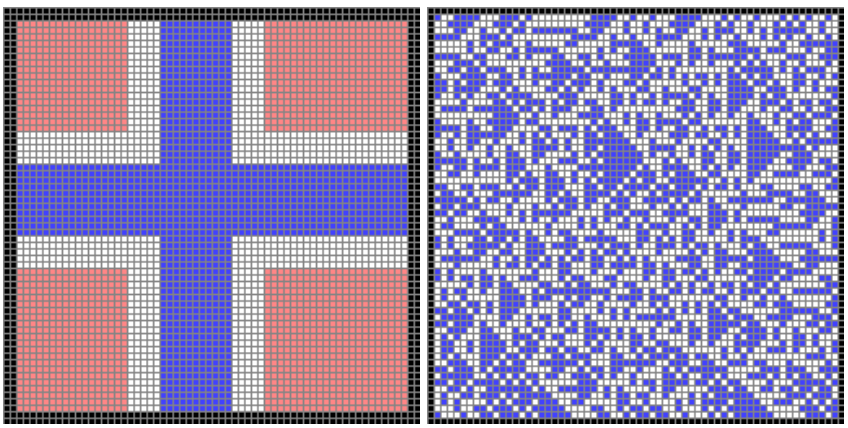

Figure 6.8: The patterns used to assess the scalability of the morphogenetic system are the *Norwegian flag* and the *CA-Generated* pattern, here illustrated with size 64x64.

	Genetic encoding	
Phenotype size	Direct encoding	Morphogenetic system
8x8	128	192
16x16	512	224
32x32	2048	256
64x64	8192	288
96x96	18432	320
128x128	32768	320
256x256	131072	352

Table 6.2: Size of the genetic string in bits for the direct encoding and the morphogenetic system for the various phenotype sizes.

and the morphogenetic system are listed in table 6.2.

Evolution is performed with the same parameters as in section 6.3 for 2000 generations and the maximum and average of the maximum fitness scores are measured at the last generations. Figure 6.9 illustrates the scalability of the direct genetic encoding and the morphogenetic system. With the largest phenotypes (256x256) the morphogenetic system achieves a higher fitness than the direct genetic encoding, which indicates that it scales better than the direct encoding on this problem.

The direct encoding reaches the maximum fitness for arrays up to size 32x32. The larger search space then limits the fitness scores for larger arrays. The morphogenetic system achieves lower fitness scores than the direct encoding on the smaller patterns (up to size 64x64 and 96x96 for the *Norwegian flag* and the *CA-generated* pattern respectively). However with larger phenotypes the morphogenetic system outperforms the direct encoding with both patterns.

The morphogenetic system tends to exploit the frequency of the colors to maximize the fitness because it can easily generate large patches of cells with a uniform color. This happens, even if there are no diffusers, by expressing the most common color when signal intensities are uninitialized. Indeed even if signals are uninitialized, they still have a default "reset" value which can be exploited by the evolutionary process to express a default type of cells. This is illustrated in figure 6.7 at the beginning of development: except for diffusers cells all have uninitialized signals, yet they all express the background color of the *Norwegian flag*, which is the most common color in the target circuit. As a consequence the morphogenetic system starts with higher maximum fitness than the direct encoding in the first generation. It achieves a maximum fitness close to 0.5 in the first generation with the *CA-generated pattern* which mostly contains an equivalent distribution of two colors, even though each cell can take one of 4 different colors. With the *Norwegian flag* it expresses by default the background color of the flag because it is the most frequent color. It then places diffusers on the branches of the flag to generate locally the correct colors (figure 6.7).

Therefore it is likely that in the worst case the fitness obtained with the morphogenetic system does not drop below the normalized area that the most common color in the phe-

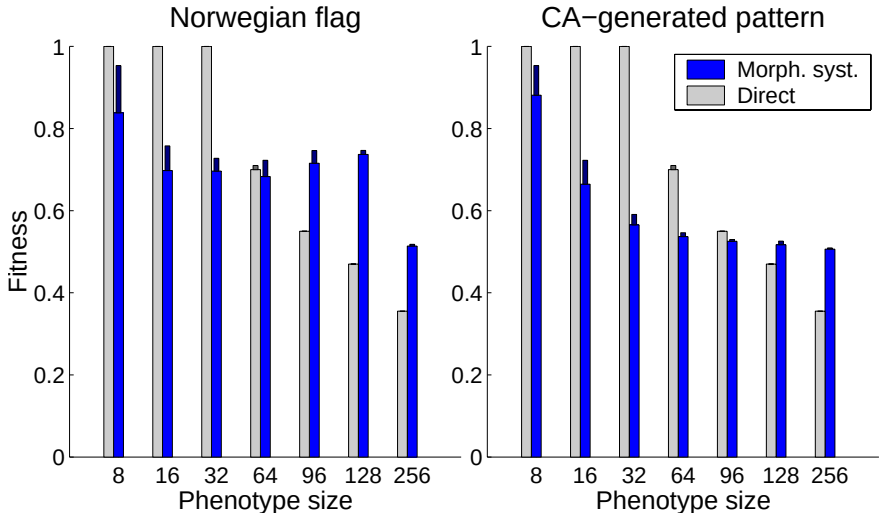

Figure 6.9: Scalability of the morphogenetic system and direct encoding when evolving the *Norwegian flag* and the *CA-generated pattern*. The bars indicate the maximum and the average of the maximum fitness scores measured at the last generations of 20 runs (for the 128x128 and 256x256 pattern 5 runs were done because of the long time required for the runs to complete). The wider bar indicates the average of the maximum fitness obtained in each run. The thinner bar indicates the absolute maximum fitness. The morphogenetic system scales better than the direct encoding on the larger phenotypes.

notype covers, even for larger phenotypes. In comparison the direct genetic encoding starts with a maximum fitness score that is close to 0.25 because each pixel is randomly assigned to one of the four possible colors and has a $\frac{1}{4}$ probability of matching the target pixel color.

For comparison purposes the number of diffusers (16) remained identical for all the phenotype sizes. This is not enough diffusers to entirely cover the larger multi-cellular arrays with signals and fitness may be improved by increasing the number of diffusers. This is investigated in section 6.6.

In summary, the morphogenetic system scales better than the direct encoding when increasing the size of the target phenotype.

6.5 Fault-tolerance

In this section we show that the dynamics of development may be exploited to provide fault-tolerance to evolved patterns of cells.

We consider transient events that damage the state of the cell. This can happen for instance when radiations corrupt memory elements. We assume that development continues to operate normally. In this case the cell functionality may be recovered. We assume that the necessary circuitry is available to detect corrupted states (e.g. by doing periodical checksums of the cell's memories) and that a reset of the cell ensues in such cases. Therefore in case of fault all the variables describing the state of the cell take the default reset

value. The cell forgets whether it was diffusing signals, and all the signal intensities are flagged as uninitialized. Here we show how the intrinsic dynamics of the morphogenetic system can recover the cell functionality after a reset, not actually on how to design the fault detection mechanism.

The state of a cell depends on the expression table and on the signal intensities. Both are be stored in memories, and thus they may both be disrupted by faults. The expression table is part of the genome and it is identical in all cells. Therefore circuitry can be designed to recover the expression table from neighboring cells with a majority voting scheme. Hence we consider that such a fault can always be recovered, as long as at least one cell is intact in the system. The signal intensities however do differ in each cell, so they cannot be recovered in the same way. Nonetheless, there is a strong relationship between signal intensities in neighboring cells as the signal intensities decrease linearly with the Manhattan distance to the diffusers. Therefore signal intensities can be approximately reconstructed from the neighborhood.

To predict the signal in a cell from that of its neighbors the morphogenetic system is slightly modified. Instead of setting a signal by taking the value of the first initialized signal in a neighboring cell decremented by one, the diffusion rules assign the smallest value for which the signal gradient with all the initialized neighbors is -1, 0 or 1. This gives the same result as the original rules in fault-free conditions but allows better recovery in case of faults. Recovery is not always possible as there are cases where this rule does not predict correctly the signal intensities, in particular when faults occur on diffusers. Figure 6.10 illustrates what happens when the multi-cellular circuit is subject to three faults. One of the fault occurs on a diffuser and cannot be recovered correctly. The two other faults can however be recovered with the gradient rule given above.

Fault tolerance in the morphogenetic system is compared to a direct genetic encoding. The direct encoding is not capable of fault-tolerance, but it serves to highlight the benefits of the dynamics of the developmental system. A fault alters randomly the color of a pixel in the case of the direct encoding.

To ensure that fault-tolerance is provided by the developmental process and not by evolution, individuals are evolved in fault-free conditions before being tested. The best evolved phenotype of the Norwegian flag on the 64x64 array is used for testing. This pattern and size is selected because the fitness of the morphogenetic system and of the direct encoding are very similar and higher than the trivial solution consisting of exploiting only the frequency of colors (the fitness would be 0.37 if the morphogenetic system initialized all the cells with the most common color). The damage rate (percentage of faulty cells) is varied between 0% and 100%. The damage process is repeated 100 times for each damage rate on the same evolved, fault-free phenotype.

Figure 6.11 illustrates the results. While the direct encoding is subject to a linear decrease in fitness, the morphogenetic system shows a superior resistance to faults. The morphogenetic system benefits from the fact that signal intensities vary with continuity and can be easily reconstructed. Also, evolution assigns the most frequent color of the target pattern to the default cell type, which explains the fitness value of 0.37 with 100% of faults. A fitness of 0.25 is obtained in this case with the direct genetic encoding, because pixels are assigned randomly one of the four possible colours.

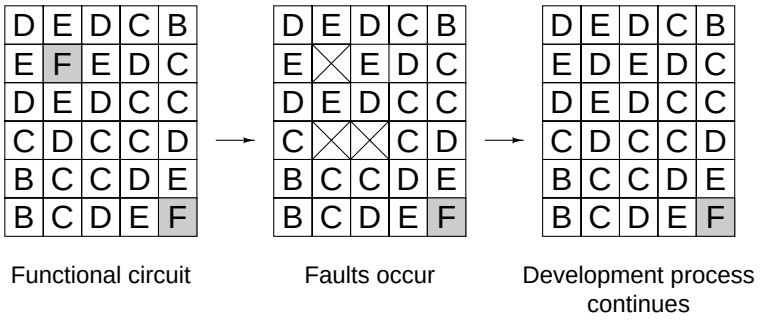

Figure 6.10: Example of faults occurring in the multi-cellular circuit. The numbers inside the cells indicate the intensity of the signal in hexadecimal. Cells in gray represent diffusers. The left array represents the evolved, functional, circuit. The middle array represents the signal in the cells once faults occur. Here faults occur in three cells (represented by the cross). As a consequence these cells are reset: the signal intensities are reset and the cell forgets whether it was a diffuser. The right array represents the recovered circuit after a developmental step. The signal in the top faulty cell is recovered from the neighbors which are E, E, E and E. The diffusion rules assign the smallest signal value for which the signal gradient with all the neighboring cells is -1, 0 or 1. Signals which respect this gradient rule are D (gradient of -1 with E), E (gradient of 0) or F (gradient of 1). The smallest signal is D which is thus the value recovered in the cell. In this case the recovery is incorrect because the signal was originally F. However in the two other faulty cells recovery is possible. The bottom left faulty cell sees the neighboring signals C, C and E. The signal respecting the gradient rule is the signal D which has a gradient of 1 with signal C, and -1 with signal E. No other signal intensity satisfies the gradient criteria. The bottom right faulty cell sees the neighboring signals C, C and D. The only signal that respect the gradient rule is signal C which has a gradient of 0 with neighboring signal C, and gradient -1 with signal D.

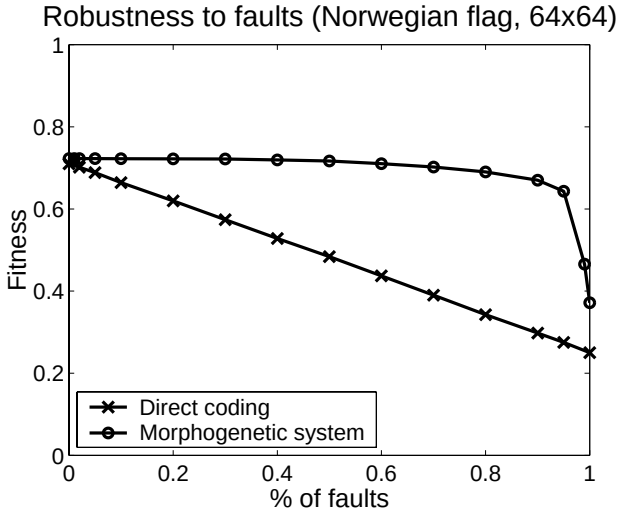

Figure 6.11: Fitness obtained on the *Norwegian flag* (size 64x64) with the direct genetic encoding and with the morphogenetic system after recovery from different fault rates.

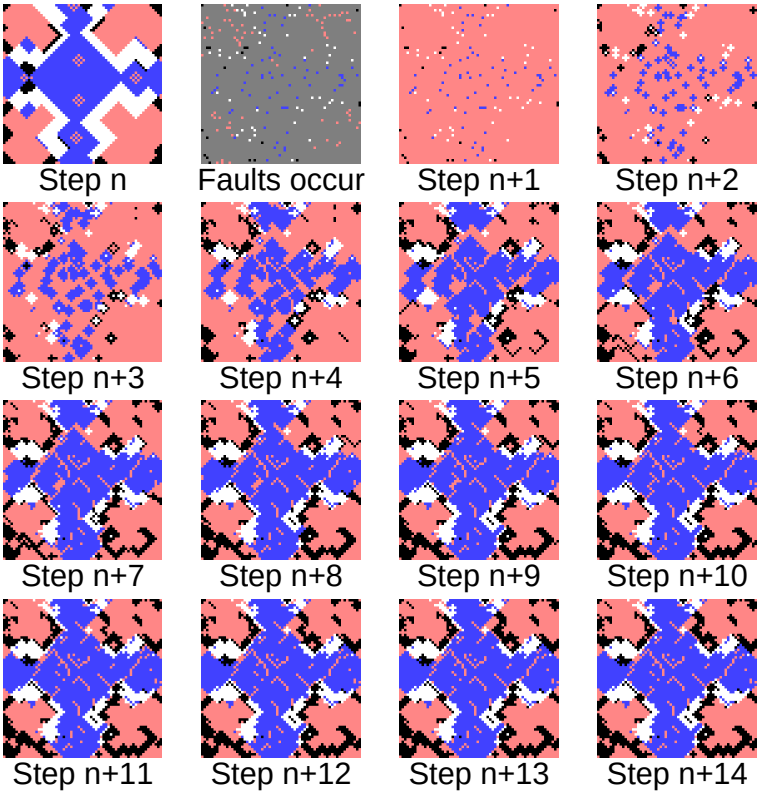

Figure 6.12: Recovery of the 64x64 *Norwegian flag* after 95% of faults. The first picture is the evolved pattern. The second shows the pattern after damage. The remaining pictures illustrate the pattern after each additional developmental step.

Figure 6.12 illustrates the recovery after a fault in the case of the evolved 64x64 *Norwegian flag* (95% of faults). The first picture shows the evolved pattern, the second shows the pattern after the faults occur, and the remaining pictures are snapshot of the pattern after each additional developmental step.

Figure 6.13 illustrates the 64x64 *Norwegian flag* recovered from different fault rates. The recovered pattern is very similar to the original one up to high fault rates.

6.6 Analysis

In this section we investigate the performance of the morphogenetic system at generating specific 2D phenotypes in function of the number of diffusers and the number of signal types. We then compare one morphological characteristic of phenotypes obtained with the morphogenetic system and the direct encoding, which is the size of connected areas of identical colors. Finally we do a preliminary analysis of the fitness landscape generated by the morphogenetic system. An analysis of the influence of the parameters of the morphogenetic system on the phenotypic complexity is given in appendix C.

Unless otherwise noted the fitness function, target patterns, genetic algorithm parameters and morphogenetic system parameters are the same as those used in section 6.3 and

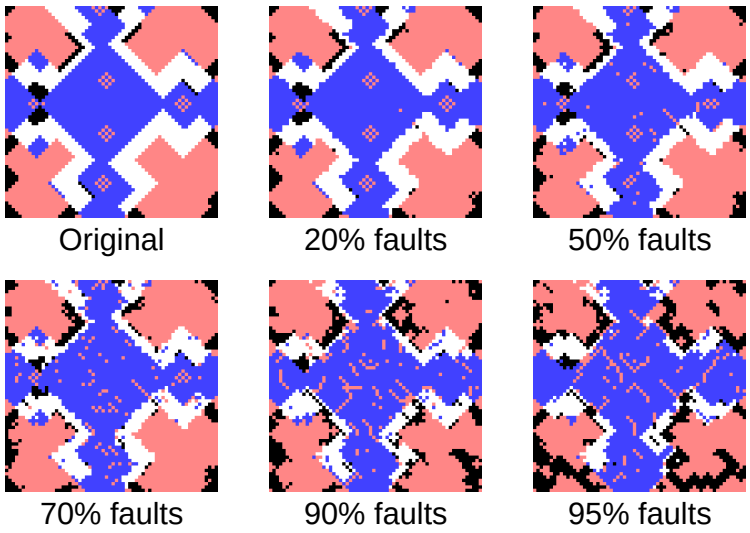

Figure 6.13: Patterns recovered by development with different fault rates in the case of the evolved 64x64 *Norwegian flag*.

6.4.

6.6.1 Number of diffusers

Diffusers affect a limited area of the phenotype around their position (a 31x31 diamond centered on the diffuser). Therefore larger phenotypes tend to require more diffusers so that all the cells of the system have the chance to receive the required signal intensities to express the correct functionality, as illustrated in figure 6.14. Results presented in the figure are 10-run averages for all the pattern sizes, except for the 64x64 pattern for which the results are 5-run averages. In the latter case further runs may be required to ensure statistical significance.

The influence of the number of diffusers in relation with the phenotype size is especially visible in the case of the checkerboard pattern: with a single diffuser the maximum fitness is reached for patterns up to size of 16x16. With larger patterns the fitness decreases because diffusers do not manage to set the signals in all the cells.

The combination of signals allows the expression of complex or irregular patterns. With no diffusers or when sufficiently far from them, cells all express the same default functionality (i.e. signal intensities are all to the default value). With more diffusers more complex phenotypes can be expressed. This is evidenced in the case of the *mixed 2*, the *CA-generated* and the *Norwegian flag* patterns: by increasing the number of diffusers the phenotypes generated by the morphogenetic system match more closely the target structures.

Although adding more diffusers increases the size of the genetic string, the morphogenetic system does not seem to show the degradation of performance typically associated with larger search spaces.

In summary the required number of diffusers should be selected according to the phe-

notype size and the structure of the target patterns, but selecting a high number of diffusers seems to be a safe choice in these experiments.

6.6.2 Number of signal types

In the previous sections the morphogenetic system used four types of signals. The number of signal types affects the maximum number of functionalities which can be expressed by the morphogenetic system. With n signal types, each encoded on m bits, there are $2^{n \cdot m}$ possible states of signals in cells and therefore up to $2^{n \cdot m}$ different functions can be expressed.

Empirical tests showed that 4 signal types, which are encoded on 4 bits, are adequate for the applications described in this chapter. This corresponds to a maximum of $2^{4 \cdot 4} = 65536$ different functionalities which is more than the number of functionalities which are actually used. Therefore several combinations of signals may express the same cell functionalities.

Adding more signal types has little influence on the performance of the morphogenetic coding in these experiments, as illustrated by figure 6.15 which shows the fitness scores obtained after evolving the *CA-generated* pattern with different number of signal types. Results presented in the figure are 5-run averages.[3] Results obtained with the evolution of other target patterns are similar.

While the minimum number of signal types should allow the expression of all the functionalities required in the system, selecting a too high number of signal types does not seem to affect the performance of the morphogenetic system, even if this increases the size of the search space. There is however an influence on the phenotypic complexity of the number of signals when the number of diffusers is changed. This aspect is discussed in appendix C.

6.6.3 Morphological characteristics

The morphogenetic system tends to generate random phenotypes that display larger connected areas of cells of identical functionalities than a direct encoding (figure 6.16).

We assess this by generating 100 random 8x8 binary phenotypes with a direct genetic encoding and the morphogenetic system (16 diffusers, 2 functionalities, 4 signal types). For each phenotype, we detect connected areas of cells of identical functionalities[4] and we measure the size of all the connected areas. From this we estimate the probability for a cell to belong to a connected area in function of the size of the connected area (figure 6.17).

[3] Although more runs may be required to ensure statistical significance, the standard deviation on 5 runs of the maximum fitness scores reached at generation 2000 is very small (< 0.007). This tends to indicate that the results are likely to be similar with more runs.

[4] We consider two cells of identical color to be connected if they touch along the horizontal or vertical axis.

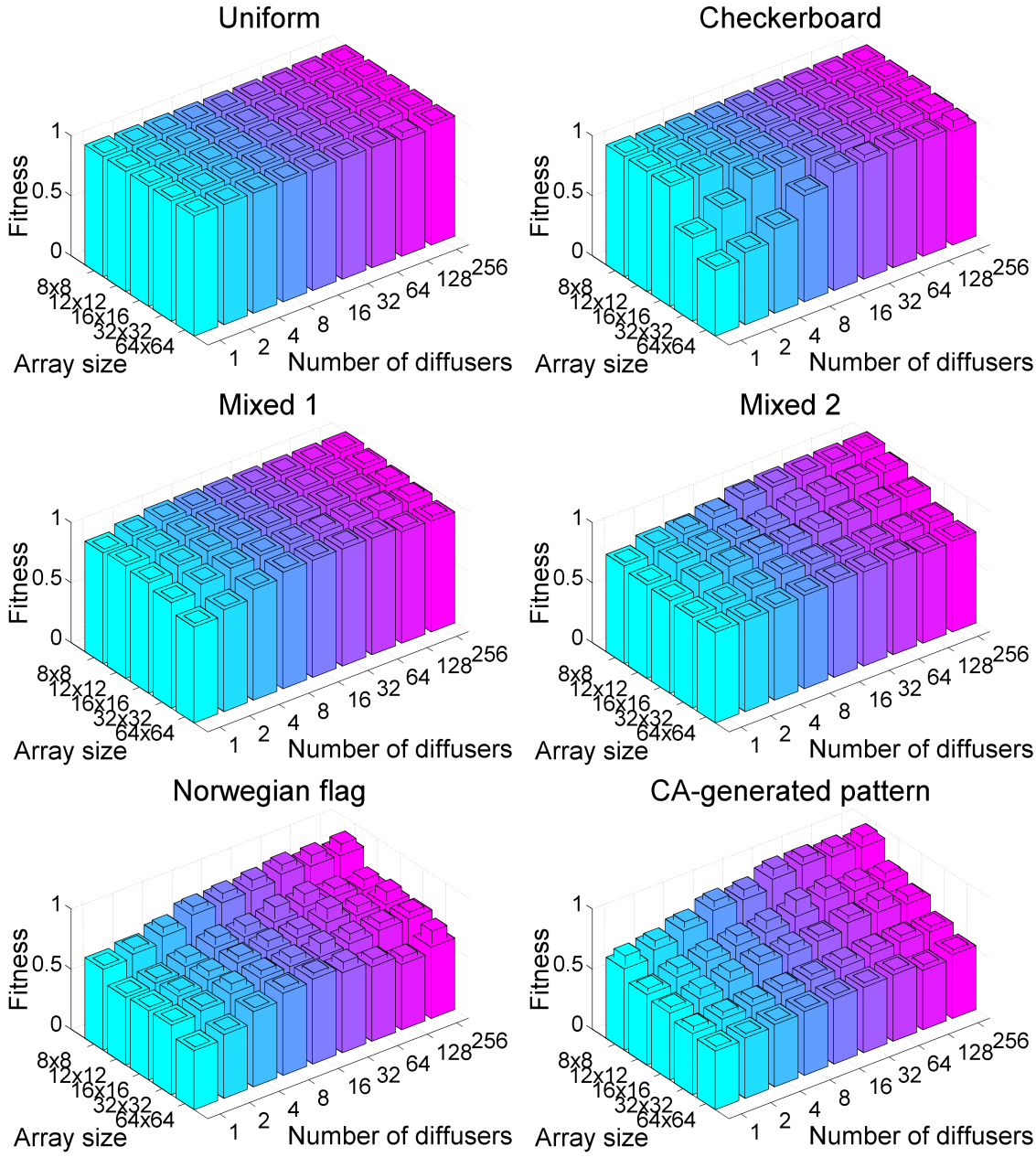

Figure 6.14: Influence of the number of diffusers on the maximum and average fitness obtained after 2000 generations. The number of signal types is 4. The bars represent the average and the maximum of the maximum fitness obtained in 5 runs. With larger or more complex phenotypes, increasing the number of diffusers allows to increase the maximum fitness score.

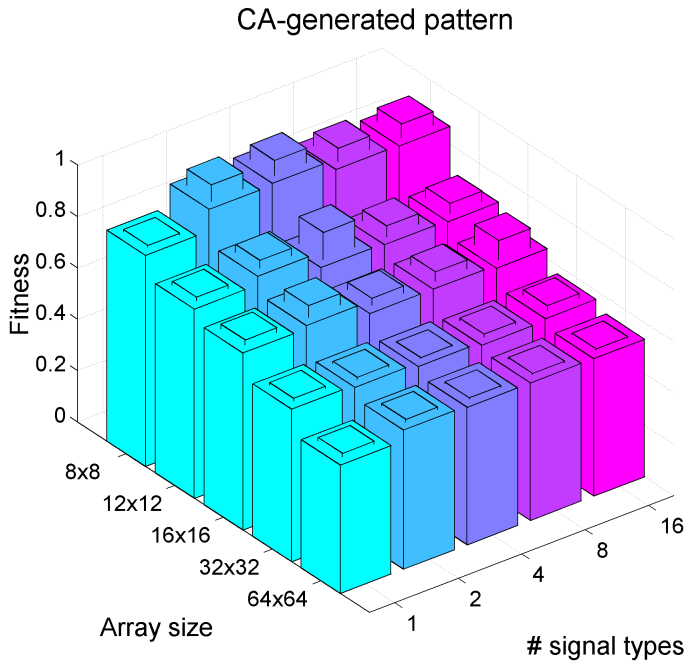

Figure 6.15: Influence of the number of signals on the fitness which is obtained after 2000 generations when evolving the *CA-generated* pattern. The bars represent the average and the maximum of the maximum fitness obtained in 5 runs. The number of diffusers is always 16. Note that changing the number of type of signals has little influence on the fitness which is obtained.

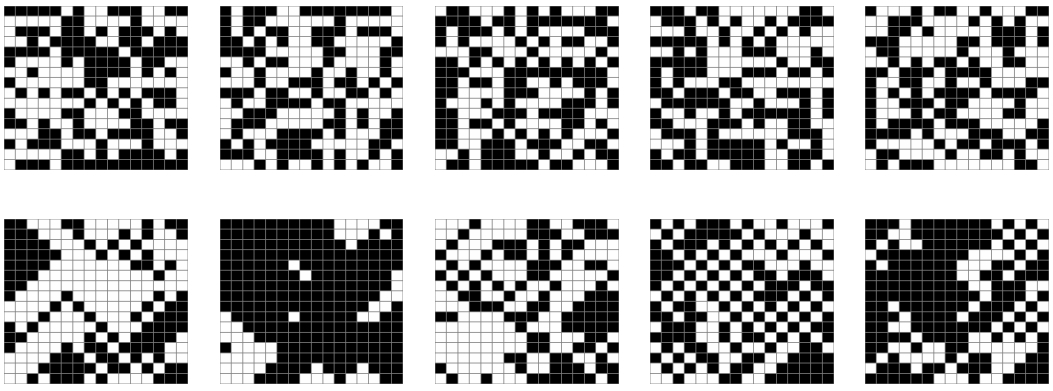

Figure 6.16: Randomly generated 16x16 phenotypes with the direct encoding (top line) and the morphogenetic system (bottom line). Each cell can take one of two functionalities (black or white cell). The direct genetic encoding assigns one bit per cell. The morphogenetic system uses a family of 2 functionalities, 16 diffusers, and 4 signal types. The genetic string of the multi-cellular system is randomly generated in both case. Phenotypes generated with the morphogenetic system tend to display larger connected areas of cells of identical functionalities (colors).

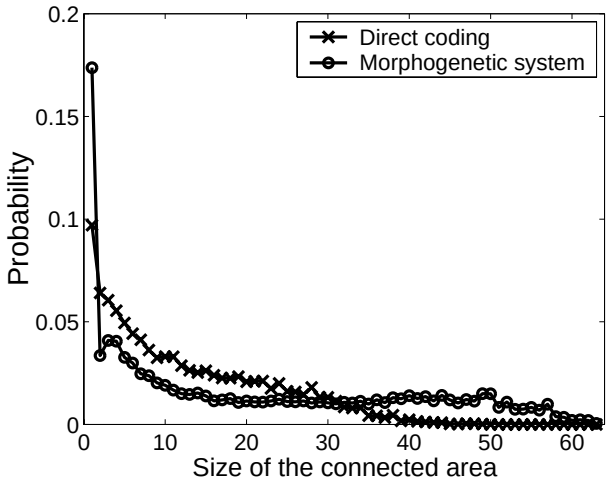

Figure 6.17: Probability for a cell to belong to a connected area of cells of identical functionalities in function of the size of the area. With the morphogenetic system, cells tend to belong more to areas of size 1 than with the direct coding; and cells also belong more to areas of larger size (30 and more cells), at the expense of areas of size in the range 2 to 20.

In comparison to the direct genetic encoding, cells tend to belong more to larger connected areas and less to smaller ones with the morphogenetic system. By generating large connected areas of identical cells the morphogenetic system may easily achieve a fitness which is equal to the proportion of the most common cell color in the circuit. This may explain why the morphogenetic system evolved very easily phenotypes to resemble e.g. the *uniform* pattern in section 6.3.

6.6.4 Fitness landscape

The ruggedness of the fitness landscape is often linked to the evolutionary search difficulty [147]. The ruggedness is investigated by performing random walks using the mutation operator starting from points of maximum fitness on the evolved 64x64 *Norwegian flag*. The genetic encoding and mutation rate are the same as used during evolution in section 6.4. Taking the best evolved individuals, 3000 random walks of 30 steps are performed with the morphogenetic system and the direct genetic encoding. Figure 6.18 shows that the fitness drops faster with the morphogenetic system when moving away from a point of maximum fitness, which seems to imply a more rugged fitness landscape in the case of the morphogenetic system. The better performance of the morphogenetic system observed in previous experiments thus does not seem to come from a smoother fitness landscape. This may imply that the morphogenetic system benefits essentially from its smaller search space size in comparison to the direct genetic encoding.

We considered the ruggedness according to the mutation operator, however one-point crossover is also used during evolutionary search. Therefore the ruggedness measured here is an approximation of the effective ruggedness seen during evolutionary search. Ruggedness may be further investigated by considering an alternative view where the search space is a connected graph whose vertices are crossed through the action of the

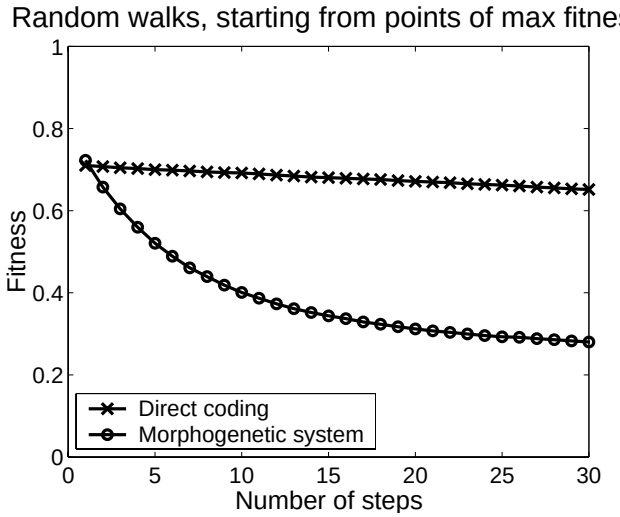

Figure 6.18: Comparison of random walks performed on the evolved 64x64 *Norwegian flag*. The fitness drops faster with the morphogenetic system when moving away from points of maximum fitness. This seems to indicate that the fitness landscape is more rugged with the morphogenetic system.

search operators. Each operator defines its landscape and thus an evolutionary algorithm makes transitions on a mutation, a crossover and a selection landscape [77]. This view may allow to measure ruggedness according to other search operators (e.g. one-point crossover). It is however more difficult to interpret [147] and for this reason it was not considered here.

6.7 Hardware implementation

In this section we describe how the morphogenetic system fits in the cell architecture introduced in chapter 4, and we implement the morphogenetic system on the POEtic chip. The hardware implementation behaves identically to the software simulations and therefore it achieves the same performance in terms of evolvability and scalability as indicated in previous sections. The implementation however does not use the fault tolerant rules of the morphogenetic system (section 6.5), since these translate in a larger implementation. We do not consider the implementation of any cell functionality; this is done in chapter 7.

In chapter 4 we introduced the notion of cells composed of three layers: the phenotype layer which is the functional part of the cell, the mapping layer where development is implemented, and the genotype layer which contains the genetic code of the entire multicellular circuit (figure 6.19 left).

The morphogenetic system maps the genetic string of the circuit into the circuit functionality. It therefore belongs to the mapping layer of the cells. We refer to the hardware implementation of the morphogenetic system in a cell as a *morphogenetic element*.

The morphogenetic element is illustrated in figure 6.19 (right). It is connected to its four immediate neighbors to receive their signal intensities (inputs In 0 to In 3 in the

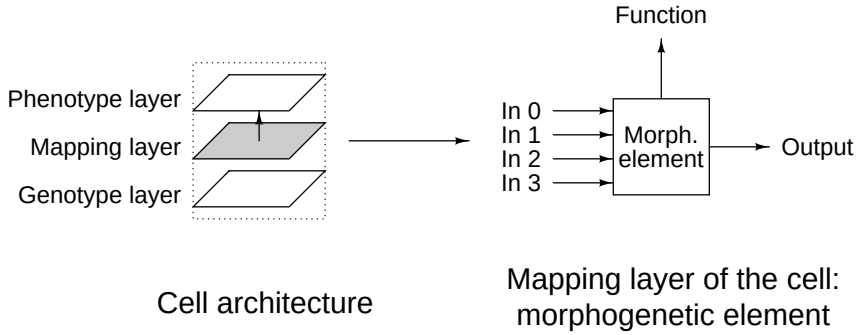

Figure 6.19: Cells of POEtic circuits are composed of three layers: phenotype, mapping and genotype (left). The morphogenetic system maps the genetic string of the circuit into a cell functionality and belongs to the mapping layer of the cell. The hardware implementation of the morphogenetic system in the cell is referred to as the morphogenetic element (right). The morphogenetic element has inputs (In 0 to In 3) and an output to exchange signal intensities with its neighbors. It has a function output which indicates what is the functionality that the phenotype layer of the cell must take within the family of predefined functionalities. In the three-layered structure this function output is sent to the phenotype layer of the cell, as indicated on the left by the arrow pointing upwards.

figure). It has one output, which is connected to its neighbors and through which the morphogenetic element sends its own signal intensities. Finally the morphogenetic element has a function output, which indicates to the phenotype layer of the cell which functionality it must take within the predefined family of functionalities required for evolution with the morphogenetic system.

In this implementation the morphogenetic element is designed to handle 4 entries in the expression table (i.e. there may be a maximum of 4 different cell functionalities) and the number of signal types is 4. This corresponds to the settings used earlier in this chapter to evolve structures of differentiated cells (e.g. in section 6.3).

The implementation on the POEtic chip requires to configure the molecules of the chip to execute the algorithm of the morphogenetic system described in section 6.2. The result of the implementation is illustrated in figure 6.20 which shows the molecules of the POEtic chip that implement the morphogenetic element. In total 56 molecules are used. The hardware implementation allows relatively fast development since a developmental step requires 256 clock cycles and therefore complete development, which requires 16 developmental steps, takes 4096 clock cycles. In contrast to a software implementation, this time is independent of the size of the multi-cellular circuit. The implementation is detailed in appendix D.

6.8 Discussion

Combining a genetic encoding with a developmental system in an evolutionary system may provide dynamic reorganization capabilities to POEtic circuits. For instance reorganization may occur when the circuit is expanded with new cells, when the environment

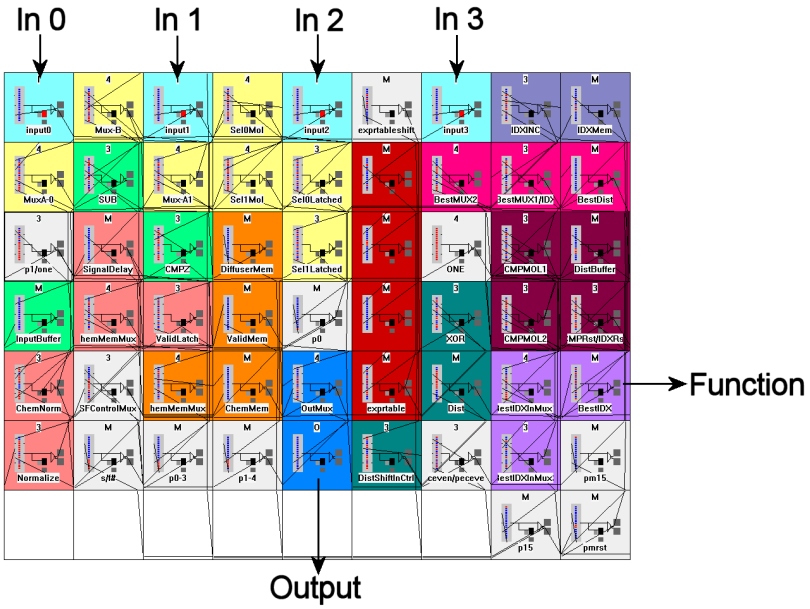

Figure 6.20: Implementation of the morphogenetic element with the molecules of the POEtic chip. 56 molecules are required to implement the morphogenetic element. The morphogenetic element receives the signals of its north, east, south and west neighbors using the 4 inputs which are connected using the dynamic routing mechanism of the POEtic chip. The output sends the signal values to neighboring morphogenetic elements in the same way. The function output indicates the functionality that the cell must take.

changes, or when sensors or actuators are connected to the circuit. Direct genetic encodings are not suited for this, because the number of cells in the system must be known in advance and the genotype to phenotype mapping is static. With the morphogenetic system, a cell newly connected to the circuit would differentiate according to the signals emitted from its neighbors. Dynamic reorganization may be mediated by "chemicals" (i.e. signals produced by diffusers) injected in the multi-cellular circuit from the environment or special cells (e.g. sensors or actuators). However, the capacity of the morphogenetic system to provide a meaningful functionality to cells dynamically added to the circuit, together with dynamic reorganization of POEtic circuits at the level of the morphogenetic system, have not yet been investigated.

According to Streichert et al. multi-cellular systems that grow and differentiate according to a developmental program may rely on *exogenous* structuring factors (i.e. structuring factors or chemicals are provided to the multi-cellular system from the outside), or they may emerge purely from *endogenous* self-organization (i.e. the "cell program" controls the growth and differentiation without external intervention) [155]. The morphogenetic system is based on exogenous structuring factors which are the diffusers. Diffusers are similar to *morphogens*, the chemicals which convey positional information in the development of biological organisms [188]. This approach has one disadvantage which is that the number of diffusers must be increased when the size or complexity of the phenotype increases (see section 6.6), which in turns increases the size of the genetic

string. However this approach has the advantage that diffusers can be placed locally in areas of higher complexity in the phenotype. An endogenous approach on the other hand needs to encode the entire phenotype in a cell program of limited size, and this may also place a limit on the size or complexity of the phenotypes which can be encoded in this way.

The morphogenetic system partly adapts the expression of cell types to the phenotype statistics. This occurs through evolution of the expression table. The expression table assigns a functionality to all the cells, even if they have no signals yet initialized (e.g. at the beginning of the development, or when a cell is too far from any diffuser). Evolution exploits this to assign the most common cell type to cells with uninitialized signals. This is evident in the evolution of phenotypes to resemble specific 2D patterns. Evolution tends to assign the most common color (e.g. the background of the *Norwegian flag*) to uninitialized cells and further refinement is modulated by the diffusers. The morphogenetic system benefits from this when evolving patterns, since in the first generation it may already find individuals whose fitness is close to the proportion of the most common cell type (direct encodings achieve a fitness close to $\frac{1}{n}$ where n is the number of cell types). The morphogenetic system also benefits from this in the fault tolerance task, where cells express by default the most common color of the pattern even at high fault rates.

The morphogenetic system employs hard-coded signaling and expression mechanisms that tend to generate diamond-shaped patterns (figure 6.7). This may limit the maximum fitness that can be achieved by the morphogenetic system. However these hard-coded mechanisms are also one of the reason for the simplicity of the morphogenetic system. More general developmental systems use an evolved cell program that controls the production of chemicals and the differentiation of cells [32, 117], but these may translate in more complex implementations.

The morphogenetic system has been optimized for low computational complexity. It does not use multiplications, divisions, nor floating point operations. The signaling and expression mechanisms can be implemented with increments, decrements, logic operations and comparisons. In a software implementation, development and evaluation of the fitness of an 8x8 phenotype with a family of 4 functionalities (experiments of section 6.3) proceeds at the speed of approximately 10000 individuals/sec on a 2.08GHz AMD Athlon XP CPU. The speed scales down with the size of the phenotypes: phenotypes of size 64x64 are evaluated at 90 individuals/second. As a reference, a direct encoding evaluates 60000 individuals/sec with an 8x8 phenotype.

In comparison, the artificial embryogeny of Federici [35], which relies on a neural network controlling the growth of a multi-cellular system, achieves approximately 1000 individuals/sec in the same experiment with an 8x8 phenotype [36]. While that developmental system is slower than the morphogenetic system, it is more biologically plausible as development starts from a single cell and cell growth or death is physically located (i.e. cell duplication “pushes” neighboring cells). Still, on a common task, the scalability and fault-tolerance of the artificial embryogeny and of the morphogenetic system showed to be similar [134]. Therefore the morphogenetic system fulfills its objective of low computational complexity and at the same time it performs well compared to a more complex developmental system.

The low computational complexity of the morphogenetic system allows a compact hardware implementation on the POEtic chip. According to the terminology introduced in chapter 3, the hardware implementation of the morphogenetic system is an intrinsic, online and cellular implementation. The cellular implementation of the morphogenetic system is fast and possibly more robust than centralized sequential implementations such as those done in specialized hardware or in software. In particular, development time is constant regardless of the number of cells in the system. Online development is however exploited only in the software model where we showed that it may bring fault-tolerance. The hardware implementation does not take advantage of online development since the fault-tolerant diffusion rules are not yet used. The hardware applicability of these rules remains the object of future work.

The results obtained in software regarding fault-tolerance bear some similarities with those shown by Miller [117] and Federici [134]. The major difference is that fault-tolerance is provided to the morphogenetic system by *design* (i.e. the fault-tolerant diffusion rules are designed to approximate the signal intensities in a cell based on the signal intensities of neighboring cells), whereas in Miller's and Federici's work fault-tolerance is *emergent*. In the latter case the evolved cell program controlling development is capable of growing cells to replace faulty ones without explicit evolutionary pressure to this end.

The morphogenetic system currently does not allow variable diffusion ranges, and it relies on predefined cell functionalities which are not evolved. Future work may consider improvements in these aspects. Variable diffusion ranges may allow more efficient evolution of phenotypic structures by letting long range signals shape large structures while signals of shorter range take care of local details. The cell functionalities may be encoded in the genetic string of the circuit so that new functionalities can be created or modified by the evolutionary process. We showed that the parameters of the morphogenetic system (number of diffusers or number of signal types) have an influence on the fitness which can be obtained. Since selecting the appropriate parameters may be a difficult task, evolution may be used to adapt the number of diffusers or signal types to the size and complexity of the target phenotype.

6.9 Summary

In this chapter we introduced a minimalist developmental system and genetic encoding suited for multi-cellular systems such as POEtic circuits. We called this system the morphogenetic system. The morphogenetic system is inspired by gene expression and cellular differentiation. It can be implemented in a fully distributed way, and it assumes only local communication between immediate neighboring cells, which allows it to be applied to circuits regardless of their size. In addition it achieves low computational complexity which makes it suited for compact hardware implementation. In particular it does not use multiplications, divisions, nor floating point operations.

We found that the morphogenetic system was capable of evolving patterns of diversified structures with relatively high fitness scores, and that in terms of fitness it scaled better to larger phenotypes than a direct genetic encoding on the patterns which were

considered. According to the classification introduced in chapter 3, the morphogenetic system is capable of *online* development. We showed in software that online development allows to recover from faults up to high fault rates by exploiting the dynamics of the developmental system.

Finally we showed that the morphogenetic system is well suited for hardware implementation on the POEtic chip. It uses very few resources (only 56 molecules), and allows development at relatively high speed in constant time (4096 clock cycles), regardless of the size of the circuit.

In conclusion we demonstrated that the morphogenetic system could be used to evolve structures of differentiated cells, although we did not yet consider cells with a real functionality other than a color. In the following chapters we will demonstrate that the morphogenetic system can be used to evolve functional multi-cellular circuits for various applications.

7 Evolutionary morphogenesis of spiking networks

Abstract[1]

In the previous chapter we introduced the morphogenetic system and we showed that it could evolve structures of differentiated cells. Yet these structures did not have any functionality. In this chapter we demonstrate that the morphogenetic system can be used to evolve functional circuits. We evolve multi-cellular spiking neural networks to perform pattern recognition and to control a mobile robot in a task of obstacle avoidance. The multi-cellular spiking controller is implemented on a FPGA embedded on the mobile robot. We find that the morphogenetic system outperforms a direct genetic encoding when evolving these multi-cellular spiking neural networks.

7.1 Introduction

In the previous chapter we introduced the morphogenetic system and we showed that it could evolve structures of differentiated cells. Yet these structures did not have any functionality. The objective of this chapter is to demonstrate that the morphogenetic system can be used to evolve functional circuits.

In chapter 2 we evidenced the problem of scalability in evolvable hardware and we suggested to evolve circuits composed of high-level functional blocks that can process analog values, since analog circuits seem to have smoother fitness landscapes.

For this reason we consider here multi-cellular circuits whose cells implement spiking neurons. Spiking neurons model biological neurons that exchange information by short binary events called action potentials, or spikes, which are sent through their axon to connected neurons [49, 105].

Spiking neurons are high-level functional blocks in comparison to the elementary logic gates often used in evolvable hardware. Furthermore they exchange binary spikes, which match digital hardware, but they may encode analog information in the temporal

[1] Part of this work was published in [135, 136]. Stéphane Hofmann contributed by developing the FPGA module for the Khepera robot during his master project under my supervision.

patterns of spikes. Spiking neurons are suited for efficient analog implementations [74], or fast digital implementations [61, 143], and they may be optimized for use with limited resources [43, 171]. Furthermore their computational power is at least identical to that of multilayer perceptrons and sigmoidal networks and in some cases greater [106, 141].

Spiking neurons have been used previously as controllers in evolutionary robotics, e.g. to perform vision-based obstacle avoidance [40, 43] and phototaxis [29].

In this chapter we thus consider the evolutionary morphogenesis of multi-cellular circuits composed of spiking neurons. We evolve these circuits to perform pattern recognition, and to control the navigation of a robot in a task of obstacle avoidance. Evolution with the morphogenetic system is compared with a direct genetic encoding.

The experiments described in this chapter are done in a software simulation of the multi-cellular circuit. The robot controller is however later implemented in hardware on a FPGA. This demonstrates that the spiking neuron model is suited for multi-cellular hardware implementation. The hardware implementation behaves in the same way as the software simulation. Therefore the controllers evolved in simulation can be transferred seamlessly to the hardware controller.

This chapter is organized as follows. Section 7.2 introduces the neural model which is used in subsequent applications. Section 7.3 describes the evolutionary morphogenesis of spiking neural networks to do pattern recogition. Section 7.4 describes the evolutionary morphogenesis of the robot controller. Section 7.5 describes the the hardware implementation of the robot controller. The results are discussed in section 7.6 before concluding in section 7.7.

7.2 Evolution of multi-cellular spiking neural networks

In order to evolve multi-cellular spiking neural networks with the morphogenetic system, we assume that a cell of the multi-cellular circuit can implement the functionality of a spiking neuron.

The spiking neuron model used in the following experiments is a discrete-time, integrate-and-fire model with leakage and a refractory period. Each neuron has weighted inputs (+2 or -2 depending on whether the presynaptic neuron is excitatory or inhibitory) from connected neurons, according to the connectivity patterns shown in figure 7.1. It has one more connection from an external input, e.g. to connect a sensor, with fixed weight of +10. The neuron integrates the incoming spikes in the membrane potential, according to the weights of the connections. Once the membrane potential reaches a threshold (fixed to 4), the neuron fires (emits a spike), resets its membrane potential to 0 and enters a refractory period where it does not integrate incoming spikes for one time step. After the integration phase and if the neuron has not fired, leakage decrements the membrane potential by 1 (or increments it if the potential is below 0), so that the asymptotic potential is 0. Figure 7.2 shows the effect of incoming spikes on the membrane potential.

The main characteristic of this model compared to others is that few computations are needed to update the neuron state at each network step (e.g. no multiplications or exponentials). This neural model has been previously implemented on a microcontroller with

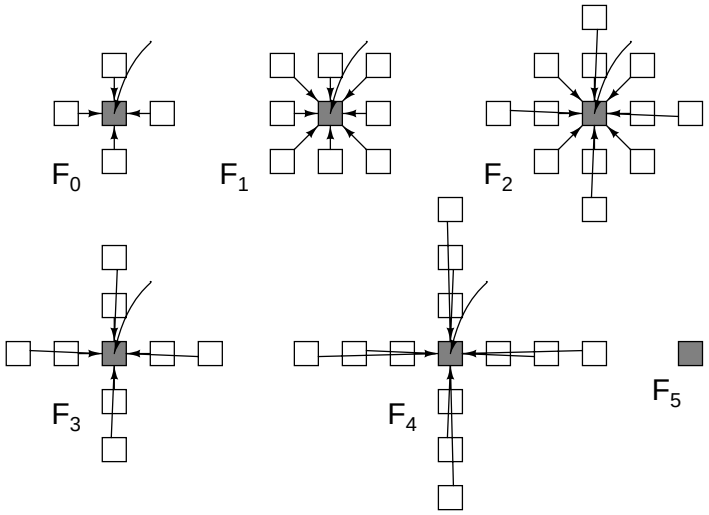

Figure 7.1: A family composed of 12 functions is used when evolving neural networks. The functions are spiking neurons with different connectivities (the 6 connectivities shown in the figure) and with either excitatory or inhibitory characteristics. Each cell of the multi-cellular circuit implements a single neuron, shown in gray. It receives inputs from neighboring neurons (outlined), which are implemented in neighboring cells. Each neuron has an extra external input (curved arrow). Neuron F_5 is equivalent to a void cell. At the boundary of the cellular array the connectivity is truncated (no periodic boundary condition).

only 60 bytes of RAM [43] and it is well suited for compact hardware implementation (see section 7.5).

The family of functionalities required by the morphogenetic system consists of neurons with different patterns of incoming connections, and either excitatory or inhibitory characteristics. The interconnections among spiking neurons are considered as part of the cell functionalities. The family of functionalities is illustrated in figure 7.1. In these experiments there are 12 different functionalities that can describe a large number of complex and recurrent neural architectures, depending on the circuit size and genetic code. The expression table of the morphogenetic system therefore contains 12 entries.

Table 7.1 indicates the parameters of the morphogenetic system and of the genetic algorithm. The size of the genetic string with the morphogenetic system is 320 bits: 12

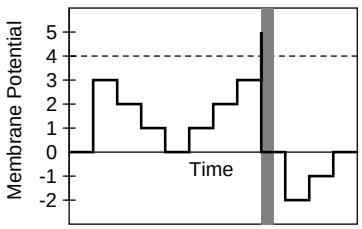

Figure 7.2: Effect of incoming spikes on the membrane potential. Each time a spike is received the membrane potential is increased. When a threshold (dashed line) is reached the neuron emits a spike (gray column) and the potential is reset to 0.

GA parameters	Genetic coding
Population: 50	Coding: morphogenetic system
Crossover: 20%	Diffusers: 16
Mutation: 0.5%	Expression table: 12 entries
Selection: rank (15 best selected)	Chromosome size: 320 bits
Elitism: 5 individuals are copied unchanged	

Table 7.1: Parameters of the genetic algorithm and of the morphogenetic system.

entries in the expression table * 16 bits + 16 diffusers * 8 bits (6 bits for the coordinates and 2 bits for the type of diffuser).

In the following sections evolution with the morphogenetic system is compared with a direct genetic encoding. The parameters of the genetic algorithm are the same as with the morphogenetic system. The functionality of each cells is encoded by 4 bits, and therefore the size of the genetic string with the direct encoding is 256 bits (4 bits/cells * 64 cells).

7.3 Pattern recognition

The circuit of 8x8 neurons illustrated in figure 7.3 is evolved to recognize characters (any other pattern could be used) using two training sets: one set contains corrupted versions of two characters, the other set contains random patterns which have on average the same percentage of black pixels as the characters. The circuit must tell whether the current pattern is one of the two characters or not by raising the firing rate of one specific neuron in the circuit. The input pattern is applied on a subset of the neurons through their external input. Each neuron receives one pixel of the pattern: if the pixel is black it receives one spike every two time steps, otherwise it receives no spikes. The network is run for 100 time steps with the input applied to it, afterward the activity of the output neuron is read. The activity (number of spikes) of that neuron indicates whether the input pattern is a character (threshold=50% of maximum spike number).

Bottom of figure 7.3 shows the training set for the recognition of the characters A and C (noted as A+C). The upper line shows the subset of patterns to recognize. The second line contains random patterns that must be rejected. The fitness of the network is evaluated by presenting successively all the patterns of the training set. The fitness score is the number of times the network correctly classifies the input pattern. The maximum normalized fitness is 1, corresponding to the successful classification of the 20 patterns in the training set. The experiments are performed with four different training sets, for the recognition of characters A+B, A+C, A+D and A+E.

The circuit is evolved one hundred times for each of the four training sets. Figure 7.4 shows the evolution of the fitness. The morphogenetic system outperforms the direct coding, both when comparing the maximum fitness reached at a given generation and when comparing the number of runs that have reached the maximum fitness after 50 generations. Table 7.2 reports the number of runs (on the 100 runs performed) where the maximum fitness is reached. Averaged over the four training sets, runs reaching the maximum fitness with the morphogenetic encoding are more than the double than with

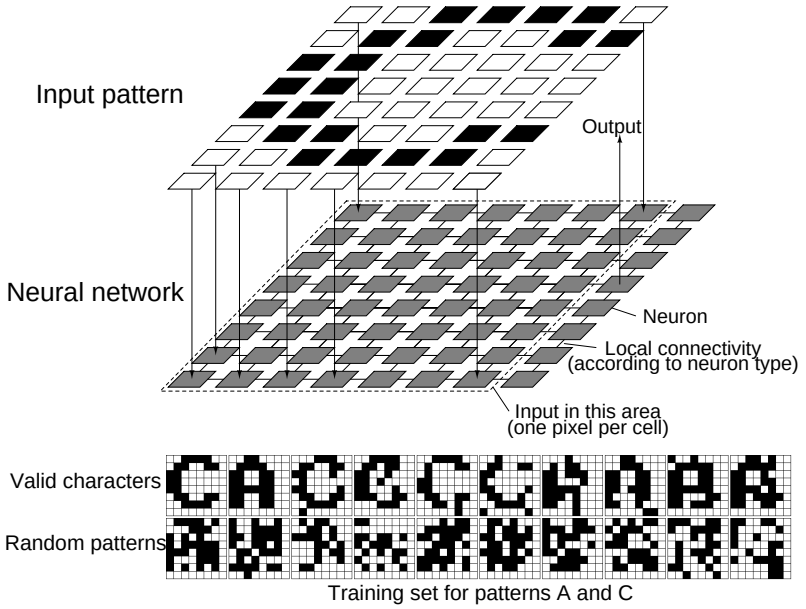

Figure 7.3: Top: spiking network doing pattern recognition. The input pattern is applied on an array of 7x8 neurons on the left of the network. Each neuron receives the input from one pixel of the pattern. The activity of the output neuron indicates whether a character is recognized. Bottom: training set for the recognition of the patterns A and C. It is composed of 20 patterns. The upper 10 are the patterns to recognize which are the letters A and C. The lower 10 are random patterns that the network must reject.

the direct encoding.

The capacity of the evolved networks to generalize to unseen examples is tested with two sets of generalization patterns each composed of 125 patterns. The first set contains corrupted versions of the two characters that where used during training. The other set contains 4 corrupted versions of all the characters in the alphabet with the exception of the two that must be recognized (i.e. in total $4 \cdot 24 = 96$ patterns) and 29 random patterns which have on average the same percentage of black pixels as the characters. The generalization performance is measured by presenting each pattern successively to a circuit and looking whether it correctly classifies the pattern: patterns of the first set should be detected as being the characters used during training, whereas patterns of the second training set should be rejected. The generalization performance is measured on the best evolved circuits of each run and averaged (see table 7.3). Results indicate that circuits evolved with the direct encoding and the morphogenetic system are capable of a moderate degree of generalization.

7.4 Robot controller

A spiking network is evolved as a controller for the two-wheel differential drive miniature mobile robot Khepera [120]. The objective is to navigate while avoiding obstacles using the information coming from the proximity sensors of the robot.

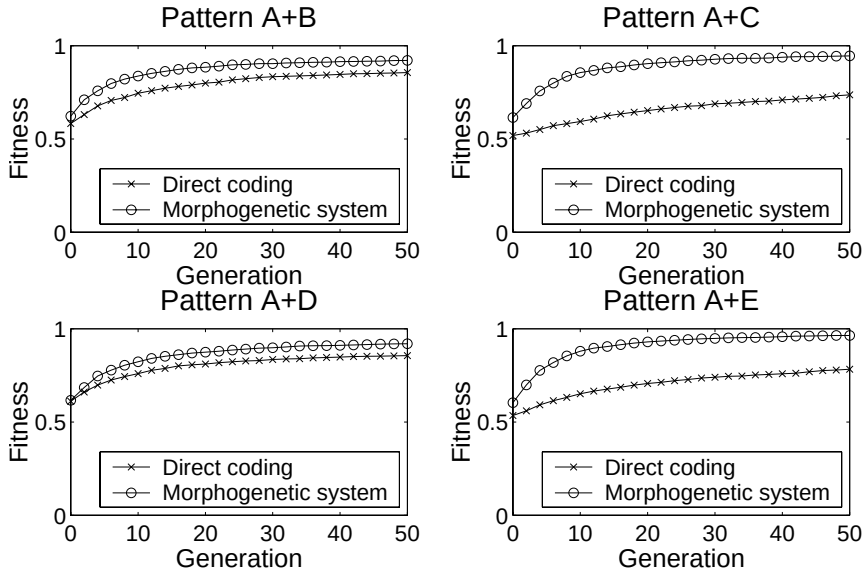

Figure 7.4: Evolution of the maximum fitness (average of 100 runs) on the four training sets of the pattern recognition task.

Training	Direct	Morph.
A+B	18	27
A+C	8	34
A+D	19	20
A+E	14	54
Total (max 400):	59	135

Table 7.2: Number of runs, out of 100 performed with each training set, reaching the maximum fitness.

Generalization	Direct	Morph.
A+B	0.67	0.67
A+C	0.56	0.65
A+D	0.67	0.69
A+E	0.60	0.64
Overall:	0.62	0.66

Table 7.3: Proportion of patterns that are successfully classified in the generalization set by the best evolved circuits (average of the generalization performance of the best evolved circuits of each of the 100 runs).

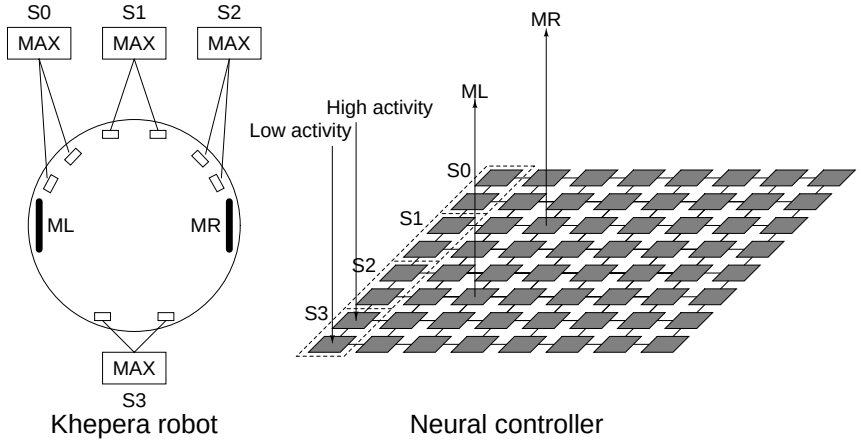

Figure 7.5: The Khepera robot (left) and the neural controller (right). The Khepera has 8 proximity sensors. They are grouped by two, taking value of the most active sensors, to have 4 sensory inputs S0 to S3. The network receives S0 to S3 as sensory inputs. Each input is mapped to two input neurons (low and high activity indicated in the figure). The neurons ML and MR control the speed of the wheels according to their activity.

Figure 7.5 illustrates the robot and the neural controller. There are eight input neurons organized as four sensory groups of two neurons (S0 to S3). These input neurons are connected to the proximity sensors of the robot. Each group has one "low activity" neuron and one "high activity" neuron. When farther than about 5 cm from the obstacles, none of the inputs are stimulated. Between 5 cm and about 1 cm the "low" activity input is stimulated. When closer both the "low" and "high" activity inputs are stimulated. A stimulated input receives a spike train of period 2 (one spike every two time steps). An input which is not stimulated receives a spike train of period 9. Noise is introduced by varying the period of the input spike trains: the period is randomly increased by one time step with a 50% probability.

Neurons ML and MR are used to set the speed of the wheels, which is inversely proportional to their activity. When the neuron does not fire the speed of the wheel is +80 mm/s. With maximum activity the speed is -80 mm/s. The speed scales linearly in between. This allows the robot to move forward when no obstacles are sensed and thus when there is potentially no activity in the network.

The robot has a sensory-motor period of 100ms. During that period, the network is updated 20 times. At the end of the sensory-motor period, the speed of the wheels is updated and the proximity sensors are read to compute the spike trains for the next sensory-motor cycle.

The spiking neuron model used here is the same as that used in the pattern recognition experiment, with the exception that the membrane potential of the neurons is not allowed to go below zero. This is a simplifications that allows a more efficient hardware implementation, yet its influence on the performance of evolved networks is minimal.

The fitness of the robot is measured on two tests of 30 seconds in a rectangular arena (40x65 cm). It is the average of the fitness computed at each sensory-motor step using the following equation [41]: $f = \overline{v} \cdot (1 - \Delta v) \cdot (1 - p)$, where $\overline{v}$ is the average speed of

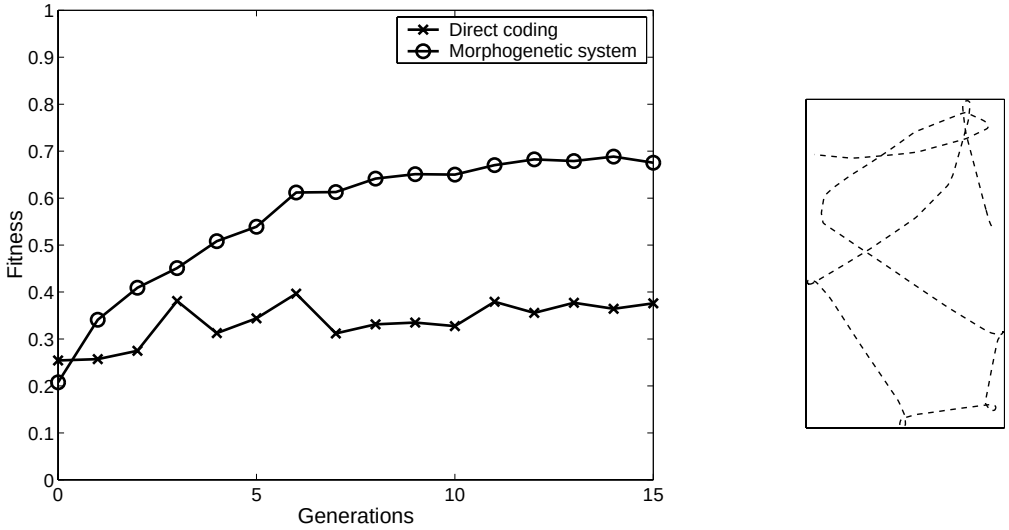

Figure 7.6: Left: Evolution of the best fitness in the obstacle avoidance task (average of 10 runs on a physical robot). Right: Typical trajectory of the robot in the arena.

the two wheels, Δv is the absolute value of the difference of speed of the wheels, and p is the value of the most active sensor ($\overline{v}$, Δv and p are in the range [0;1]). The three parts of this function aim to 1) maximize the speed of the robot, 2) minimize the rotation, and 3) maximize the distance to the obstacles.

The evolutionary parameters are the same as in section 7.3. Ten runs are done with the morphogenetic system and the direct genetic encoding using the real robot. Figure 7.6, left, illustrates the evolution of the fitness. The morphogenetic system achieves a higher fitness score than the direct coding. Moreover, after 15 generations, only five runs manage to find individuals displaying obstacle avoidance behavior with the direct coding, whereas with the morphogenetic system individuals displaying this behavior are found in all ten runs. Figure 7.6 (right) shows the typical behavior of a robot in the arena.

During evolution the robot might run into walls. However, since the fitness function penalizes such behaviors, over time the robot tends to steer away when getting too close to obstacles. The evolution of a controller (15 generations) takes about 13 hours. The robot is capable of avoiding obstacles also in different arenas (e.g. square instead of rectangular) and with various obstacles placed within the arena.

7.5 Hardware implementation of the robot controller

In this section we describe how the multi-cellular spiking neural controller described in section 7.4 is implemented on a FPGA module embedded on the Khepera robot. A FPGA module is used rather than the POEtic chip because the POEtic chip is not yet available, still an implementation on a commercial FPGA allows to estimate the resources required for an implementation on the POEtic chip.

The FPGA module is an extension module that can be plugged on the Khepera robot.

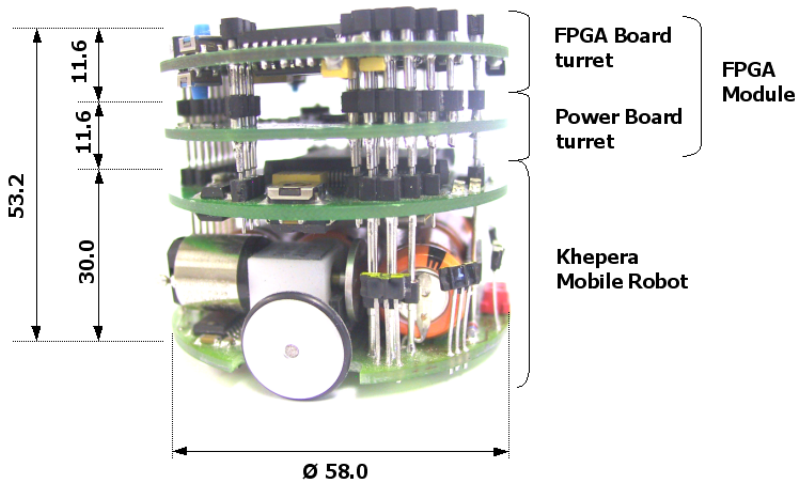

Figure 7.7: The FPGA module, here mounted on top of the Khepera robot, is composed of two circuit boards (or turrets): one for the FPGA and another for the power supply.

When the FPGA module is plugged on the robot, the FPGA can control the robot and access its sensors. Figure 7.7 illustrates the FPGA module on the Khepera robot.

The FPGA is configured following the structure of the POEtic chip, with an *organic subsystem* that contains the multi-cellular circuit (i.e. the robot controller described in section 7.4) and an *environment subsystem* that executes the genetic algorithm and the morphogenetic system, and that interfaces the sensors and motors of the robot with the neurons in the multi-cellular controller (figure 7.8).

The organic subsystem is composed of an array of cells where each cell implements a spiking neuron. In chapter 4 we introduced the notion of cells composed of three layers: the phenotype which is the functional part of the cell, the mapping where development is implemented, and the genotype which contains the genetic code of the entire multi-cellular circuit (figure 7.9 left).

We already described a circuit composed of cells following this three-layered structure in chapter 5, and we extensively considered the mapping layer in chapter 6 with the morphogenetic system. For this reason we focus in this section on the implementation of the phenotype layer of the cells which is the spiking neuron. The genotype and mapping layers are *virtually* implemented in software in the processor of the environment subsystem.

Figure 7.9 (right) illustrates the functional part of the cell and how it fits in the three-layered structure. The functional part of the cell has inputs, which are used to receive incoming spikes from neighboring cells, and outputs, which are used to send spikes to neighboring cells. In addition the cell has a function input which indicates the functionality that the spiking neuron has to take within the family of functionalities defined for the morphogenetic system. The function input allows to select an excitatory or inhibitory neuron with various connectivity patterns, as explained in section 7.2.

To allow the cell functionality to change with the function input, hardware dictates that cells must be totipotent. This means that they must implement all the functionalities

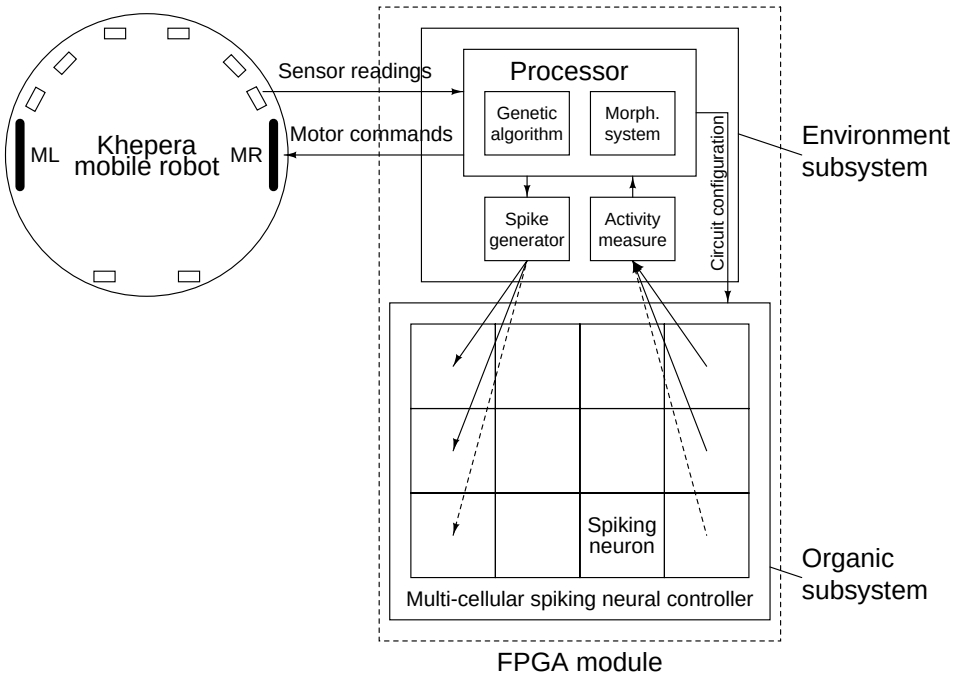

Figure 7.8: The FPGA is configured to implement the multi-cellular robot controller with a structure similar to the POEtic chip, with an organic subsystem and an environment subsystem. The organic subsystem contains the multi-cellular spiking network. Each cell therefore implements a spiking neuron. The environment subsystem consists of a processor that runs the genetic algorithm and the morphogenetic system and that communicates with the Khepera robot to set the motor speeds and read its sensors. In addition the processor sets the inputs of the sensory neurons in the spiking network and reads the output of the motor neurons. This is done via a hardware spike generator unit (that sends spike trains to neurons of the network) and an activity measurement unit (that gives the average activity of neurons of the network).

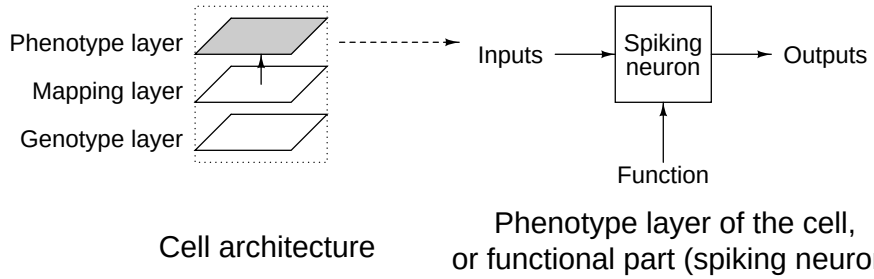

Figure 7.9: Cells of POEtic circuits are composed of three layers: phenotype, mapping and genotype (left). The phenotype layer (indicated in gray) implements the functionality of a spiking neuron (right). Cells have inputs and outputs to exchange spikes with neighboring cells. In addition, since the functionality of the cell must change at run-time when the genetic string of the circuit changes, cells have a function input that indicates which type of spiking neuron the cell must implement, according to the family of predefined functionalities. In the three-layered structure of the cell this function input comes from the mapping layer, as indicated on the left by the arrow pointing upwards.

of the function family of the morphogenetic system, and at run-time the appropriate functionality is selected according to the function input. In particular the cells are connected to all the neighboring cells are required to implement the connectivity patterns described in figure 7.1.

The function input comes from the mapping layer of the cell. Here the mapping layer is virtually implemented by the processor in the environment subsystem. The processor in the environment subsystem executes the evolutionary algorithm and the morphogenetic system. The genetic string of the circuit is therefore stored in the memory of the processor and the development of the circuit with the morphogenetic system is done in software. Once the genetic string is decoded with the morphogenetic system, the processor sets the function input of all the cells, thereby configuring the multi-cellular circuit according to the genetic string.

The processor then interfaces the multi-cellular network with the mobile robot. It sets the inputs and reads the outputs of the multi-cellular network with two a programmable spike generator and an activity measurement unit. The spike generator is used to send spike trains to input neurons according to the sensory information of the robot. The activity measurement unit is used to read the average activity of output neurons to set the speed of the wheels of the robot. At the same time as the robot moves under the control of the network, the processor measures the fitness of the robot according to its speed and sensory readings.

The hardware implementation of the multi-cellular circuit behaves identically to the software simulation described in section 7.4. Therefore the experimental results obtained with the hardware model are identical to those obtained with the software model of the spiking network. In particular the genetic strings of circuits evolved with the software model may be transferred seamlessly to the hardware implementation and the behavior of the robot is identical to that obtained with the software model of the circuit.

Results of the implementation show that the network can be updated 1.2 million times

	Direct encoding		Morphogenetic system	
Training set	Fit	Std dev	Fit	Std dev
A+B	0.585	0.039	0.596	0.044
A+C	0.519	0.037	0.607	0.054
A+D	0.611	0.032	0.605	0.043
A+E	0.529	0.048	0.614	0.052

Table 7.4: Fitness and standard deviation of the fitness at the pattern recognition task in a sample of 5000 randomly generated networks.

per second on the FPGA module. About 98% of the resources of the FPGA are used for the implementation (8222 logic elements on a total of 8320 that the FPGA contains). The multi-cellular network takes 5939 logic elements, the remaining logic elements are used to implement the environment subsystem.

Appendix E describes in more details the FPGA module and the hardware implementation of the spiking neurons.

7.6 Discussion

In this chapter we evolved multi-cellular circuits composed of spiking neurons with the morphogenetic system. We found that the morphogenetic system outperformed a direct genetic encoding when evolving circuits to perform pattern recognition and to control a robot.

In the case of pattern recognition, the morphogenetic system tends to generate individuals of higher fitness and with higher standard deviation of the fitness in the first generation in comparison to a direct encoding (table 7.4). Although the differences are small, the higher initial fitness and higher standard deviation may explain why the morphogenetic system reaches higher fitness scores than the direct genetic encoding. The reason why the morphogenetic systems generates individuals of higher fitness in the first generation is not clear, but it may come from morphological differences between phenotypes randomly generated by both encodings, such as the one evidenced in section 6.6.3. We also evidenced in section 6.3 that some phenotypic structures can be easily evolved with the morphogenetic system (e.g. the *uniform*, *checkerboard* and *mixed 1* patterns). Similarly, circuits capable of pattern recognition may be easier to evolve with the morphogenetic system. Further statistical investigation with more sets of training patterns is necessary to determine whether the morphogenetic system consistently outperforms a direct genetic encoding on this task.

In the case of the robot controller, the morphogenetic system generates large patches of interconnected excitatory neurons that link sensors to motor neurons, causing a reversal of wheel speeds when the robot approaches an obstacle. While the direct genetic encoding can potentially also achieve this, it seems to be slower at it. In this application the morphogenetic system may thus exploit its capacity to generate larger structures of identical cell functionalities in comparison to the direct encoding (section 6.6.3).

The multi-cellular implementation of the robot controller on the FPGA module allows to estimate the resources required for an implementation in the POEtic chip. The elementary logic units of the POEtic chip (molecules) and of the FPGA (logic elements) are both based on a 16-bit look-up table (LUT) which can be configured as two 3-input LUTs to implement efficiently arithmetic operations. Assuming that a logic element of the FPGA is roughly similar to a molecule, about 6000 molecules are required for the implementation of this circuit in the POEtic chip.

The implementation of the multi-cellular circuit on the POEtic chip may however benefits from features of the chip. The dynamic routing mechanism of the POEtic chip allows to build connections between components at *run-time* (chapter 4). This feature does not exist on conventional FPGAs. Therefore the neurons implemented on the FPGA were connected to all their potential neighbors, and the appropriate connectivity was selected at run-time. With dynamic routing cells may connect at run-time to the appropriate neighbors and this may reduce the size of the implementation. Furthermore dynamic routing may be used to evolve new connectivity patterns at run-time. This is not possible with the implementation described here.

The hardware spiking network achieves a high update speed (1.2 million updates per second). High update speed can be interesting for example in voice recognition systems or in character recognition systems for postal addresses where a fast throughput is needed. In particular the network implemented here may also be used for the task of pattern recognition of section 7.2.

In the robotic application the network is updated every 5 milliseconds only, and therefore the hardware implementation is slowed down. Further space optimizations may be considered such as using bit-serial arithmetics that allows more compact implementation of arithmetic functions (e.g. additions, comparisons) at the expense of slower execution speed. Also several neurons may be updated by a single “cell” through time-multiplexing which may allow to reduce significantly the size of the implementation.

The spiking neuron forms the phenotype layer of the three-layered cells introduced in chapter 4. The spiking neuron can be combined with the hardware morphogenetic system introduced in chapter 6 (section 6.7) which forms the mapping layer of the cell. Notably the function input of the spiking neurons can be connected to the function output of the morphogenetic elements. Therefore the combination in hardware of the developmental system provided by the morphgenetic system with the functionality provided by the spiking neurons is only a matter of technical implementation.

7.7 Summary

In this chapter we evolved multi-cellular circuits composed of spiking neurons with the morphogenetic system. We evolved them to perform pattern recognition and to control the navigation of a mobile robot in a task of obstacle avoidance. We found that circuits evolved with the morphogenetic system outperformed those evolved with a direct genetic encoding.

We then described the hardware implementation of the robot controller on a FPGA.

This implementation showed that the model of spiking neuron is suited for implementation in multi-cellular hardware. It also allowed us to estimate the resources required for an implementation on the POEtic chip.

In conclusion, this chapter demonstrated that the morphogenetic system could be used to evolve functional multi-cellular circuits. The evolutionary morphogenesis of more complex circuits can now be tackled. At this stage we have considered the combination of evolution and development in multi-cellular circuits. In the next chapters we will apply the evolutionary morphogenesis to circuits capable of learning, thereby considering the third source of bio-inspiration of the POE model.

8 Evolutionary morphogenesis of learning circuits

Abstract[1]

In this chapter we include learning, the third source of bio-inspiration of the POE model, in multi-cellular circuits. Cells of the multi-cellular circuit implement a leaky integrate and fire spiking neuron model that is capable of learning by modifying its synaptic weights with spike-timing dependent plasticity.

Our objective is to demonstrate that the morphogenetic system can evolve multi-cellular circuits capable of learning. We consider a synthetic learning task that is a preliminary to a robotic application requiring learning described in the next chapter. The task consists in learning the direction of motion of synthetic moving stimuli and, after learning, discriminating the direction of motion of stimuli with respect to the direction that was learned.

We first show how learning is performed and how the direction of motion of a stimulus may be detected from the neural activity. We introduce a quantitative measure of the learning performance and we show that this learning performance can be increased over that of a reference network by evolution with the morphogenetic system. In this task the morphogenetic system outperforms a direct encoding.

8.1 Introduction

In this chapter we include learning, the third source of bio-inspiration of the POE model, in multi-cellular circuits. Our objective is to demonstrate that multi-cellular circuits capable of learning can be evolved by the morphogenetic system.

Cells of the multi-cellular circuit implement a leaky integrate and fire spiking neuron model that is capable of learning by modifying its synaptic weights with spike-timing dependent plasticity. The rationale for using spiking neurons is the same as in chapter 7, with the addition that here we consider a time-dependent learning problem that suits the temporal dynamics of spiking neurons. This spiking neuron model is suited for efficient

[1] Part of this work was published in a deliverable of the POEtic project [42].

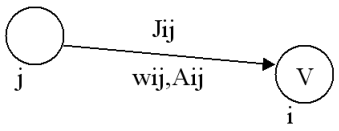

Figure 8.1: Neuron i receives inputs from neuron j. V represents the membrane potential of the neurons. J_{ij} is the synaptic contribution from neuron j to neuron i. W_{ij} and A_{ij} describe the efficacy of the synaptic link between neuron j and i.

implementation in hardware and an optimized implementation on the POEtic chip was proposed by Torres et al. [170, 171, 172].

We consider a learning task where the circuit learns the direction of motion of a synthetic moving stimulus applied to it and, after learning, discriminates the direction of motion of the moving stimulus with respect to the direction that was learned [34]. This task is a preliminary to a more complex learning task in a robotic application shown in the next chapter where the motion of visual cues in the environment of the robot is learned and used for navigation.

We start by analyzing how, after learning, the direction of motion of the stimulus applied to the network can be deduced from the patterns of neural activity. Afterwards we define a measure of the *learning performance* that quantifies how much the patterns of activity differ depending on the stimulus direction.

The learning performance is used as the fitness function in evolutionary experiments. We evolve parameters of the multi-cellular networks with the morphogenetic system to increase their learning performance. We compare the results with a reference network and with evolution with a direct genetic encoding.

All the experiments described in this chapter are done using a software simulation of the multi-cellular circuit.

This chapter is organized as follows. In section 8.2 we describe the neural model and the learning mechanism. In section 8.3 we describe the experimental setup which is used to measure the learning performance and do evolutionary experiments. In section 8.4 we define the learning performance. Section 8.5 shows how the morphogenetic system can be used to improve the learning performance of the circuit. Results are discussed in in section 8.6 before concluding in section 8.7.

8.2 Neural model and learning rules

The neural model is an integrate and fire model with leakage and a refractory period, with learning based on Spike-Timing Dependent Plasticity (STDP) rules [34]. According to these rules, the synaptic weights tend to increase when pre-synaptic neurons emit spikes before post-synaptic neurons, whereas they tend to decrease when post-synaptic neurons emit spikes before pre-synaptic neurons. This model, with the exception of learning, was previously used in simulations of thalamo-cortical circuits [69], and the learning mechanism is a variation of the mechanism proposed by Fusi et al. [46].

• S_i: spike variable (either 0 or 1) • V_i: membrane potential • θ: threshold • W_{ij}: synaptic weight • A_{ij}: synaptic activation. Takes discrete values, e.g. [0;1;2] • J_{ij}: synaptic contribution
• YD: learning variable • $YDMax$: maximum value of YD • L_{ij}: learning state of synapse

Table 8.1: Variables describing the dynamics of a neuron. The variables in the lower part of the table describe the learning dynamics. Variables denoted with the subscript $_{ij}$ refer to a synapse going from neuron j to neuron i.

8.2.1 Neural model

Figure 8.1 shows a neuron i receiving an input from a neuron j. The variables that describe the dynamics of the neuron are listed in table 8.1. The spike variable S_i indicates if neuron i emits a spike. It is one when the membrane potential V_i rises above the firing threshold θ (the neuron emits a spike) or zero otherwise:

$$\begin{aligned} V_i(t) > \theta &\rightarrow S_i(t) = 1 \\ V_i(t) \leq \theta &\rightarrow S_i(t) = 0 \end{aligned}$$

If the neuron emits a spike, the membrane potential for the next period is 0 (reset of the membrane potential). If not, the contributions of the synapses J_{ij} are added to the membrane potential, and leakage is applied by the way of the decay constant $KMem$:

$$\begin{aligned} S_i(t) = 1 &\rightarrow V_i(t+1) = 0 \\ S_i(t) = 0 &\rightarrow V_i(t+1) = KMem \cdot V_i(t) + \sum J_{ij}(t) \end{aligned}$$

Neurons can either be excitatory or inhibitory. If neuron j is inhibitory the contribution J_{ij} to neuron i is negative. The constant $KMem$ can be different for excitatory or inhibitory neurons.

The synaptic contribution J_{ij} is highest just after neuron j fires. It then decays toward zero with the decay constant $KSyn$:

$$\begin{aligned} S_j(t) = 1 &\rightarrow J_{ij}(t+1) = J_{ij}(t) + W_{ij} \cdot A_{ij}(t) \\ S_j(t) = 0 &\rightarrow J_{ij}(t+1) = KSyn \cdot J_{ij}(t) \end{aligned}$$

The constants W_{ij} and A_{ij} represent the synaptic weight and activation. The synaptic weight is a fixed parameter of the model while the synaptic activation is affected by learning. The constants $KSyn$ and W_{ij} may be different depending on the type of the pre- and post-synaptic neuron (excitatory or inhibitory). In this case their name is completed by two subscripts ab. a and b are either 0 or 1 and indicate the type of the post- and pre-synaptic neuron respectively (0 for excitatory, 1 for inhibitory). The state of the neuron is updated every $\Delta t = 1$ millisecond and the various decay constants Kx (e.g. Ksyn,

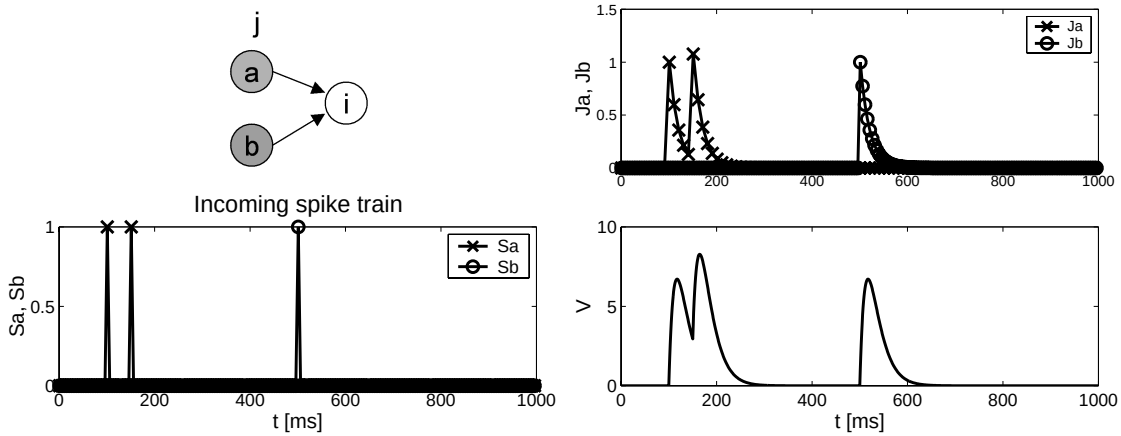

Figure 8.2: Illustration of the neural model. The two neurons a and b are connected to neuron i. Variables Sa and Sb represent the spikes emitted by neuron a and b (lower left plot). Ja and Jb are the synaptic contribution of neuron a and b to neuron i. The synaptic contribution is maximum when the pre-synaptic neuron fires and then it decays (upper right plot). Eventually V represents the membrane potential of neuron i (lower right plot). The constants used in this example are: $KSyn = 0.95$, $KMem = 0.937$, $W_{ij} \cdot A_{ij} = 1$.

Kmem) are given for this update period. The constant Kx implements the exponential decay of the variable x, whose time constant τx relates as follows:

$$Kx = e^{-\Delta t/\tau x}.$$

If the neuron update time Δt is changed, the decay constants Kx must be computed from the time constants τx according to the above formula.

Figure 8.2 shows an example network where two neurons a and b send spikes to a common neuron i; it illustrates the evolution of the synaptic contributions and membrane potential over time.

8.2.2 Learning rules

The learning mechanism affects the synaptic activation variable A_{ij} of connections between excitatory neurons. The other connections are not subject to learning. The synaptic activations of the connections that learn take predefined discrete values, for example [0;1;2;4]. The synaptic activation is updated according to the timing of the pre- and post-synaptic spikes (STDP). A pre-synpatic spike occurring before a post-synaptic spike tends to increase the synaptic activation, whereas a post-synaptic spike occurring before a pre-synaptic spike tends to decrease the synaptic activation.

The synaptic activation is controlled by the variable L_{ij} which is the learning state of the synapse. When there is no activity the learning state drifts to 0. However, when there is frequent correlated pre- and post-synaptic spikes, this variable increases or decreases. When reaching a positive threshold Tp or a negative threshold Tn, the synaptic activation changes (increases or decreases) and the learning state is normalized.

The learning state is updated according to spike timings. The learning variable YD is indicative of spike timings. It takes a maximum value when a spike is emitted, and decays toward 0 otherwise with the decay constant $KLearn$:

$$\begin{aligned} S_i(t) = 1 &\rightarrow YD_i(t+1) = YDMax \\ S_i(t) = 0 &\rightarrow YD_i(t+1) = KLearn * YD_i(t) \end{aligned}$$

The learning state is then updated as follows:

$$L_{ij}(t+1) = KAct * L_{ij}(t) + YD_j(t) * S_i(t) - YD_i(t) * S_j(t)$$

Finally, the learning state is normalized when it crosses a threshold and the synaptic activation is either increased or decreased:

$$\begin{aligned} L_{ij}(t) > Tp \rightarrow L_{ij}(t+1) &= L_{ij}(t) - (Tp - Tn), A_{ij} \nearrow \\ L_{ij}(t) < Tn \rightarrow L_{ij}(t+1) &= L_{ij}(t) + (Tp - Tn), A_{ij} \swarrow \end{aligned}$$

$A_{ij} \nearrow$ means that A_{ij} increases, for instance from 0 to 1, from 1 to 2, or from 2 to 4 if the activations are in the set [0;1;2;4]. $A_{ij} \swarrow$ means that A_{ij} decreases in a similar way.

In this chapter, unless otherwise noted, the synaptic activation of excitatory to excitatory connections takes the values 0 or 1, and prior to learning it is set to 1. The activation of other synaptic connections is always 1.

Figure 8.3 illustrates the effect of pre- and post-synaptic spikes on the learning state L_{ij} of the synapse between neuron j and neuron i.

8.3 Learning setup

The multi-cellular circuit used to learn moving stimuli consists of a network of 20 by 20 neurons, with each neuron connected to its 24 neighbors in a 5x5 neighborhood [34]. Bottom of figure 8.4 illustrates this network. Each cell represents a neuron (the connectivity is not depicted). Darker cells represent inhibitory neurons, which represent 20% of the total number of neurons. This network is used as the *reference network* in the evolutionary experiments. The parameters of the neural model are shown in table 8.2.

A moving stimulus is applied to the network. Applying a stimulus means that the membrane potential of each neuron receives a contribution which is proportional to the amplitude of the stimulus. The amplitude of the stimulus remains constant within a column of neurons, but varies in the horizontal direction. The stimulus moves in the horizontal direction, either to the left or to the right. It is periodical, meaning that the stimulus wraps around at the edges of the network. Top of figure 8.4 shows the shape of the stimulus and its direction of motion.

The stimulus is characterized by its shape, speed, amplitude, and noise. The shape is kept fixed in all experiments. Here it is $sin(x)^{10}$, with x varying between 0 and 2π across the network. As the stimulus is periodical its speed is indicated in Hertz. A speed of 1 Hz means that the bump in the stimulus moves from the left to the right of the array in one

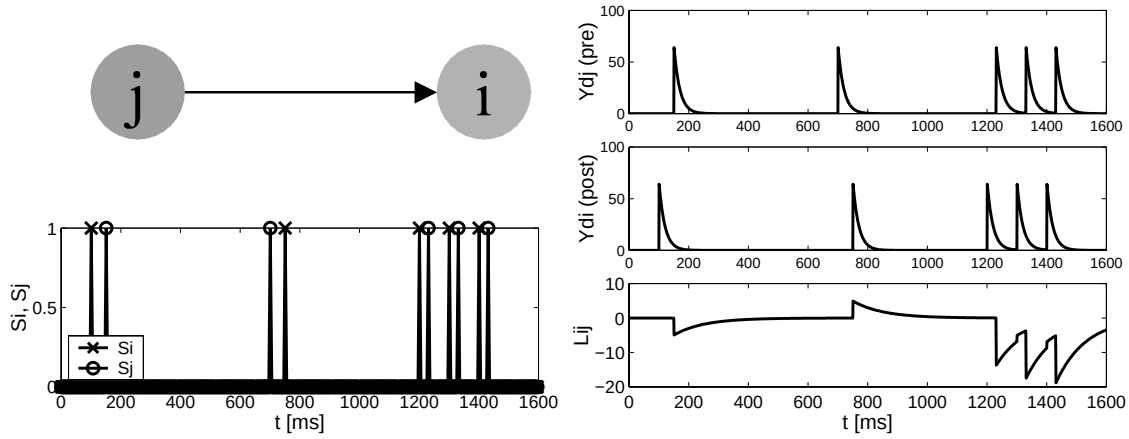

Figure 8.3: Illustration of the learning mechanism. Neuron j is connected to neuron i through synapse ij. Neuron i emits a spike before neuron j near t=200ms. Then neuron j emits a spike before neuron j near t=800ms. There is then a succession of spikes near t=1000ms where neuron i fires shortly before neuron i (bottom left plot). The effect of the spikes on YD_i and YD_j and on the learning state of the synapse L_{ij} is illustrated in the right plots. L_{ij} decreases when a pre-synaptic spike occurs before a post-synaptic spike; in the opposite case L_{ij} increases. The constants used in this example are: $YDMax = 64$, $KLearn = 0.95$, $KAct = 0.99$.

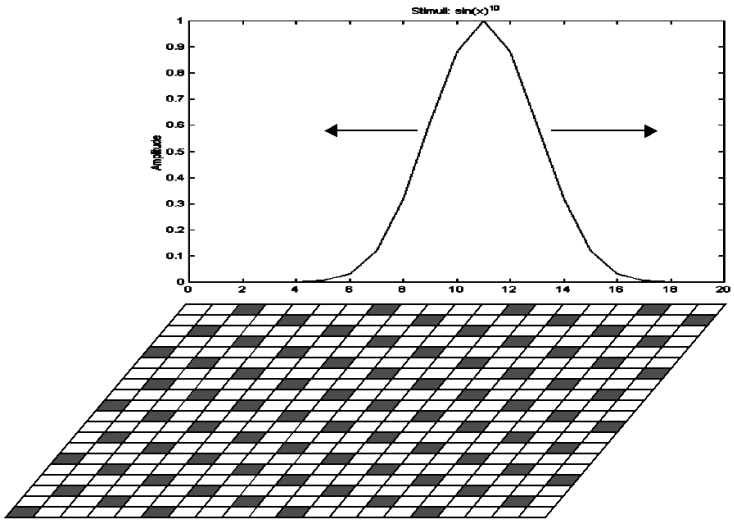

Figure 8.4: The bottom of the figure shows the network composed of 20 by 20 neurons. Each neuron is connected to its 24 neighbors. Darker cells represent inhibitory neurons. This network is used as a reference network in evolutionary experiments. The top of the figure shows the stimuli which is applied to the network. It moves along the horizontal direction. The stimulus amplitude is identical for all neurons within the same column.

Wr_{00}	0.2	Wr_{01}	-4
Wr_{10}	2	Wr_{11}	0
$KSyn_{00}$	0.95	$KSyn_{01}$	0.72
$KSyn_{10}$	0	$KSyn_{11}$	0
$KMem0$	0.937	$KMem1$	0.875
$KAct$	0.999	$KLearn$	0.95
$YDMax$	64	A_{00}	[0;1]
A_{other}	1	θ	10
Tp	128	Tn	-128
Time step	1 ms		

Table 8.2: Parameters of the neural model.

second. Noise is applied to each neuron as part of the stimulus. It is Gaussian noise with mean 0 and is characterized by its standard deviation.

This network is used to learn moving stimuli as follows. First the network undergoes a *learning phase* and then a *recall phase*. During the learning phase the learning mechanism is activated and the network is repeatedly stimulated with a stimulus moving in the *forward* or learned direction along the horizontal axis during some "$TLearn$" time.

Upon successful learning the synaptic connections tend to orient along the direction of the stimulus which is learned (figure 8.5). This occurs because connections oriented along the backward direction tend to see post-synaptic spikes before pre-synaptic spikes, and according to the learning rule this decreases their synaptic activation.

Afterwards learning is turned off by freezing the weights of the synaptic connections and the network undergoes a recall phase in which the network may be used to discriminate the direction of motion of the stimulus applied to it. This aspect is detailed in the next section.

The parameters of the stimulus (speed, amplitude and noise) need not be the same for the learning and recall. In this chapter the stimuli parameters are: during learning, amplitude of 128 mV/64[2], speed of 6 Hz and noise standard deviation of 0; during recall, amplitude of 40 mV/64, speed of 6 Hz and noise standard deviation of 20 mV/64. Appendix F describes how to select these parameters and their influence on learning.

8.4 Learning performance

Since the objective of this chapter is to evolve networks in order to increase their learning performance, we need to define a measure of the *learning performance* which is used as the fitness function in the evolutionary experiments.

We want to learn stimuli which move along the horizontal direction (e.g. left or right), and subsequently detect from the patterns of activity of the network whether the stimulus currently applied to the network moves in the learned (or forward) direction or in the opposite (or backward) direction. Therefore we define the learning performance as a

[2]The unit of amplitude and noise standard deviation are mV/64 because the hardware implementation of the neural model uses fixed-point arithmetic where one unit is equal to 1/64 mV.

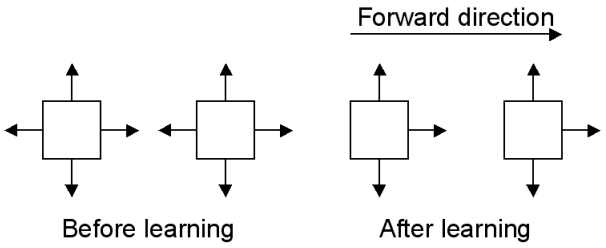

Figure 8.5: On the left, before learning, connections between neurons exist in all directions. After learning (right), the connections which are oriented along the backward direction (opposite to the direction of learning) see their synaptic activation frequently decreased to zero by the learning mechanism (the post-synaptic spikes tend to occur before the pre-synaptic spikes), effectively "removing" the connections.

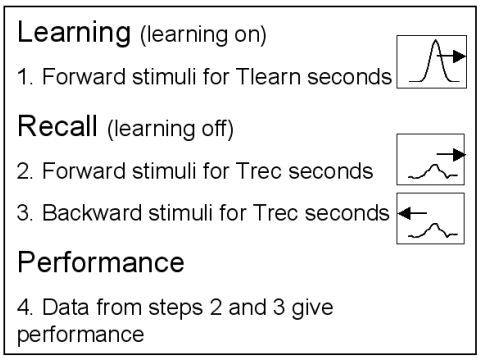

Figure 8.6: The process which is used to collect data to measure the learning performance. After the learning phase there is the recall phase where the stimulus is applied again, once moving in the forward and once in the backward direction. The learning performance is computed based on the data collected during the recall phase.

measure that indicates how much the patterns of neural activity are *different* after learning, when the stimulus moves in the learned direction or in the opposite direction. In other words, this measure indicates how *sensitive* is the network to the moving stimulus. Higher learning performance means that the network is more sensitive and therefore it is potentially easier to detect the direction of motion of the stimulus from the network activity.

The measure of the learning performance must satisfy two criteria: its value should be close to zero when the learning time $TLearn$ is zero, and its value should increase with increased learning time.

The learning performance is measured from the neural activity during the recall phase. The moving stimulus is applied to the network while recording its activity: the stimulus first moves for "$TRec$" time in the same direction as that of learning (the forward direction), then it moves for the same time in the opposite direction (the backward direction). Figure 8.6 illustrates the process used to record the activity. Unless otherwise noted the learning time $TLearn$ is 10 seconds and the recall time $TRec$ is 3 seconds.

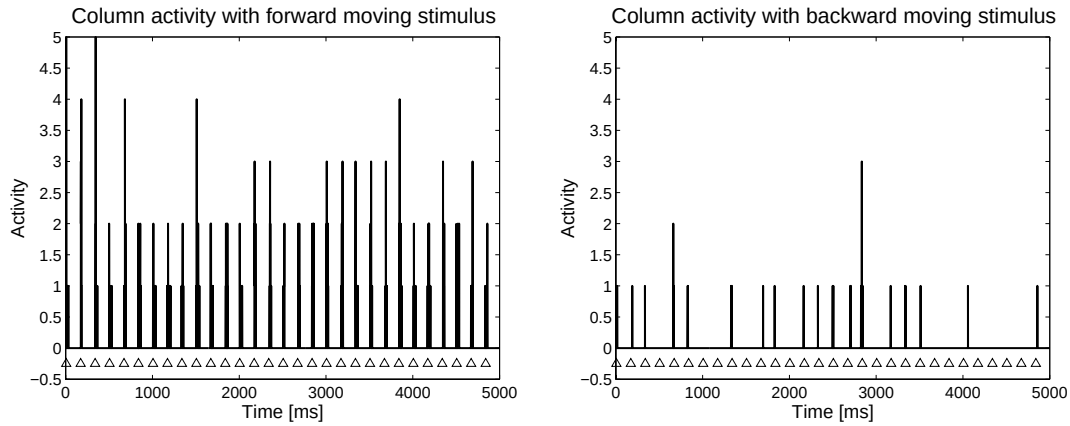

Figure 8.7: Activity in a measurement column in the center of the network with a forward (left plot) or backward (right plot) moving stimulus after 10 seconds of learning. The arrows at the bottom indicate when the bump of the stimulus is situated over the measurement column. When the stimulus moves forward there is always activity in the measurement column when the bump of the stimulus passes over this column. When the stimulus moves backward this is not always the case. This shows that the network activity tends to be correlated with the stimulus when it moves in the forward direction. Also the network shows a higher activity when the stimulus moves forward.

Figure 8.7 shows the number of spikes $\rho(t)$ observed in a measurement column of 20 neurons situated in the center of the network in function of the stimulus direction after learning. The activity of the neurons tends to be correlated with the stimulus when it moves in the forward direction and the activity of the network is also higher in this case.[3]

The measure of the learning performance is based on the observation that the network activity is higher when the stimulus moves in the forward direction than when it moves in the backward direction. The learning performance F_{act} is defined as follows:

$$F_{act} = Fforward - Fbackward,$$

[3]The higher and correlated activity observed with the forward stimulus can be understood by considering the connectivity patterns among neurons that appear after learning (figure 8.5). Connections tend to orient themselves along the direction of the stimulus which is learned. Let us assume that a neuron fires when the bump of the stimulus passes over it (i.e. the combined contribution of the Gaussian noise and the stimulus leads to the neuron firing). The firing of this neuron tends to contribute (increase) the membrane potential of neurons situated along the forward direction, whereas it seldom increases the membrane potential of those situated along the backward direction because the synaptic activation of most of these connections is 0. The noise and amplitude of the stimulus are selected such that this contribution is critical to ensure the firing of neighboring neurons when the bump of the stimulus passes over them. Therefore the higher membrane potential of the neurons in the forward direction increases their firing probability when the stimulus moves forward. On the contrary neurons along the backward direction relative to the neuron which fired tend to have a lower membrane potential than those situated along the forward direction and therefore if the stimulus moves in the backward direction the probability of these neurons firing is lower. As a consequence the activity of the network when the stimulus moves backward tends to be lower than when the stimulus moves forward. In this case the activity in the measurement column may also decorrelate from the stimulus (i.e. sometimes there may be no activity in the measurement column when the stimulus moves backward).

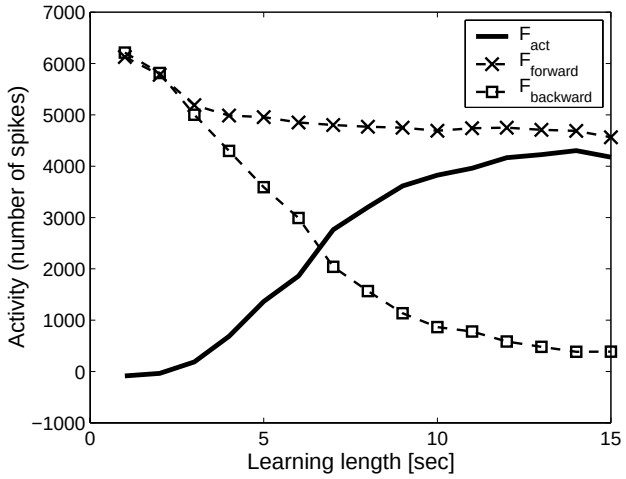

Figure 8.8: Learning performance F_{act} in function of the duration of learning $TLearn$. The recall time $TRec$ is 3 seconds. The figure also illustrates $Fforward$ and $Fbackward$. The vertical axis represents the difference of spike counts measured during the time $TRec$ for F_{act}, and the spike counts measured during the time $TRec$ for $Fforward$ and $Fbackward$.

where $Fforward$ and $Fbackward$ are the number of spikes counted in the network (or also the network activity) while the stimulus moves in the forward respectively backward directions during the recall time $TRec$. We also refer to $Fforward$ and $Fbackward$ as the forward and backward activity respectively.

This measure is simply a difference of spike counts, which has the advantage of being easy to compute. Figure 8.8 shows $Fforward$, $Fbackward$ and F_{act} in function of the learning time $TLearn$. F_{act} is close to 0 when $TLearn$ is small, and it increases smoothly with longer learning time, which satisfies the criteria for a measure of the learning performance.

8.5 Evolution of the learning circuits

In this section we demonstrate that the multi-cellular circuit introduced above can be evolved with the morphogenetic system to increase its learning performance. We compare the performance of the evolved circuit to the reference network illustrated in figure 8.4. The fitness function is the measure of the learning performance F_{act} described in section 8.4.

Many parameters of the network may be evolved but we only consider the evolution of two of them, which may be difficult to select by analytical methods: the type of the neurons, and the maximum synaptic activation. In the first case we compare the results of evolution with a direct genetic encoding.

GA parameters	Genetic coding
Population: 50	Coding: morphogenetic system
Crossover: 20%	Diffusers: 16
Mutation: 1%	Expression table: 2 entries
Selection: rank (15 best selected)	Chromosome size: 192 bits
Elitism: 5 individuals are copied unchanged	

Table 8.3: Parameters of the genetic algorithm and of the morphogenetic system.

8.5.1 Neuron type

The reference network (figure 8.4) contains a regular distribution of excitatory and inhibitory neurons. Instead of using this predefined distribution of neurons, we evolve the type of the neurons with the morphogenetic system. Each cell can therefore implement one of two predefined functionalities (an excitatory or inhibitory neuron) and the expression table of the morphogenetic system contains two entries. The parameters of the genetic algorithm and of the morphogenetic system are shown in table 8.3. The parameters of the stimuli during learning and recall are those indicated in section 8.3. $TLearn$ is 10 seconds, $TRec$ is 3 seconds. Evolution is repeated 5 times and figure 8.9 shows the evolution of the maximum and average fitness averaged on the 5 runs. The morphogenetic system manages to find networks that have a better maximum fitness than that of the the reference network (horizontal line in the figure) (T-test, p=0.014).

As a comparison, we evolve the type of the neurons with a direct genetic encoding. The parameters of the genetic algorithm are indicated in table 8.3. The genetic string of the circuit consists of one bit per neuron (excitatory or inhibitory neuron). The size of the genetic string is thus 400 bits. The initial population is seeded with 10 genetic strings representing networks composed of 80% of excitatory neurons, which is the ratio of excitatory neurons in the reference network. After more than 100 generations no networks achieve a higher fitness than that of the reference network (figure 8.10).

Analysis of the networks in the last generation shows that the networks obtained with the morphogenetic system tend to have large regions of interconnected excitatory neurons, whereas networks obtained with the direct genetic encoding tend to have smaller regions containing excitatory neurons (figure 8.11). Furthermore on average on the entire population, networks evolved with the morphogenetic system tend to display more excitatory neurons (65%) than networks evolved with the direct encoding (52%). A higher number of excitatory neurons tends to increase the activity in the network and this may lead to higher fitness scores since the fitness is linked to the network activity.

8.5.2 Neuron type and maximum synaptic activation

In previous evolutionary run the synaptic activation variable A_{ij} could take the value 0 or 1 under the action of the learning rule. A larger range of values for the synaptic activation may however be used. To determine the upper bound of the synaptic activation we resort to evolution. The minimum value of A_{ij} is zero, but the maximum value of A_{ij} ($AMax$) is evolved in the set [0;1;2;4] (the maximum value of A_{ij} is the same for all the neurons

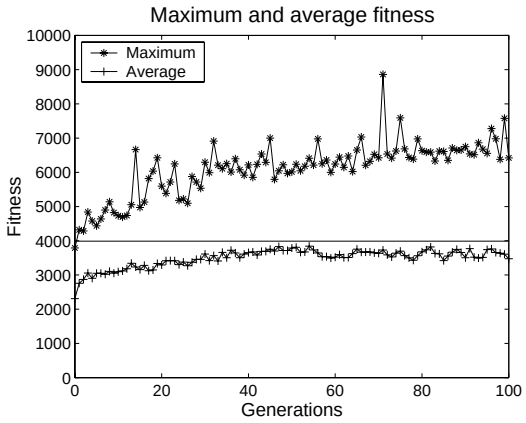

Figure 8.9: Maximum and average fitness when evolving the type of the neurons with the morphogenetic system (average of 5 runs). The horizontal line shows the performance of the reference network.

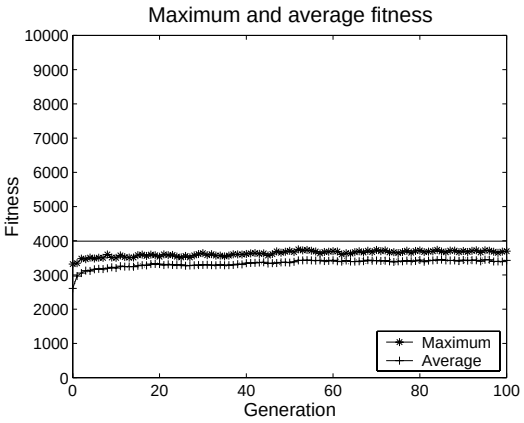

Figure 8.10: Maximum and average fitness when evolving the type of the neurons with the direct genetic encoding system (average of 5 runs). The horizontal line shows the performance of the reference network.

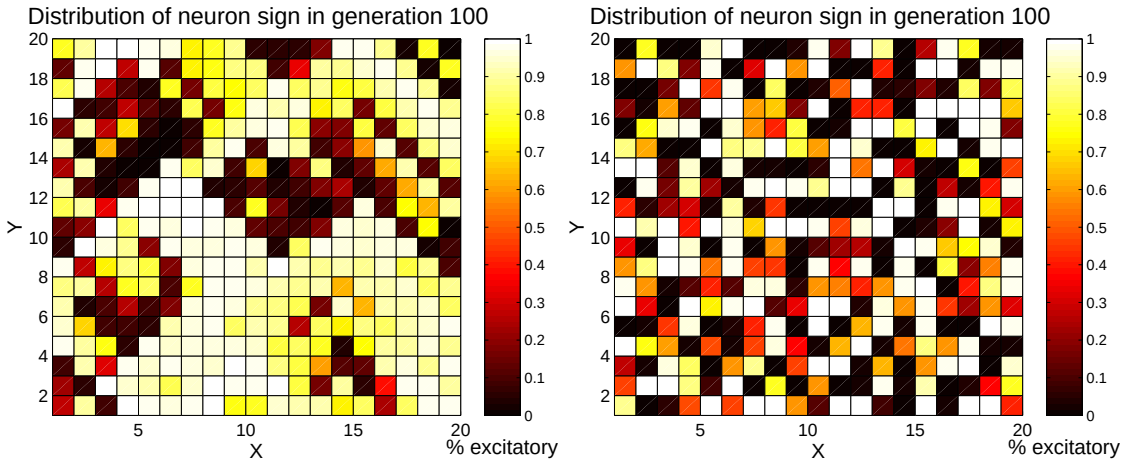

Figure 8.11: Distribution of excitatory neurons at generation 100 in a typical evolutionary run with the morphogenetic system (left) and a direct genetic encoding (right). Each square represents one neuron of the network and the color intensity is the percentage of excitatory neurons in the population at that location (brighter colors mean mostly excitatory neurons). Networks obtained with the morphogenetic system tend to have large regions of interconnected excitatory neurons, whereas networks obtained with the direct encoding tend to have smaller regions of excitatory neurons.

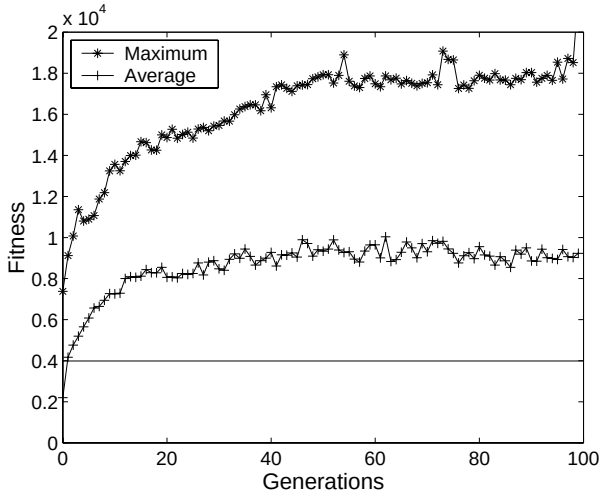

Figure 8.12: Evolution of the maximum and average fitness (average of 5 runs) when evolving the neuron type and $AMax$.

in the network). The length of the genetic string is 194 bits. The first 192 bits encode the type of the neurons with the morphogenetic system as above. The last 2 bits encode $AMax$ in binary (i.e. 0, 1, 2 or 4). The networks are evolved with the same stimuli parameters as above. Figure 8.12 illustrates the maximum and average fitness obtained with this setup. In comparison to the previous results obtained with the morphogenetic system with $AMax = 1$, the maximum fitness that is obtained here is higher (T-test, $p<0.001$).

Analysis of the evolved networks shows that evolution favors $AMax = 2$ (i.e. A_{ij} can take the values of 0, 1 or 2) (figure 8.13). Indeed allowing higher values of A_{ij} let spikes have a stronger influence on the membrane potential of post-synaptic neurons. This may increase the fitness of the network by allowing higher activity when the stimulus moves in the learned direction. There is however a trade-off between the value of $AMax$ and the risk of generating locked oscillations. Locked oscillations correspond to the state of the network where the neurons are not sensitive anymore to the input stimulus and continuously emit spikes. With high values of the synaptic activation a spike may increase the potential of connected neurons in such a way that they immediately fire and sustain the firing, even if the stimulus is removed or moves in the backward direction. Therefore evolution has to balance between higher activity (and potentially higher fitness) that higher values of $AMax$ may provide and the risk of generating locked oscillations that tend to decrease the fitness.

8.6 Discussion

We developed a measure of the learning performance based on the network activity that we used as the fitness in evolutionary experiments. In a hardware implementation the learning performance can be measured efficiently with a simple counter. Alternate mea-

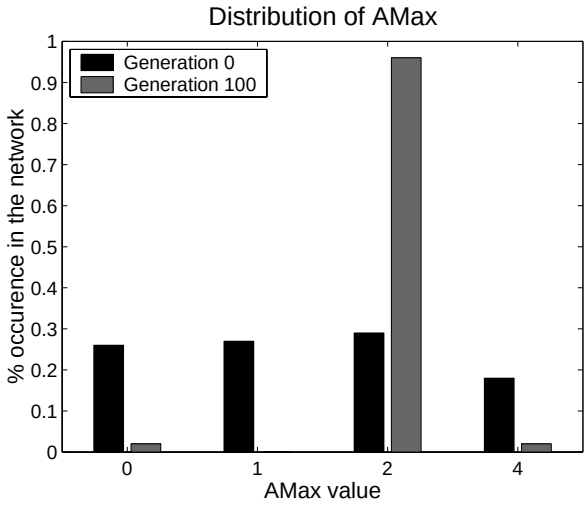

Figure 8.13: Distribution of the values of $AMax$ at generation 0 and generation 100 in the population of networks (average of 5 runs). Evolution favors $AMax = 2$.

sures based on firing synchrony or firing correlation may be devised (see appendix F) but they are not used here because they are less precise or more computationally demanding than the one described in this chapter.

Since the network activity varies in function of the stimulus direction during recall, the direction of motion can be detected by comparing the activity of the network to a predefined threshold. Alternatively a "readout" neuron that is connected to all the neurons of the network may be used: by selecting appropriate constants of the neural model, the readout neuron can be made to fire only with a forward moving stimulus.

We evolved the neuron type and the maximum synaptic activation with the morphogenetic system and we showed that it was possible to increase the learning performance through evolution. In particular the morphogenetic system outperformed a direct encoding when evolving the type of the neurons. Networks obtained with the morphogenetic system displayed large areas of connected excitatory neurons whereas those evolved with a direct encoding showed smaller areas of connected excitatory neurons. Large areas of excitatory neurons may increase the activity in the network and therefore allow to achieve higher learning performance. In this task the morphogenetic system may thus benefit from its ability to easily set large area of cells to identical functionalities, as we showed in chapter 6.

In some evolved networks neurons may enter in locked oscillations, meaning that they continuously emit spikes regardless of the stimulus direction. The detection of the stimulus direction may be more difficult in the presence of locked oscillations. Nonetheless this problem may be alleviated simply by selecting after evolution an appropriate threshold against which the network activity is compared to determine the direction of motion of the stimulus. Alternatively this threshold may also be evolved.

The multi-cellular network is capable of learning stimuli moving in a range of speeds which is defined by the neural time constants (appendix F shows the influence of the stimuli speed on the learning performance). By evolving the neural time constants the

dynamics of the network may adapt to the of stimuli likely to be encountered in the environment, e.g. stimuli moving at different speeds than the one we used here.

In this chapter we considered learning in the biological sense (i.e. change of synaptic weights over time). We did not consider the issue of generalization that is usually associated with learning in machine learning. Generalization might be considered as the capacity of a network to learn a stimulus moving at a particular speed and by generalizing from this experience to also detect stimuli moving at other speeds. As mentionned above, the range of speeds in which learning is possible is controlled by the network time constants. Therefore, although the network is indeed learning in a biological sense, in a machine learning sense it is mostly memorizing events.

8.7 Summary

In this chapter we included learning in multi-cellular circuits and we considered the evolutionary morphogenesis of these learning circuits.

We used a multi-cellular network composed of spiking neurons with spike-timing dependent plasticity to learn and subsequently discriminate the direction of motion of a stimulus applied to the network. We considered this task because it is a time-dependent learning problem which is suited to the temporal dynamics of spiking neurons, and because we intend to evolve this circuit in the next chapter to control the navigation of a mobile robot for a learning task in an environment with moving visual cues.

We showed that, after learning, the direction of motion of a stimulus applied to the network could be deduced from the neural activity. From this observation we devised a measure of the learning performance which indicates how much the network activity changes in function of the stimulus direction. We used this measure as the fitness function in evolutionary experiments.

Finally we evolved multi-cellular circuits with the morphogenetic system to increase their learning performance. We successfully found networks displaying higher learning performance than a reference network. Furthermore the morphogenetic system outperformed a direct genetic encoding when evolving the type of the neurons in the network. We therefore demonstrated that the morphogenetic system could be applied to the evolution of multi-cellular circuits capable of learning.

9 Evolutionary morphogenesis of learning mobile robot controllers

Abstract[1]

In the previous chapter we considered the evolution of multi-cellular circuits capable of learning. We used these to learn and discriminate synthetic moving stimuli. Our objective in this chapter is to build upon the results that we obtained in the previous chapter and demonstrate that these multi-cellular circuits can be evolved with the morphogenetic system to control the navigation of a robot in a task that requires learning capabilities. Learning is implemented with the same neural model as in the previous chapter.

We evolve with the morphogenetic system a multi-cellular circuit that controls the navigation of a robot in an environment with moving visual cues. The task of the robot is to learn one of the moving visual cues, and afterwards it has to perform a homing behavior toward the cue that it has learned. Therefore learning at the synaptic level of the neural model induces learning at the behavioral level of the robot. We call this application *motion-based* navigation because the information that the robot has to rely on to navigate is the motion of the visual cues. Results show that the multi-cellular controller can be successfully evolved with the morphogenetic system, and behavioral tests show that the robot is able to home towards the learned cue from most of the locations in the arena. Since the morphogenetic system combines evolution and a developmental system, the application described in this chapter combines the three axis of the POE model of bio-inspiration: evolution, development and learning.

9.1 Introduction

In this chapter we want to investigate the evolutionary morphogenesis of multi-cellular circuits capable of learning to control a robot in a task that requires learning capabilities.

[1]Part of the experiments were carried out by Tiago Bertolote who studied the optimization of the size and connectivity of the retina during a term project. Thierry Frank contributed by developing the LED displays used in the robotic experiments during another term project. Both projects were carried out under my supervision.

In particular we want to explore evolutionary robotics in the context of *dynamic* environments, where the meaning of environmental cues can only be understood by their change over time. This means that the sensors of a robot must be integrated over time to reveal information such as, for instance, the direction or the velocity of movement of environmental cues (e.g. robots or objects in the environment). Among dynamic environments we consider those with dynamic *visual* cues as particularly interesting. In comparison to common robotic sensors such as ultra-sound or infra-red which give information about a specific point in the environment, vision gives an entire "picture" of the environment: the environment is sensed up to very long distances and with a wide field of view using nowadays cheap cameras. Therefore vision has the potential to bring more information about the environment and in a shorter amount of time than other sensors. Vision is not only rich in information in the spatial domain (e.g. a static image) but also in the time domain. For instance changes across several frames may allow to detect the speed of motion (of the robot or of other objects in the environment), to detect changing signals (e.g. a blinking alarm sign), to perform chasing behaviors (estimating the speed of the target with respect to its own speed), to detect doors opening or closing, etc.

In this chapter we therefore consider a robotic task in an environment with dynamic visual cues. The task builds on the results that we obtained in the previous chapter where we showed that multi-cellular circuits composed of spiking neurons with spike-timing dependent plasticity could learn and discriminate synthetic moving stimuli.

The robot is placed in an arena and has to learn a moving visual stimulus in the environment (the dynamic cue). After learning the robot has to perform a homing behavior toward the learned stimulus. If the robot can successfully do this, it means that the neural controller can learn moving visual cues and later on react to the learned cue by triggering a particular robot behavior. Learning at the synaptic level of the neural model thus induces learning at the behavioral level of the robot. Since navigation is based on moving visual cues, we say that this application consists of *motion-based* navigation.

In this chapter we start by assessing whether a multi-cellular circuit can learn and discriminate real moving stimuli in the robotic environment. This is necessary because real stimuli may differ from the synthetic stimuli used in the previous chapter and furthermore the rate of acquisition of images is limited by technical factors in comparison the rate that was used in the previous chapter.

Afterwards we investigate whether the morphogenetic system can evolve a multi-cellular circuit to control the navigation of a mobile Khepera robot for the task described above. We perform extensive behavioral tests of the best evolved controllers on the real robot.

Since the primary objective of this chapter is to investigate whether the morphogenetic system can evolve multi-cellular circuits for this robotic task, all experiments are done in a software simulation of the multi-cellular circuit. The implementation of the multi-cellular circuit on the POEtic chip is a matter of technical development. We briefly outline how this implementation may look like and we estimate the number of POEtic chips necessary for it.

The rest of this chapter is organized as follows. Section 9.2 describes the experimental setup and the robotic task, and it highlights its challenges. Section 9.3 describes the neural

model and the learning circuit used in this task. In section 9.4 we assess whether a multi-cellular circuit is capable of learning and discriminating real visual stimuli in the robotic environment. In section 9.5 we describe the evolution with the morphogenetic system of the multi-cellular circuit to control the navigation of the robot. Results are discussed in section 9.6 where we also estimate the number of POEtic chips required for the hardware implementation of the circuit. The chapter concludes in section 9.7.

9.2 Setup and robotic task

The robotic setup (figure 9.1) consists of a rectangular arena (40 cm x 70 cm) with two LED displays (30 cm wide, 5 cm high) placed on opposite sides which show a vertical 3 cm wide luminous bar moving along the horizontal direction as indicated in the figure. The two bars move in opposite directions for an observer placed in the arena. The bars move at a speed of 6 Hz and are periodical: they move from one extremity of the display to the other 6 times per second.

The robot is a two-wheel differential drive Khepera robot [120] with a custom CMOS digital camera module capable of acquiring images at 50 frames per second. The camera module is described in appendix G. The camera field of view is approximately 25°. This means that the display fills the field of view when the robot is placed in the center of the arena.

The robot sends the data from the camera and its sensors via a cable to a desktop computer that simulates the multi-cellular spiking network, and in return it receives the motor commands from the computer.

The robotic task consists of the two phases illustrated in figure 9.2. First the robot is still, facing one of the moving bars. This is the *learning* phase. During this time the spiking neural controller is allowed to modify its synaptic weights according to the neural learning rules. Afterwards learning is deactivated and the robot is randomly placed in the environment and has to do a *homing* behavior which consists in navigating back toward the learned bar. Since both moving bars are displayed at all time during the experiment, the robot cannot simply home toward a luminous bar, but it really has to learn and discriminate the direction of motion of the bar. In the evolutionary experiments, fitness areas placed on the floor of the arena in front of the displays are used to measure the fitness of the robot using its floor sensor.

This robotic task relies on moving stimuli to perform navigation. Since the robot is itself moving, the relative motion of the objects that the robot perceives varies in function of the robot own motion (ego-motion). In other words, the motion vectors of objects in the environment sum up with the own motion vector of the robot to give the relative motion vectors that the robot perceives. Therefore the robot ego-motion may hamper the detection of the desired stimuli. This is a challenging task that the evolved controller needs to solve. Another challenge is that the moving stimulus may be difficult to perceive when the robot is very close to it (because it appears during a short time in the field of view of the robot) or far from it (because it fills only a small part of the field of view of the robot).

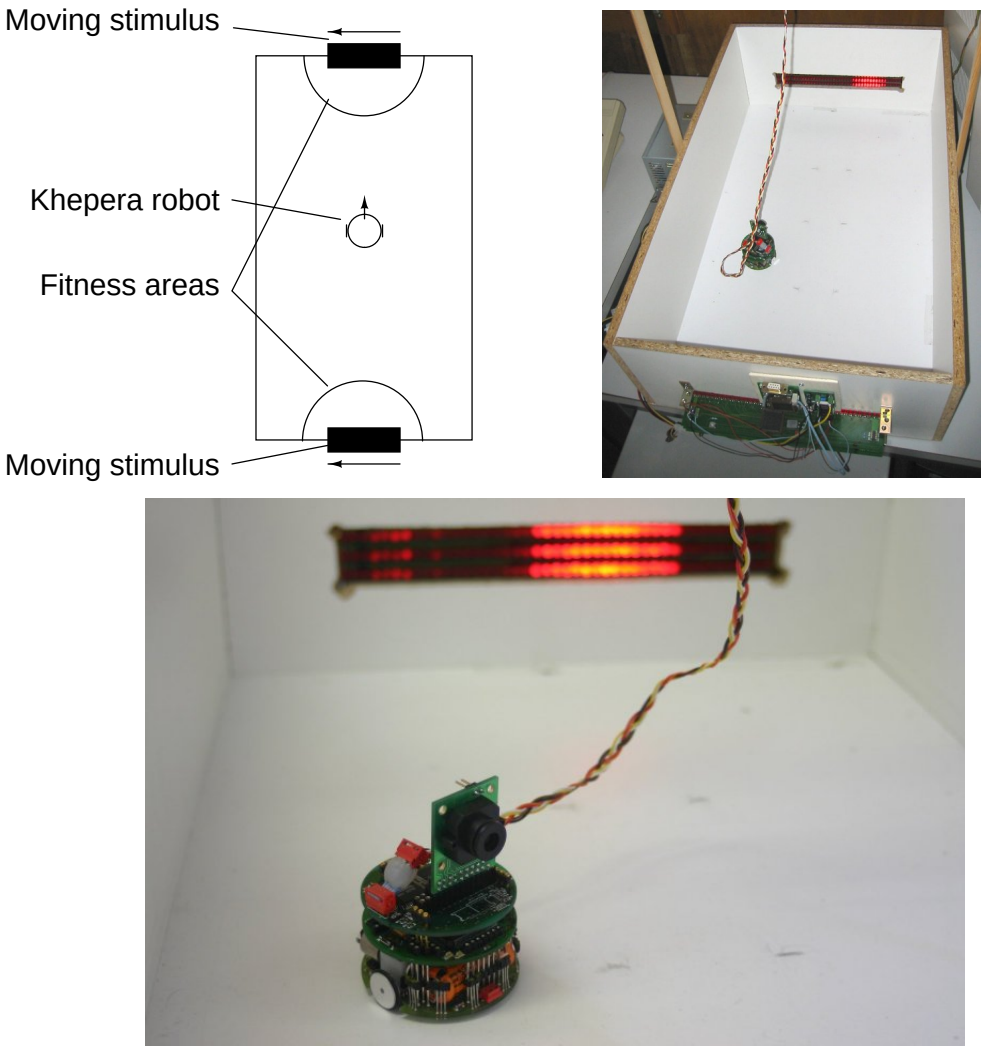

Figure 9.1: Top: schematic of the arena used in the robotic experiment (left) and real setup (right). Two displays placed on the opposite sides of the arena show a vertical bar moving in the direction indicated by the arrows. The fitness areas placed on the floor next to the displays are used to measure the fitness of the robot in the evolutionary experiments using the floor sensor of the Khepera robot. The two fitness areas have different colors so that it is possible to detect on which one the robot is located. Bottom: close-up of the Khepera robot with its custom 2D camera. The robot receives power and motor commands from the cable and sends its sensory information (vision, proximity sensors, speed of the wheels) to the computer simulating the neural network by the same cable. The LED display in the background shows the moving stimuli.

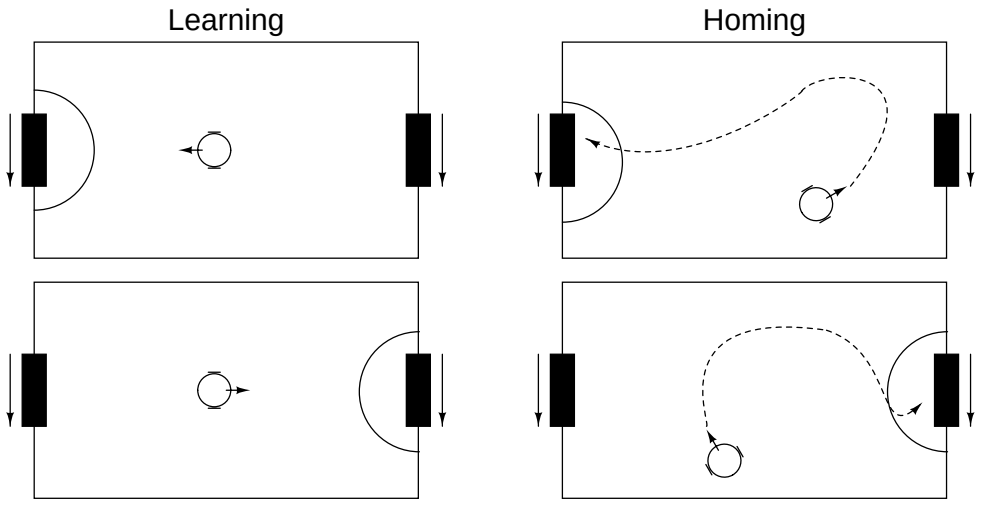

Figure 9.2: The robotic task consists of two phases. During *learning* (left figures) the robot faces one of the moving bars and the synaptic weights change according to the learning rule of the spiking neurons. Afterwards the robot is randomly placed in the arena and has to do a *homing* behavior; i.e. it has to navigate toward the learned bar (right figures).

To minimize the time required for the experiments, and in particular the evolutionary experiments, they are all done with a software simulation the mobile robot. The best evolved controllers are however validated on the real robot afterwards. The Khepera simulation is a minimalist simulation with noise added to the sensors and actuators [75]. The camera is simulated as a 1D linear camera using ray-tracing techniques. Figure 9.3 illustrates what the robot "sees" when placed in the center of the arena and facing the left-moving stimulus.

9.3 Neural model and learning retina

The neural model is the leaky integrate and fire model with spike-timing dependent plasticity that we described and analyzed extensively in chapter 8. The parameters of the model are the same as those used in that chapter (table 8.2).

In order to learn and discriminate moving visual stimuli, we use a 2D array of neurons with local lateral connections to their neighbors, as in chapter 8. However, instead of using synthetic stimuli, real stimuli coming from a camera mounted on the robot are used. We refer to this network as the *retina* since it relates to the robot vision.

Chapter 8 showed that a network of 20x20 neurons with each neuron connected to its 24 neighbors could learn moving stimuli. However the experiments of that chapter were slower than real-time (i.e. a 1 ms update of network took more than 1 ms to compute). This was not an issue in these simulations, however here it is important that the network controlling the robot runs in real-time since the robot is continuously moving. Therefore the network size and the connectivity neighborhood are reduced. An 8x8 network with local lateral connectivity to 14 neighbors is used: the neighborhood is 5 neurons horizontally and 3 neurons vertically. The rationale leading to this network is explained in

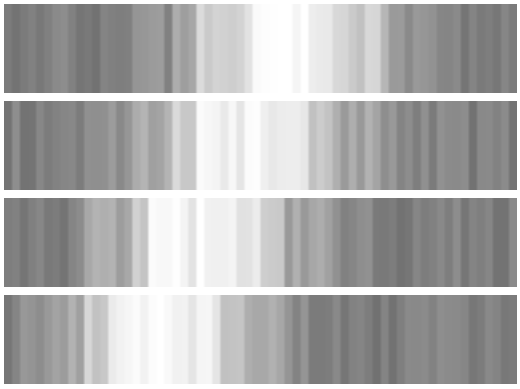

Figure 9.3: Simulated vision of the robot when located in the center of the arena and facing the left-moving stimulus at 6 Hz. Each image represents a horizontal line of pixels that is vertically situated in the middle of the field of view of the robot. Each image is taken after 10 ms.

appendix H.

Figure 9.4 illustrates how this retina is used in the robotic setup and the distribution of excitatory and inhibitory neurons in the retina. The visual stimulus is acquired from a camera mounted on the Khepera robot (the camera is described in appendix G). After acquisition, the image is preprocessed on the robot. Preprocessing consists of digital image stabilization which is necessary because the robot tends to pitch forward and backward while moving. Finally a single line image which is 24 pixels wide is sent every 20 ms to the computer simulating the neural network. A single line image is used because the stimulus moves along the horizontal axis only.

The line of pixel is used as the stimulus which is applied to the neural network. Each pixel is applied to a column of neurons (e.g. the leftmost pixel is applied to the leftmost column of neurons). This means that the membrane potential of all the neurons of a column of the network receive a contribution which is proportional to the brightness of the pixel. When the line of pixels is wider than the number of neurons it is downsampled accordingly (subsampling at regular intervals).

The contribution of the pixels is scaled so that the brightest pixel (value of 255 on 8 bits) corresponds to an increase of the membrane potential at each network step of 2 mV during learning and 0.6 mV during homing. In addition Gaussian noise with a standard deviation of 0.3 mV is added to the membrane potential of each neuron during homing. These parameters are selected according to the characterization done in chapter 8 to ensure that learning is possible. The effective stimulus is however affected by environmental factors such as ambient light, flickering of neon lights, distance to the display, and brightness and contrast gains of the camera.

As in chapter 8 the retina is updated every millisecond. Due to technical limitations images are acquired from the camera every 20 ms. Therefore the network receives the same visual input during 20 ms. This departs from experiments of chapter 8 where the synthetic stimulus was “acquired” every millisecond. The output of the retina (the V box in figure 9.4) is the number of spikes measured during 100 ms.

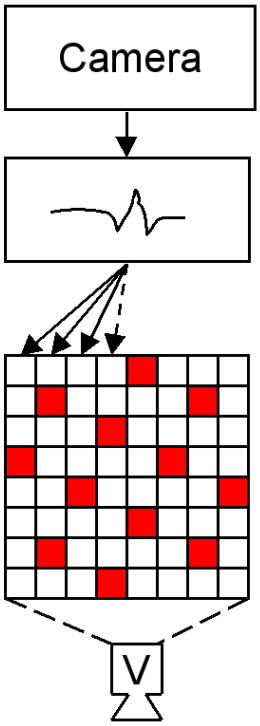

Figure 9.4: The learning network, or retina, is a 2D array of 8x8 neurons, with each neuron connected to its local neighbors (5x3 neighborhood). It is stimulated every millisecond by the input coming from the camera after preprocessing (image stabilization): each column of neurons receives a contribution to the membrane potential which is proportional to the brightness of the corresponding pixel. The V box (V for vision) at the bottom is the number of spikes occurring in the retina during 100 ms. Dark cells represent inhibitory neurons, the other cells are excitatory neurons.

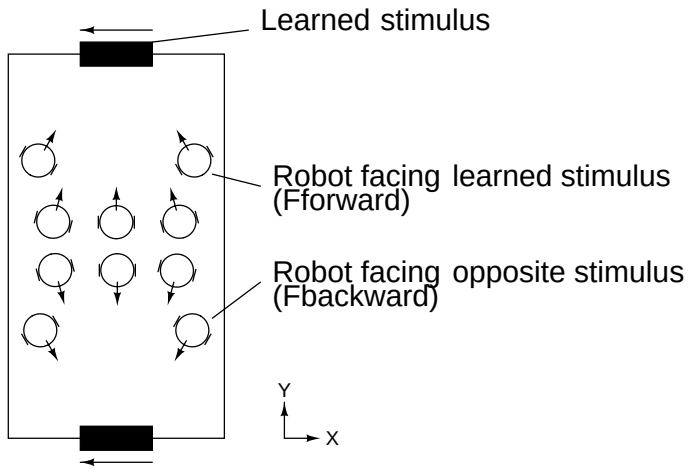

Figure 9.5: After learning, the activity of the retina is measured for all the positions of the robot in the arena. This is done twice: once with the robot facing the learned stimulus, and once with the robot facing the opposite stimulus. The robot is does not move during the measures.

9.4 Learning in the robotic setup

In this section we show that the retina can learn and discriminate the moving stimuli in the robotic setup, even though the real stimuli differ from the synthetic stimuli used in the chapter 8: here we use luminous bars whereas in the chapter 8 we used sinusoidal stimuli. In addition we illustrate the influence of the robot location in the arena and of ego-motion on the activity of the retina.

In the following experiments, when learning is deactivated, the noise and the contribution of the pixels to the membrane potential of the neurons correspond to the settings indicated in section 9.3 for the homing.

The effect of the position of the robot in the arena is tested as illustrated in figure 9.5. First the robot learns one of the stimulus for 30 seconds (e.g. the left-moving stimulus in the figure). Afterwards learning is deactivated and the activity of the retina is measured for all the positions of the robot in the arena, with the robot always facing one of the stimulus (the robot does not moves). This is repeated twice: once with the robot facing the learned stimulus, and once with the robot facing the opposite stimulus. For each position of the robot the number of spikes emitted by the retina during 1 second is counted. $Fforward$ and $Fbackward$ are respectively the number of spikes recorded while the robot faces the learned stimulus and the opposite stimulus. The measures are replicated three times and averaged.

Figure 9.6 illustrates the activity $Fforward$ and $Fbackward$ in function of the position of the robot in the arena. The results show that the activity of the retina is about 10 times higher when the robot is located near the center of the arena and faces the learned stimulus (brightest "half moon" in the left plot) in comparison to the activity when it faces the opposite stimulus. This means that, in this area, it is possible to easily detect which stimulus the robot faces by monitoring the retina activity. Therefore learning and discrim-

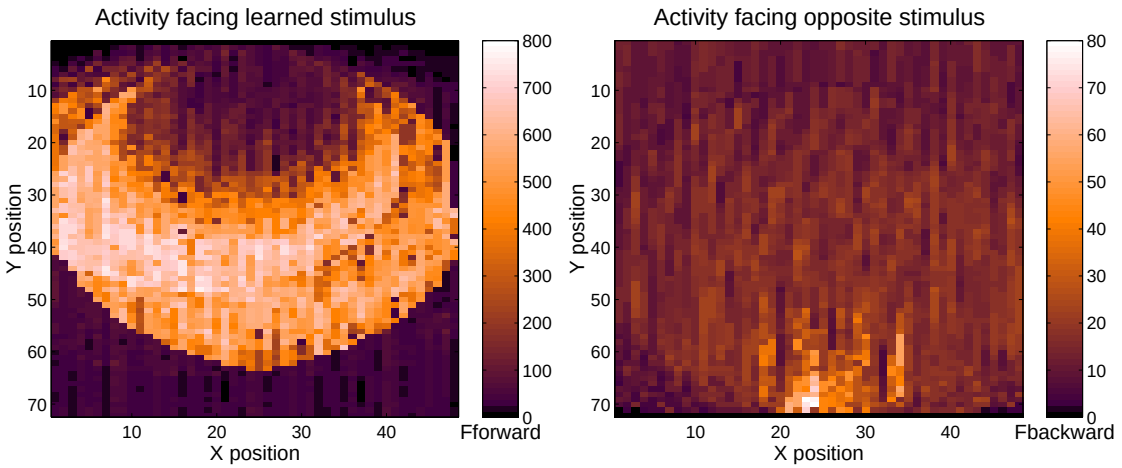

Figure 9.6: Color map representing the activity of the retina as function of the position of the robot in the arena. The left figure is $Fforward$ (activity when facing the learned stimulus) and the right figure is $Fbackward$. Note that the scale is smaller for $Fbackward$ for better rendering.

inating real stimuli seem possible, even if the stimuli and the image acquisition rate differ from those used in chapter 8.

When the robot faces the learned stimulus and is very far from it the activity of the retina decreases. This is caused by the display occupying only a small portion of the robot field of view: only a limited number of neurons are stimulated and this decreases the activity of the network. When the robot faces a stimulus and is very close to it, the robot tends to see periodical flashes of light. Therefore it cannot discern the direction of motion of the stimulus and the activity of the retina also decreases. If the intensity of the stimulus is high enough (e.g. when the robot is very close to the display) the neurons might however still fire. This is can be seen for $Fbackward$ in the figure, but it also occurs for $Fforward$, except that it is not visible on the plot for color scaling reasons. Therefore when the robot is very close or very far from the displays it may not be possible to know which stimulus the robot faces from the activity of the retina.

The angle at which the robot sees the stimuli (i.e. the position of the stimuli in the retina) and the robot own motion may also influence the retina activity. To assess this, the activity of the retina is measured in function of the robot angle while the robot is rotating with a constant speed, either clockwise (right rotation) or counterclockwise (left rotation), as illustrated in figure 9.7. The robot is placed in the center of the arena and learns a stimulus during 30 seconds (e.g. the left-moving stimulus in the figure). Afterwards learning is deactivated and the activity of the retina is measured during 100 ms while varying the robot angle. During the measure the robot rotates on the spot at different speeds. A rotation speed of +/- 1 means that e.g. the left wheel is set to +8mm/s and the right wheel to -8mm/s.

Figure 9.8 illustrates the activity of the retina in function of the angle of the robot, its rotation speed, and its direction of rotation. The results indicate that the speed and the direction of rotation of the robot have an influence on the network activity. At low

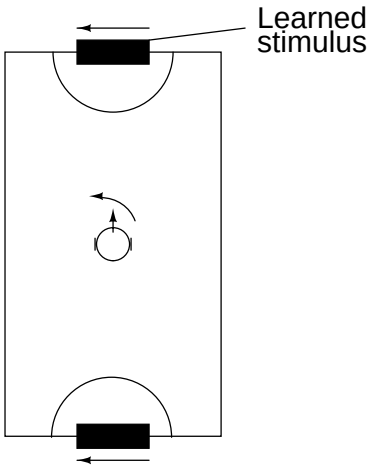

Figure 9.7: The robot is in the center of the arena and learns the upper left-moving stimulus. Afterwards the activity of the retina is measured while the robot rotates, for different rotation speeds.

rotation speed the retina reacts in the same way to the learned stimulus regardless of the direction of rotation of the robot. At higher speed the retina tends to react differently in function of the direction of rotation of the robot. In particular with one of the direction of rotation (right rotation) there is almost no activity in the retina when the robot faces the learned stimulus. This is caused by the relative motion of the stimulus in the field of view of the robot. When the robot rotates right the relative velocity of the learned stimulus (that moves left) in the field of view of the robot increases. With increased relative velocity the stimulus contributes during less time to the membrane potential of the neurons and thus they are less likely to fire. This can also be understood by considering the tuning curves introduced in section F.1.

In summary, the activity of the retina can easily be used to determine whether the robot faces the learned stimulus when the robot is static and located near the center of the arena. The motion of the robot may however hamper the detection of the learned stimulus by influencing the activity of the retina. Therefore the evolved neural controller needs to take this aspect into account to achieve successful homing behavior.

9.5 Evolution of the multi-cellular robot controller

Since we have verified that learning is possible with the retina, we now investigate the evolution of a multi-cellular controller for the robotic task described in section 9.2 with the morphogenetic system described in chapter 6.

The evolved circuit is composed of 8x8 cells, as illustrated in figure 9.9. The functionality of each cell is a spiking neuron capable of learning according to the model presented in section 9.3. All the neurons receive the vision input and they are all connected to their neighbors in a 5x3 neighborhood.

We evolve with the morphogenetic system the type of the neurons (excitatory or in-

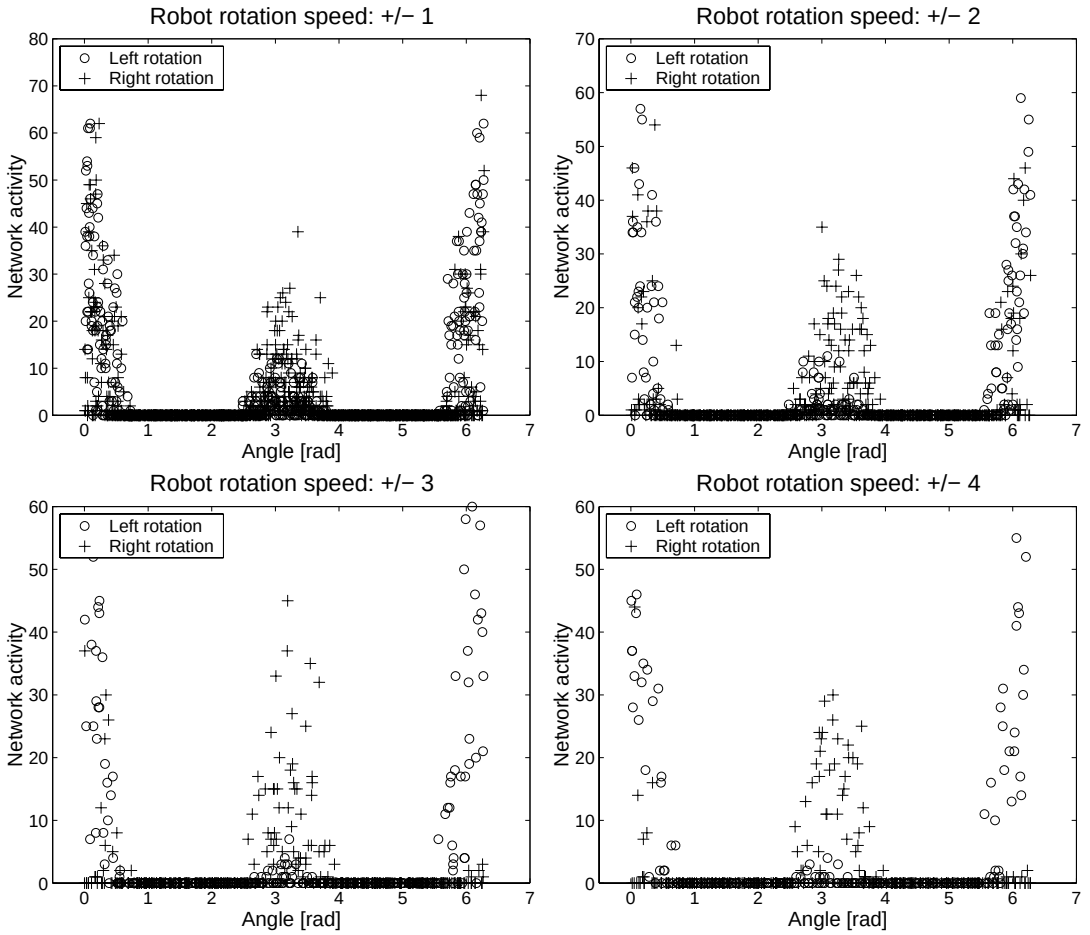

Figure 9.8: Retina activity in function of the robot angle for different rotation speeds. An angle of 0 or 2π means that the robot faces the learned stimulus. An angle of π means the robot faces the opposite stimulus. Each plot indicates the retina activity for left and right rotations. At low rotation speed (top left plot) there is no significant difference in activity if the robot rotates left or right. This can be seen by the similar distribution of "plusses" and "circles" in the plot. At higher rotation speed the activity of the retina tends to become different depending on the direction of rotation (see bottom right plot). When the robot rotates right, the activity of the retina is strongly reduced when the robot faces the learned stimulus (there are few "plusses" at the angle of 0 and 2π). On the other hand when the robot rotates left the activity of the retina remains similar to what is obtained at lower rotation speed (the number of "circles" at the angle of 0 and 2π rad is similar to what is obtained at lower speeds).

GA parameters	Genetic coding
Population: 50	Coding: morphogenetic system
Crossover: 20%	Diffusers: 16
Mutation: 1%	Expression table: 10 entries
Selection: rank (15 best selected)	Chromosome size: 296 bits
Elitism: 5 individuals are copied unchanged	

Table 9.1: Parameters of the genetic algorithm and of the genetic coding.

hibitory) and the motor connections. A neuron can can be either connected to the left or right motor, with either push or pull effect (i.e. the neuron makes the wheel rotate forward or backward when it emits a spike), or the neuron may not be connected to a motor at all. There are thus 10 predefined functionalities (2 signs multiplied by 5 possible motor connections).

The speed of the wheels is set after measuring the activity of the motor neurons during 100 ms. The speed is equal to the number of motor neurons spiking on the push input, minus the number of neurons spiking on the pull input, plus an evolved bias (in the range -4 to +3) which ensures that motion is possible even in the absence of neural activity. A speed of 1 corresponds to 8 mm/s. Vision is mapped to the network as with the retina described in section 9.3.

The genetic string of the morphogenetic system is evolved by a simple genetic algorithm. The parameters of the morphogenetic system and of the genetic algorithm are indicated in table 9.1. Since there are there are 10 predefined functionalities the number of entries in the expression table is 10. The total length of the genetic string is 296 bits (8 bits for the wheel bias followed by 288 bits for the morphogenetic system).

The fitness function rewards robots that home toward the learned stimulus and also, with a lesser weight, it rewards individuals that tend to move forward (this puts selective pressure on robots that explore their environment).

The instantaneous fitness f_i at each sensory-motor step is determined by the robot speed and its presence on the correct fitness area. If the robot is not on the fitness area, the instantaneous fitness is the sum of the normalized speeds of the wheels, when both spin forward. Therefore if the robot does not find the fitness area, but moves continuously at maximum speed its fitness is 1. If the robot moves over the fitness area the instantaneous fitness is 2, 3 or 4, depending on how close the robot is to the center of the fitness area. If the robot finds immediately the fitness area its fitness is 4. Formally:

$$f_i = \begin{cases} 2,\ 3\ or\ 4, & if\ robot\ is\ over\ the\ correct\ fitness\ area \\ \frac{v_L+v_R}{2}, & otherwise\ if\ v_L\ and\ v_R > 0 \\ 0, & if\ v_L\ or\ v_R \leq 0 \end{cases}$$

where v_L and v_R are the normalized speeds of the left and right wheels.

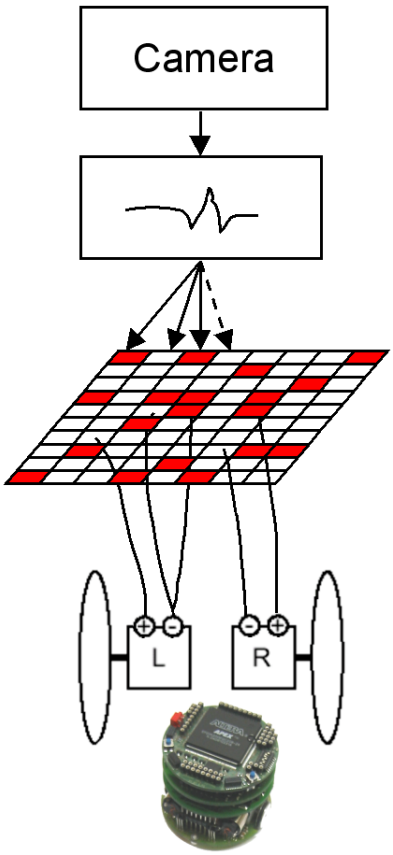

Figure 9.9: The learning-capable multi-cellular controller is a 2D network of 8 by 8 neurons. Each neuron is connected to its 14 neighbours (5x3 neighborhood). It is stimulated every millisecond by the input coming from the camera after preprocessing (image stabilisation): each column of pixels receives a contribution to the membrane potential which is proportional to the brightness of the corresponding pixel. The type of the neurons (excitatory or inhibitory, represented by the brightness of the cells in the picture) and their connections to the motors are evolved. Neurons connected to motors can have a "push" or "pull" effect, that is they make the wheels rotate forward respectively backward when they emit a spike. This is indicated by the + and - signs on the L and R boxes that represent the left and right motors of the robot.

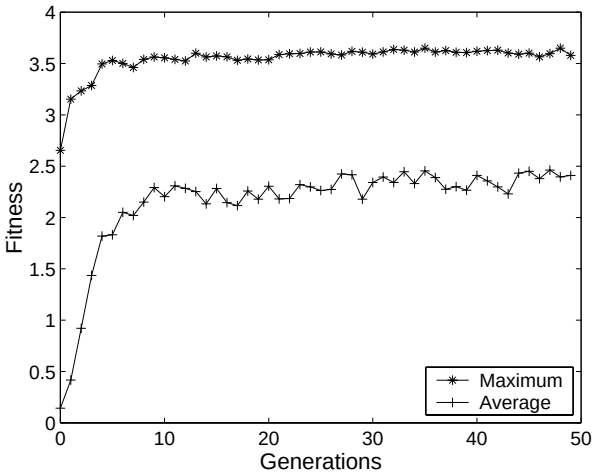

Figure 9.10: Evolution of the maximum and average fitness (average of 5 runs) at the homing task.

Since controllers should be equally capable of learning and homing towards the 2 stimuli that are in the arena, the fitness of the robot at learning and homing is measured on both stimuli. The robot first learns the left-moving stimulus and does the homing behavior. The fitness of the robot is the sum of the instantaneous fitness measured according to the above formula: $F^{Left-moving} = \sum_i f_i$. Then the robot learns the right-moving stimulus and does the homing behavior which results in the fitness $F^{Right-moving}$. To maximize the homing performance toward both stimuli and at the same time minimize the difference between the two homing behaviors we use the following overall fitness is defined:

$$F_{overall} = \frac{\sqrt{F^{Left-moving} \cdot F^{Right-moving}}}{max(1, |F^{Left-moving} - F^{Right-moving}|)}$$

The fitness of the robot is measured after learning which is done for 30 seconds with the robot placed in the center of the arena and facing one of the stimulus. After learning, the robot is randomly placed inside a 40x30cm area in the middle of the arena and the fitness of its behavior is measured during 90 seconds.

Figure 9.10 shows the evolution of the maximum and average fitness (average of 5 runs).

Tests of the best evolved controllers on the real robot show that the robots can successfully home toward the learned stimulus from a wide range of starting positions in the arena. The robots tend to make slow turns, and when they face the learned stimulus they move forward at a faster pace. Figure 9.11 shows typically observed homing behaviors. Analysis of the evolved controllers show that this behavior is obtained by setting the speed of one of the wheels with the evolved bias, whereas the other wheel is controlled by the activity of some of the neurons in the retina. This results in a rotation behavior until the robot faces the learned stimulus. In this case the activity of the retina increases, which makes the wheel controlled by the neurons rotate faster and gives straight motion.

The behaviors of the robots are quite robust for a wide range of lighting conditions: semi-dark room, room with sunlight, or room with neon light. The probable reason for

Figure 9.11: Typical homing trajectories followed by the robots after learning. The left plot corresponds to the trajectory after learning the upper stimulus; the right plot corresponds to the trajectory after learning the lower stimulus.

this robustness is that an active LED display is used: it makes a light whose intensity only weakly depends on external lightning.

The success of homing however depends on the starting position of the robot. To quantitatively assess this performance, the homing behavior of the robot is tested from a wide range of starting locations in the arena. The robot is successively placed on all the nodes of a grid which has a 1 cm grid spacing. The robot is tested 20 times with a random orientation on each node of this grid and it is allowed to move for 90 seconds. If after this time the robot finds itself over the fitness area next to the stimulus that was learned, the homing behavior is considered successful, otherwise it is not. The success rate is averaged on the 20 trials and illustrated by a color map in figure 9.12. When the robot starts close to a border it may fail to home toward the learned stimulus, either because it gets stuck in the wall or because it does not seem to perceive the stimulus. Also when the robot starts too close to the opposite stimulus, it sometimes homes on it instead of the learned one. This happens because the stimulus intensity increases when closer to the displays, and therefore also the activity of the retina, which misleads the robot.

Evolved controllers tend to have a slightly higher success rate when homing on one of the learned stimulus. This asymmetry is caused by the effect of the ego-motion of the robot that we evidenced in section 9.4: robots have a preferred direction of rotation which leads to one of the stimulus being more easily detected than the other.

9.6 Discussion

One of the challenges of this robotic task is that the own motion (ego-motion) of the robot may hamper the detection of moving stimuli. Yet the evolved neural controller is able to correctly find the learned stimulus despite the robot moving continuously. The strategy that is employed consists in limiting the speed of the robot, in particular the rotational speed. Insects such as fruit flies also rely on motion information to perform navigation. A commonly observed behavior is to navigate straight and then perform a rapid turn called a saccade. During the saccade the visual information has no influence on the behavior [157]. A similar behavior would be appropriate for the task described here, but it was not found by evolution.

There are some locations where the robot cannot perceive the target stimulus (figure

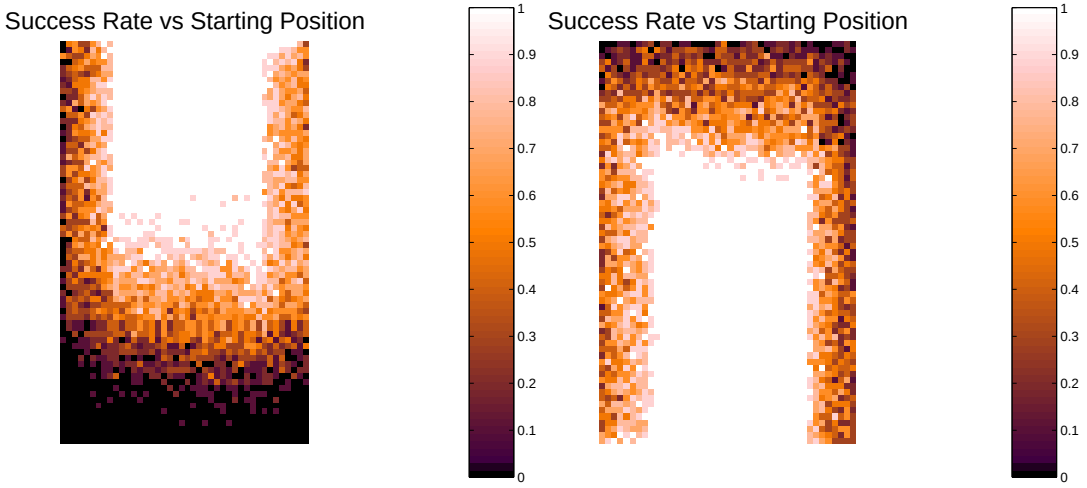

Figure 9.12: Color map of the normalized success rate of the homing behavior in function of the starting position of the robot in the arena. Brightest areas are locations where the robot has higher probability to home correctly on the learned stimulus. The left plot corresponds to the learning of the upper stimulus; the right plot corresponds to the learning of the lower stimulus.

9.6). However behavioral tests show that if the robot starts in one of those places, it is often capable of homing correctly (figure 9.12). The strategy employed by the evolved controller to achieve this consists in performing exploratory motions (e.g. large rotations) in the arena until a strong enough visual cue is found.

For the purpose of evolution we provided a family of functionalities with motor neurons that control the speed of the wheels by their activity. One drawback of relying on the activity of neurons is that the amplitude of the stimulus (e.g. the amount of light emitted by the displays) may influence the activity of the retina and therefore affect the homing behavior. In practice we ensured that both displays had the same intensity by using a common power supply.

We used predefined time constants for the neural model that we determined according to the speed of the stimuli in the arena. These time constants may however be genetically encoded. This may allow evolution to adapt the neural dynamics of the multi-cellular controllers to the type of environmental cues that the robot can perceive (e.g. stimuli moving at different speeds).

In this work we focused exclusively on vision. However there are other types of dynamic cues and the multi-cellular circuit may be extended to handle them. For instance sound cues may be used (time varying amplitude of the sound wave), as sound can be efficiently processed with spiking neurons [184].

The multi-cellular neural controller that we used in this chapter was simulated in software. The implementation of this circuit in the POEtic chip is only a matter of further technical work which may follow the lines of the implementation shown in chapter 5. The environment subsystem of the POEtic chip can execute the genetic algorithm to evolve the genetic string of the circuit. It can interface the multi-cellular neural network with the sensors and motors of the robot, and it can measure the fitness of the robot. The organic

subsystem of the POEtic chip can implement the morphogenetic system (see appendix D) and the multi-cellular neural network. In particular the neural model that we used in this chapter was implemented in the organic subsystem of the POEtic chip by Torres et al. at the Technical University of Catalunya, Barcelona [170, 171, 172]. The implementation allows to fit in the organic subsystem of a single POEtic chip an excitatory-excitatory synapse with the learning mechanism and the leaky membrane. By using time multiplexing and coupling several POEtic chips together, thereby extending the size of the organic subsystem, about 16 POEtic chips are necessary to implement the entire multi-cellular circuit of 64 neurons in hardware, assuming a reasonable operating frequency of 20 MHz[2].

9.7 Summary

In this chapter we showed the evolutionary morphogenesis of multi-cellular circuits capable of learning for a robotic task that required learning capabilities. We considered a robotic application in a dynamic environment, where environmental cues can only be understood through their temporal evolution. The objective was for a robot to learn a moving visual stimulus and then perform a homing behavior toward the learned stimulus while avoiding the other. Therefore, with the same controller, the learning of another stimulus changed the homing behavior of the robot accordingly.

We first verified that a multi-cellular circuit composed of spiking neurons with spike-timing dependent plasticity could learn and discriminate moving stimuli in the arena of the robot.

We then evolved a multi-cellular network with the morphogenetic system to control the robot. We evolved good controllers in terms of fitness and behaviors. In particular we tested the best evolved controllers in the real robot and we found that they could home toward the learned stimulus from most of the locations in the arena. We thus demonstrated that the morphogenetic system could be successfully used to evolve multi-cellular circuits to control the navigation of a robot in a task that requires learning capabilities. Finally we estimated that 16 POEtic chips are necessary for the hardware implementation of this multi-cellular circuit.

[2]The hardware implementation of the neural model requires 80 molecules of the POEtic chip for an excitatory synapse with learning and the leaky membrane. The time required to update one synapse, add it to the membrane and compare the membrane potential to the firing threshold is about 300 clock cycles [169]. Using a single POEtic chip to compute the 14 synapses of a neuron (i.e. the connectivity neighborhood of a neuron) through time multiplexing requires $14 \cdot 300 = 4200$ clock cycles for a neuron update. Since the network is updated every millisecond the chip must operate at a frequency of 4.2MHz to implement a single neuron in real-time. With a reasonable frequency of operations of 20 MHz, a single POEtic chip can update 4 neurons in real-time through further time multiplexing. Therefore the 64 neurons can be implemented in an array of 16 POEtic chips.

10 Conclusions

10.1 Summary and achievements

Conventional electronic circuits may lack applicability to ill-defined problems, robustness, or adaptivity to changing operating conditions. Electronic circuits inspired from principles observed in biology, so-called *bio-inspired* electronic circuits, have the potential to address these challenges.

The design of these circuits can take inspiration from the way biological organisms *evolve* over the generations, from the way they *develop* from a fertilized egg into multicellular organisms, and from their *learning* capabilities [146].

Until now bio-inspired hardware mostly focused on a single of these aspects: either evolution, development or learning. These three aspects are however complementary.

In this thesis we took the stance that to fully take advantage of bio-inspiration, electronic circuits should encompass all three mechanisms of evolution, development and learning. These circuits are called POEtic circuits, where POE stands for Phylogeny, Ontogeny and Epigenesis, which are respectively evolution, development and learning [178]. Although the concept of POEtic circuits in itself is not novel from this thesis [146, 178], the actual implementation and the applications of these circuits is a novel contribution of this thesis, as we will show below.

Conceptually these POEtic circuits, much like biological organisms, are multi-cellular circuits that evolve following the principles of selection and differential reproduction, they develop from a single cell and differentiate according to inter-cellular and environmental signals, and they learn during their lifetime. In comparison to conventional electronics, POEtic circuits are created automatically using evolutionary principles, even if only a partial or high-level specification of the problem is known. Development provides a complex genotype to phenotype mapping, that may lead to fault-tolerance or adaptive development in order to cope with environmental changes. Finally learning allows these circuits to memorize past events or adapt their response over time in order to improve their behavior. Since bio-inspired mechanisms may vary depending on the applications, these circuits are not directly implemented in silicon. They are obtained by programming a reconfigurable device, such as the POEtic chip, with the desired evolutionary, developmental and learning mechanisms.

This thesis dealt with the mechanisms required for the evolution of these bio-inspired electronic circuit.

In the literature we found that most evolutionary algorithms used to evolve electronic circuits do not exploit the complex dynamics of development mediated by gene regulation which is seen in biological organisms (chapter 2). They generally employ a direct genotype to phenotype mapping which seems to limit the scalability of the evolutionary approach to more complex circuits. In addition, since the genotype to phenotype mapping is static, there are no cellular or environmental interactions during development that could provide fault-tolerance or dynamic reorganization of the circuit in order to cope with environmental changes.

We argued that to fully realize the potential of POEtic circuits, an evolutionary system is required that combines both a genetic encoding and a more complex genotype to phenotype mapping provided by a developmental system. In particular our objective was to develop an evolutionary system with better evolvability and scalability than direct genetic encodings and that allows inter-cellular or environmental interactions during development. We called this evolutionary system the *morphogenetic system*.

While reviewing developmental systems applied to electronics, we made explicit different categories of developmental systems by introducing a novel classification which is based on characteristics of their hardware implementation (chapter 3). This classification allowed us to highlight one category of developmental system that is largely unexplored, and which we decided to consider for the morphogenetic system. This category consists of developmental systems that are implemented in hardware alongside the circuit under development, that operate continuously throughout the "life" of the circuit, and that allow a distributed, or cellular, implementation. Hardware and cellular implementation brings fast genotype to phenotype mapping. This, together with continuous operation, allows inter-cellular and environmental interactions to be taken into account during development.

In order to implement mechanisms of evolution, development and learning, we described a multi-cellular architecture and a cell architecture that allows a flexible combination of these mechanisms (chapter 4). The cell architecture is not an original contribution of this thesis [178], but we contributed to demonstrate how it fits in a reconfigurable device such as the POEtic chip, and we showed its applicability by evolving a multi-cellular circuit in this chip that approximated Boolean functions and that controlled the navigation of a mobile robot (chapter 5). With respect to the overall objective of this thesis, this step was important since the morphogenetic system assumed this architecture later on. In addition, the circuit we implemented is capable of growth (a simplified mechanism of development) and differentiation starting from a single cell. The growth mechanism did not originate from this thesis, but we contributed by explaining how it could be extended and serve as a foundation to implement self-repairing and self-replicating circuits in hardware.

On the basis of the architecture introduced previously we developed the morphogenetic system, a genetic encoding and developmental system inspired by the mechanisms of gene expression and cellular differentiation in biological organisms (chapter 6).

One of our main concern, which is also one of the originality of our approach, is that we explicitly designed a system that is computationally simple in order to be efficiently

implementable in hardware. In the literature, developmental systems often mimic up to some extent biological development. This usually leads to more complex systems that are not adequate for hardware implementation.

Another originality of the morphogenetic system is that it makes minimal assumptions on the circuits that are evolved. Other than assuming they are multi-cellular, it only requires local communication between neighboring cells. This allows this system to be very general: it can be applied to any circuit composed of low-level as well as high-level cell functionalities. This is important in order to evolve different type of circuits, such as circuits capable of learning. In comparison, many developmental systems applied to electronic circuits tend to operate directly on the configuration bits defining the circuit in a particular type of reconfigurable device. Instead our approach relies on functionalities predefined by the user. This allows it to be technology independent.

We implemented the morphogenetic system in hardware and we showed that, as desired, few resources are needed for its implementation and that it is fast, allowing development in constant time regardless of the size of the circuit.

We systematically tested the proposed morphogenetic system in increasingly complex applications. We first investigated its performance in terms of evolvability and scalability by evolving structures of differentiated cells (chapter 6). We found that the morphogenetic system allowed to evolve a wide range of regular as well as irregular structures, and that it provided better scalability to larger structures in terms of fitness than a direct genetic encoding used as a reference. Furthermore we showed that the dynamics of the morphogenetic system allowed to recover these structures of differentiated cells even at high fault rates. This last point particularly highlights the benefits of a developmental system running continuously within the cells of the circuit.

Afterwards we evolved functional multi-cellular circuits, although not yet capable of learning. Cells of these circuits implemented the functionality of spiking neurons. We demonstrated these circuits in tasks of pattern recognition and robot control (chapter 7). We found that the morphogenetic system outperformed a direct genetic encoding in a comparative test. We embedded the multi-cellular spiking controller in hardware on a mobile robot. This demonstrated the hardware suitability of the spiking neural model that we selected. More importantly, it also demonstrated the suitability of the whole concept of multi-cellular circuits used as hardware controllers for mobile robots.

Finally we used the morphogenetic system to evolve multi-cellular circuits capable of learning, thereby considering POEtic circuits which combine the three aspects of bio-inspiration that we mentioned earlier: evolution, development and learning. These circuits were composed of spiking neurons with a learning rule known as spike-timing dependent plasticity. We used these circuits to learn and discriminate the direction of motion of synthetic moving stimuli, and we demonstrated that the morphogenetic system could evolve these circuits to improve their learning performance (chapter 8). In a comparative test the morphogenetic system outperformed a direct genetic encoding in this task.

Building on these results we evolved multi-cellular spiking networks to control the navigation of a robot in a task that required learning capabilities (chapter 9). In this task a robot provided with vision had to learn a moving visual cue and afterwards it had to take a specific action when it encountered the learned cue in the environment. In particular,

with the same controller, the behavior of the robot changed when another cue was learned. With this application we demonstrated that spiking neurons could be used to learn moving visual cues and control the navigation of a robot according to these cues, and that learning at the synaptic level of the network could induce learning at the behavioral level of the robot. This robotic task is novel from an evolutionary robotics viewpoint, and it is also novel in the framework of POEtic circuits, since it demonstrates in a single application the advantages of the combination of evolution, development and learning in multi-cellular circuits. In particular these last two applications may not have been possible without learning in POEtic circuits.

In summary, the results that we obtained in a wide range of applications tend to confirm the generality of the morphogenetic and its better performance in comparison to conventional direct genetic encodings. The hardware implementation of the morphogenetic system shows that this can be achieved even with few hardware resources dedicated to the developmental system.

10.2 Further research directions

This thesis contributed to open several research directions, and we highlight the most important ones below.

Improved evolutionary morphogenesis

Although we obtained good results with the morphogenetic system, it is limited by the fact that the signaling and expression mechanisms are hard-coded. This implies that some phenotypes can not be genetically encoded by the morphogenetic system, but it is also one of the reason for the simplicity of the morphogenetic system and its suitability for hardware implementation. In comparison, more complex developmental systems rely on an evolved cell program that controls the signaling and expression mechanisms and therefore they may be more general.

The morphogenetic system can however be improved in several ways. For instance variable diffusion ranges may allow more efficient evolution of phenotypic structures by letting long range signals shape large structures while signals of shorter range take care of local details. These diffusion ranges could be evolved. Evolution may also be used to adapt the number of diffusers to the size and complexity of the phenotype. These modifications may improve the performance of the morphogenetic system with only a small impact on its complexity, therefore keeping with our philosophy of a simple evolutionary system.

Another research line may be to consider how new cell functionalities can be created or modified by the evolutionary process for use by the morphogenetic system.

Finally the morphogenetic system assumes that the connectivity of a cell is part of the cell functionality. This point was dictated by the reconfigurable devices available when this research started, which required the connections between components to be planned at *design-time*. Therefore the connectivity patterns of cells had to be predefined

before evolution. However these last years saw the development of the reconfigurable POEtic chip which addresses this point. The POEtic chip allows connections to be created dynamically at *run-time*. We already exploited this features to evolve the connectivity between cells in chapter 5, but not yet in the framework of the morphogenetic system. The morphogenetic system may be further extended to genetically encode the cell connectivity separately from the cell functionality.

In this thesis we used genetic algorithms to evolve the genetic string of POEtic circuits. Other search algorithms may however be considered and future work may investigate and compare different search algorithms (e.g. simulated annealing, tabu search).

Dynamic reorganization

Dynamic reorganization was partly discussed in chapter 5 where we described how a multi-cellular circuit capable of growth could be extended to perform self-repair by letting it reorganize at run-time to avoid faulty cells and use spare cells instead. Implementing this self-repair mechanism in hardware remains to be done.

Dynamic reorganization of POEtic circuits may also occur at the level the developmental mechanism within the morphogenetic system. Future work may investigate environmental or morphological interactions during development with the morphogenetic system. These interaction may allow adaptation to new environments or to changes in circuit morphology, e.g. when sensors or actuators are connected to the circuit. In particular the morphogenetic system has been designed in order to accommodate such environmental or inter-cellular interactions, that can be mediated by "chemicals" or signals diffused by the environment or special cells (e.g. sensors or actuators). Alternatively mechanisms similar to developmental growth processes implemented in self-reconfiguring modular robots may be investigated [168].

Morphogenetic fault tolerance

We showed that the dynamics of the morphogenetic system could provide fault tolerance to patterns of differentiated cells. This aspect was however not explored in hardware. It remains to be seen how this can be done, and in which context the dynamics of a developmental system can outperform traditional fault-tolerance mechanisms (e.g. triple modular redundancy).

Hardware support for bio-inspired mechanisms

In this thesis we programmed a reconfigurable device (the POEtic chip) to implement POEtic circuits. While this chip already has some support for bio-inspired mechanisms (e.g. self-reconfiguration and dynamic routing, see chapter 4), future reconfigurable devices could include, after careful analysis, more specific mechanisms in silicon.

For instance silicon could be devoted explicitly to store the genetic string of the circuit, to perform its integrity check (i.e. to detect if it is corrupted), and potentially also to transfer it to neighboring cells to implement efficiently the growth of a multi-cellular

circuit from a single cell. Silicon could also be devoted to the developmental system. For instance the diffusion of signals or the matching of signals with a functionality to express, if the morphogenetic system is taken for example, could be translated in silicon. Translating some of these mechanisms in silicon reduces their size in comparison to an implementation on a reconfigurable device. Therefore more complex circuits could be implemented within a given silicon area.

In addition we believe that the characteristics of silicon should be exploited to the maximum, and in particular also its analog characteristics.

Analog hardware allows to implement spiking neurons (i.e. the functionality of cells that we considered in this thesis) with a few MOS transistors in subthreshold mode [112]. In the same way gene regulatory networks, that are the basis of development in biological organisms, are also efficiently implemented with a few transistors [156]. Noise that occurs in analog circuits may not be an issue as it also occurs in biological neurons [186]. Furthermore advent in floating gate technologies nowadays allows to store analog values and design adaptive or learning circuits that are the core of bio-inspired systems [62].

The advantages of digital devices for bio-inspired systems should thus be weighted with the potentially more compact implementations that analog devices may offer. To conclude on a more general note, the future of reconfigurable devices for bio-inspired hardware may well lie in mixed-signal reconfigurable chips, that offer digital and analog functionalities for different bio-inspired mechanisms. These devices may not be unlike field-programmable transistor arrays [154].

A POEtic chip

This appendix complements chapter 4 and describes in more details the POEtic chip.[1] The architecture of the POEtic chip is illustrated in figure 4.6. It is composed of two subsystem, an environment and an organic subsystem. These two subsystems are described below.

A.1 Environment subsystem

The environment subsystem contains a 32-bit CPU, with 32 registers and 32 types of instructions. It has instructions for random number generations and bit manipulation that may be used in evolutionary algorithms. All the instructions are executed in one clock cycle. The CPU architecture is a Harvard architecture: the CPU has has one dedicated memory bus for the program code, and another for the program data. An AMBA bus is used to interface the CPU with its peripherals. The CPU has communication peripherals (UART, I2C, SPI and parallel port) that can be used to connect to external devices (e.g. sensors or actuators). It has two timers that can be used to generate periodical events (e.g. reading sensors). In addition it has a hardware 16x16 bit Booth multiplier, that can be used to implement multiplications at high speed in hardware. Finally it contains the organic subsystem interface. This interface allows to access the configuration bits and the status and control bits of the organic subsystem. The organic subsystem is configured through this interface (e.g. to implement multi-cellular circuits).

A.2 Organic subsystem

The organic subsystem shown in figure 4.6 is detailed in figure A.1. It is composed of a molecular array and a routing array.

The molecular array is composed of locally interconnected reconfigurable logic elements called *molecules* that can be used to implement digital logic circuits. The routing array is composed of *routing units* (RUs) that are capable of dynamic routing. Digital

[1] The images in this appendix are courtesy of Yann Thoma.

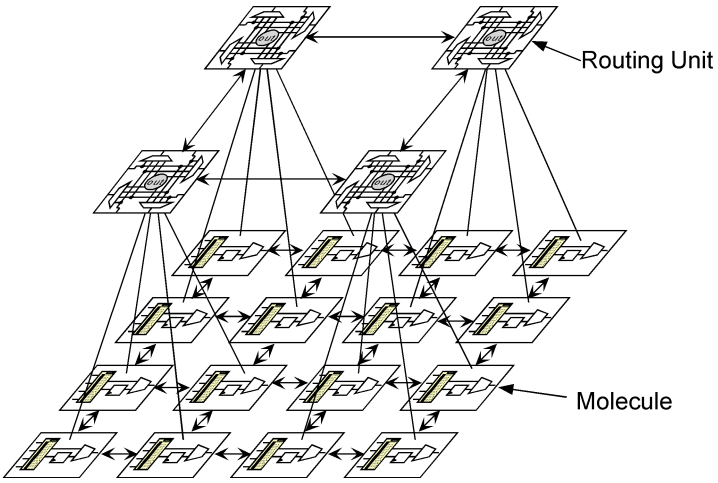

Figure A.1: The organic subsystem is composed of two layers. The lower layer is an array of molecules (reconfigurable logic elements) and the upper layer is an array of routing units which are used for long distance communication and dynamic routing.

logic circuits (such as POEtic cells) are implemented with these molecules and routing units.

Molecules are composed of a 16-bit register, that can be used to store data or implement a predefined function, a flip-flop that can be used to store one bit of data at run-time, and a switch box that is used to exchange data locally between molecules. Figure A.2 illustrates the molecular array with a close-up on the molecule and the switch box.

Even though a molecule is always composed of the same elements, it can operate in different modes according to its configuration. For example the 16-bit register can be used as a shift memory (i.e. the register stores one bit of data at each clock cycle) or as a lookup table (LUT). In this case control signals applied at the input of the register select which one of its 16 bits is provided at the output. The main operating modes of the molecules are described below and some of them are illustrated in figure A.3.

4-LUT The molecule is used as a 4-input LUT that can be used to implement a Boolean function of 4 inputs. The Boolean function is stored in the 16-bit register, and control signals applied at the input of the register select which bit is provided at the output.

3-LUT The molecule is used as two 3-input LUTs. This mode allows an efficient implementation of arithmetic operations (comparison, additions). For instance one LUT can compute an arithmetic sum and the other LUT can compute and propagate the carry signal to the next molecule.

Shift memory In this mode the molecule can be used as a memory. At each clock cycle the content of the 16-bit register is shifted and a new input bit is memorized. The input bit can come from another molecule, or it can be the output of the shift memory itself. In the latter case the molecule implements a rotating memory.

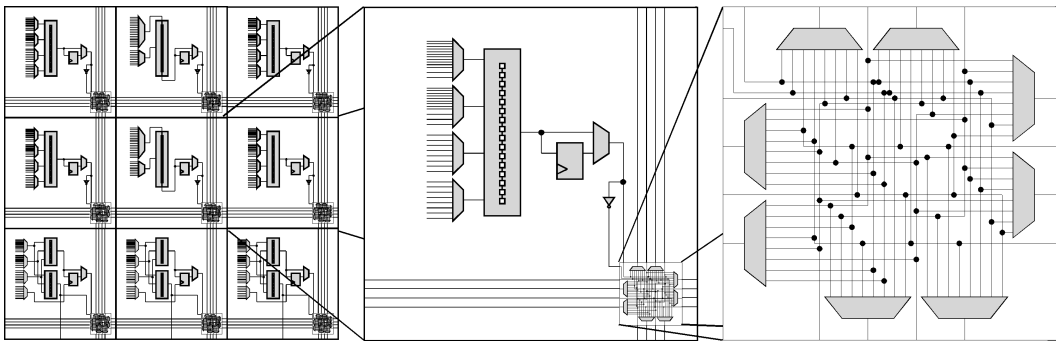

Figure A.2: The molecular array is shown on the left, with a close-up on a molecule in the center. The molecule contains a 16-bit memory that is illustrated by the vertical rectangle in the middle of the molecule. A switch box, at the bottom right of the molecule (detailed on the right), is used to exchange data between neighboring molecules using an array of interconnections that are programmed with the multiplexers shown in the figure.

Output The molecule sends data to the routing unit to which it is connected. In this way the molecule can send data over long distances to an input molecule via the routing layer.

Input The molecule receives data from an output molecule via the routing layer.

Configure Molecules in this mode are used for self-reconfiguration. A configure molecule can shift the content of its 16-bit memory into the configuration bits of a neighboring molecule, therefore changing its functionality.

The configuration of the molecule is described by 75 bits that include the content of the LUT, whether the output of the molecule is registered or combinational, whether the molecule accepts to be locally enabled by a neighboring molecule, whether the molecule is active on the rising or falling clock edge, which part of the configuration bits is modified in case of self-reconfiguration (e.g. to limit the reconfiguration to the 16-bit register, to the molecule functionality or to reconfigure the entire molecule), and finally the configuration of the switch box.

The routing array is composed of locally connected routing units which are capable of dynamic routing (figure A.4). A routing unit is connected to a group of four underlying molecules. The functionality of the routing unit is given by the input or output molecule in this group. For instance if there is one input molecule, the routing unit becomes an input, and if there is one output molecule the routing unit becomes an output. Routing units are passive when there are no underlying input or output molecules.

Routing units situated at the border of the array are connected to pins of the POEtic chip. Therefore signals on the pins can be accessed from the molecular array via the routing layer. Furthermore the chip is designed in such a way that several chips can be interconnected pin-to-pin to achieve a larger array of reconfigurable logic. In this case routing can be done across several chips via the border routing units.

Routing units support dynamic routing that builds connections automatically and at run-time. Dynamic routing relies on identifiers that are used as the addresses of the routing

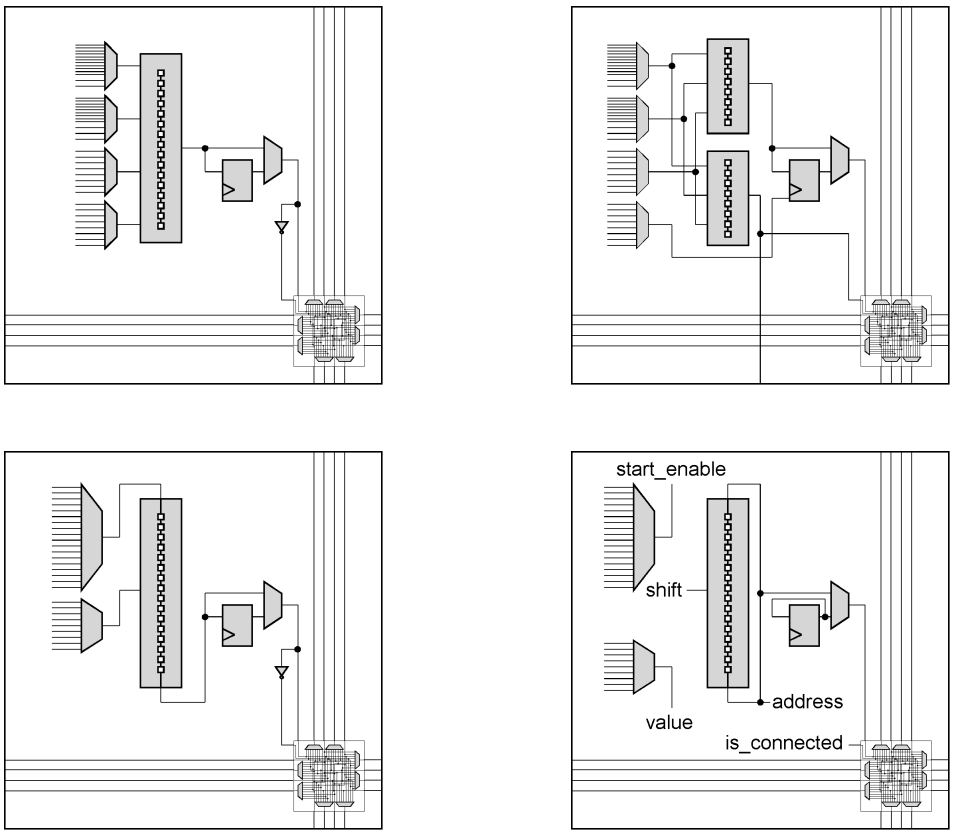

Figure A.3: Molecules can be configured to operate in different ways. The 4-input LUT (top left) is used to implement a Boolean function of 4 inputs. The 3-input LUT (top right) is used to implement two Boolean functions of 3 inputs (the 16-bit register is split in two registers of 8 bits). The shift memory (bottom left) shifts the content of the register and accepts a new input bit at each clock cycle. The output mode (bottom right) is used to send signals on long distances via the dynamic routing layer to another molecule.

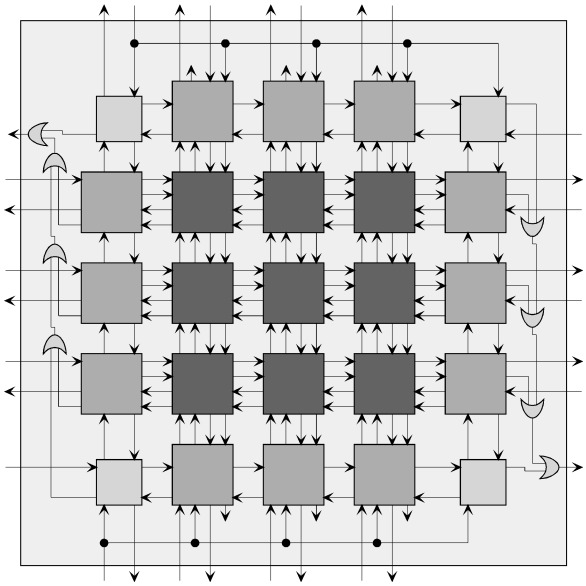

Figure A.4: The routing layer contains an array of locally connected routing units. The routing units are capable of dynamic routing: they have an address and can be inputs or outputs. When the routing process starts, logic within the routing units is used to establish a connection between input and output routing units that have the same address. Routing units on the border of the array are connected to the pins of the chip.

units. Those identifiers are stored in the 16-bit register of the corresponding input or output of molecule. A breadth first search algorithm implemented in the routing units is used to find a connection between input and output routing units that have the same identifier. Connections can also be changed, added or removed at run-time by locally reconfiguring the input or output molecules. For instance, by changing the content of the 16-bit register containing the identifier, the source or destination of a connection is changed.

Figure 4.7 illustrates the dynamic routing mechanisms by showing several connections between input and output molecules.

B POEtic tools

To simplify development on the POEtic chip we developed a software emulator of the POEtic CPU, which we called POEticIDE. It interfaces with the organic subsystem simulator called POEticMOL developed by Yann Thoma [162] to allow the co-simulation of the POEtic CPU and the organic subsystem (i.e. the reconfigurable logic). This tool was used to simulate and debug the applications described in chapter 5 before implementing it on the prototype of the POEtic chip. This appendix describes the CPU emulator and how it interfaces with the POEticMOL software.

B.1 POEtic CPU emulator

The simulation of VHDL code using standard tools (e.g. ModelSim) results in chronograms representing the temporal evolution of signals used in the circuit. The usefulness of this information is limited when complex programs are simulated.

Therefore an emulator of the CPU was designed based on the VHDL model of the CPU. This emulator is called POEticIDE (figure B.1). It executes CPU programs by the mean of a software model which decodes instructions one by one and updates the variables representing the CPU state (registers, memory content, etc.) accordingly. The CPU programs execute faster when the CPU is emulated than when the more complex VHDL model is simulated. Speedups compared to the VHDL simulation in the order of 10 to 100 were observed, even though the emulator is not fully optimized.

The emulator provides a graphical user interface which shows the status of the CPU. The code window shows the instructions in the program memory, together with their opcodes and the corresponding comments that were placed in the source assembler file. Breakpoints can be set and the code can be executed line by line or continuously until a breakpoint is reached or the execution is stopped manually. The register window and the memory window show the content of the registers and of the CPU memory, highlighting the entries which were changed by the last instruction. A console displays information relative to the simulation.

The emulator can import object files from the WinTim32 meta-assembler which was customized for the POEtic CPU [162]. The emulator can export the program in a VHDL ROM file for subsequent verification with a VHDL simulation. Export in the COE format

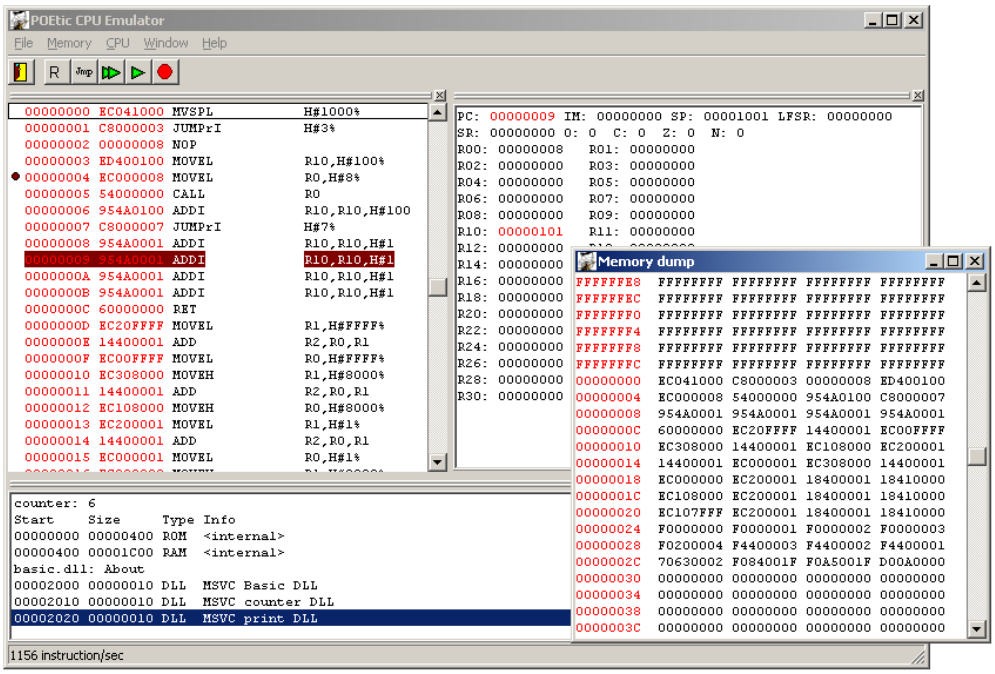

Figure B.1: The POEtic CPU emulator (POEticIDE), showing assembler code, the registers, and the memory content.

which is used by the Xilinx memory synthesis tools is also supported. This can be used to initialize the content of the program ROM when synthesizing the CPU on an FPGA.

Plugins, in the form of DLLs, can be included to emulate memory mapped peripherals. When the CPU reads or writes memory locations which correspond to the address space of a plugin the corresponding function of the DLL is called. This allows to emulate new peripherals of the system without having to modify the emulator. A peripheral emulating an UART (Universal Asynchronous Receiver/Transmitter) was implemented using this mechanism. When characters are written to its memory address, they are displayed in the console of the emulator. This gives a convenient mean of displaying program information from the assembler code in a way which is fully compatible with a hardware system using a real UART. Further DLLs were written to interface the emulator with the organic subsystem simulator. This is explained in the following section.

B.2 CPU and organic subsystem co-simulation

The CPU emulator is interfaced with the organic subsystem simulator called POEticMOL developed by Yann Thoma and described in more detail in [162]. This allows the co-simulation of CPU programs and of the reconfigurable logic of the POEtic chip.

Figure B.2 illustrates how the CPU emulator POEticIDE is interfaced with the organic subsystem simulator. The CPU emulator software contains the POEtic CPU code interpreter, the user interface and a console which can receive text information such as debug

messages. The CPU interpreter calls specific DLLs when the POEtic programs access to pre-configured memory locations. In the figure there are DLLs to simulate a robot, write to files on disk, print information in the console of the emulator, and eventually interface with the organic subsystem.

The organic subsystem interface is mapped at the same address range in the emulated POEtic CPU as in the hardware system described in chapter 5. Therefore circuits and programs can be simulated or executed on the real chip without code modification.

This organic subsystem interface allows POEtic programs to reconfigure the organic subsystem (e.g. to load new circuits in evolvable hardware experiments), to read and write the inputs and outputs of the organic subsystem, and also to read and write the status and control words of the organic subsystem. The status word indicates whether a dynamic routing operation is in progress or whether there are congestions. The control word includes the circuit chip enable and the reset and routing reset signals.

The organic interface DLL communicates by a pipe with the organic subsystem simulator, specifically the POEticIO DLL of the POEticMOL software. The pipe is used to exchange configuration information, status and control words and I/O data. To allow synchronized co-simulation of the CPU and the organic subsystem, a synchronization mechanism is implemented by events between the CPU emulator and the POEticVHDL DLL that is at the core of the organic subsystem simulator. In addition the POEtic CPU programs can control whether POEticMOL updates the display of the organic subsystem at each clock cycle. Deactivating the display allows faster simulation.

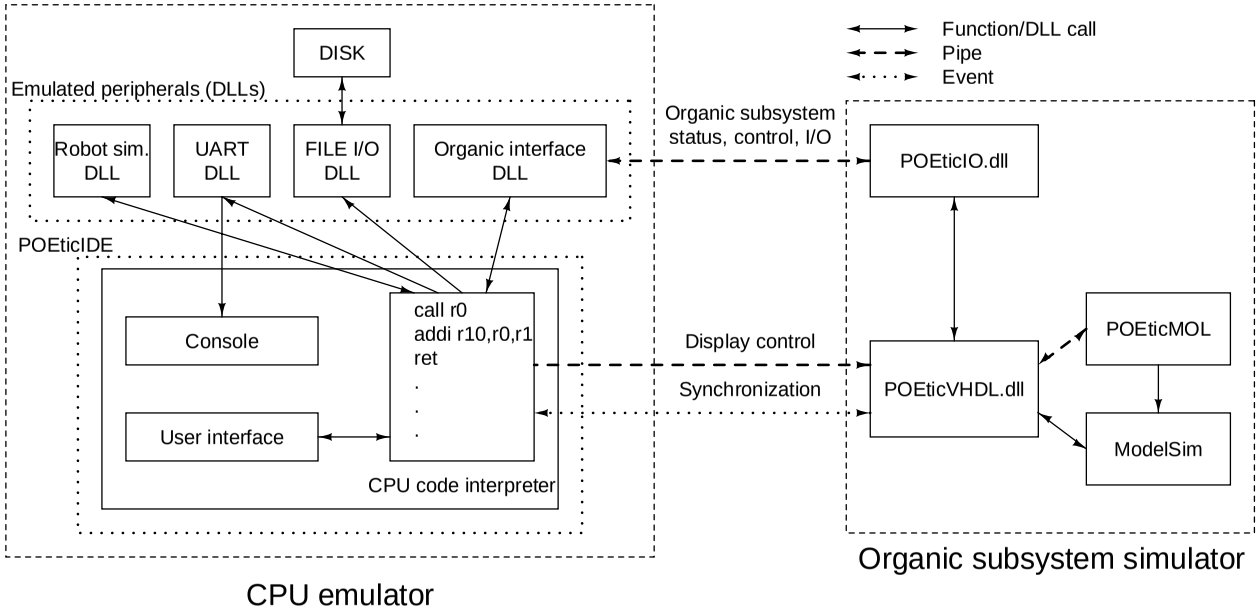

Figure B.2: Interface between the CPU emulator and the organic subsystem simulator. The arrows indicate the direction in which information is exchanged. The CPU code interpreter emulates the POEtic CPU and calls the appropriate DLLs when memory read or write operations occur in their address space. DLLs include a robot simulator, a file access DLL which is used to create and write data into files from the emulated POEtic CPU, an UART which displays data in the console of the emulator, and eventually the organic interface, which is used to communicate with POEticMOL, the organic subsystem simulator. The organic interface allows to reconfigure the organic subsystem, read and write its status and control words, and read and write its inputs and outputs. Synchronization between the CPU emulator and the organic subsystem simulator is implemented by events.

C Phenotypic complexity and morphogenetic system parameters

This appendix complements chapter 6. It shows the influence of the parameters of the morphogenetic system on the complexity of phenotypes randomly generated by the morphogenetic system. We consider the complexity of phenotypes likely to be generated by the morphogenetic system as type of representational bias. Knowing the effect of the parameters of the morphogenetic system on the complexity may help selecting appropriate parameters for evolution.

We consider the case where the morphogenetic system is used to evolve phenotypes to resemble predefined target patterns as in chapter 6. Here we use binary phenotypes (family of two functions, e.g. black and white cells). Unless otherwise noted the parameters of the morphogenetic system are the same as in section 6.3.

The parameters which affect the phenotypic complexity include the number of diffusers, number of signal types and number of entries in the expression table (i.e. several entries in the expression table may map to the same functionality). We mean by phenotypic complexity an indication of the structural irregularity of the phenotype in terms of pattern sizes and shapes. A quantitative measure of the complexity may be the Kolmogorov measure of complexity or the compressibility [98]. The latter measure consists in running a compression algorithm on the phenotype. The size of the compressed phenotype is an indication of its complexity (irregular phenotypes tend to be less compressible).

Here we consider compressibility with the Lempel-Ziv algorithm [138] as a measure of complexity. To verify that it is related to the structural complexity in terms of size and shapes of patterns in the phenotype, we compare it with another measure that is based on structural characteristics of the phenotype. This measure is the number of *blobs* in the phenotype. A blob is a computer vision term that describes a connected region (or object) in an image. Two pixels of same color that touch along the horizontal or vertical axis are considered part of the same blob [51].

The number of blobs and the size of the compressed phenotypes is measured for a large number of randomly generated binary phenotypes of various size (from 8x8 up to 64x64) with various parameters of the morphogenetic system (from 1 to 1024 diffusers, from 2 to 8 entries in the expression table and from 1 to 16 type of signals). Figure C.1 shows that there is a strong correlation between the number of blobs in a phenotype and its compressed size. Therefore compressibility is a reasonable indication of the structural

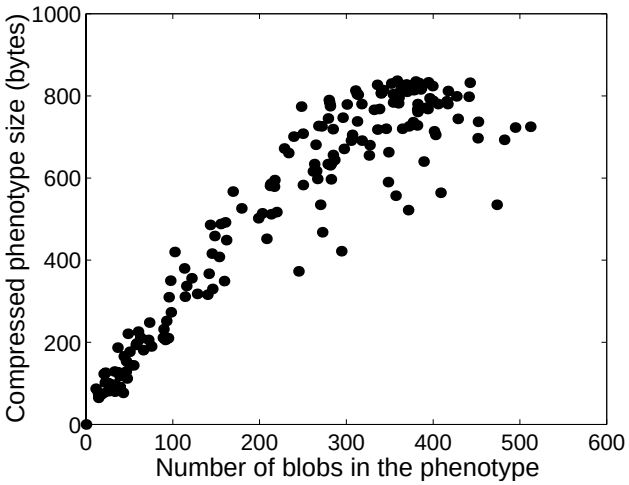

Figure C.1: The phenotypic complexity in bytes is plotted againts the number of blobs in the phenotype for many randomly generated phenotypes. The figure illustrates the correlation between these two measures of phenotypic complexity.

complexity of the phenotypes.

The bias of the morphogenetic system toward phenotypes of different complexity is measured by generating random binary phenotypes with different parameters of the morphogenetic system and measuring the complexity of the phenotypes by using the compression algorithm. All the data presented below are averages obtained on 25 random binary phenotypes.

Figure C.2 shows the effect of the number of diffusers on the relative complexity of phenotypes of different sizes. A small number of diffusers limits the complexity of the phenotype. With increased number of diffusers the complexity also increases. However, with too many diffusers, cells all tend to have signals of maximum intensities. Therefore they express the same functionality, which reduces the complexity. The number of diffusers for which the complexity is highest depends on the phenotype size.

Figure C.3 illustrates the effect of the number of signal types and number of diffusers on the complexity in the case of a 16x16 array (results with other array sizes are similar). For low number of diffusers, increasing the number of signal types decreases the complexity (see the figure with a single diffuser). This happens because chemical layers tend to have fewer or no diffusers, hence signal intensities are more likely to be uninitialized. Therefore cells express with higher probability identical functionalities corresponding to uninitialized signals. When increasing the number of diffusers, having more signal types allows more complex combinations of signals in the cells and this generates higher structural complexity in the phenotypes (see the figure with 1024 diffusers). In this case, if the number of signal types is restricted, large parts of the phenotypes are saturated with signals of maximum intensity. Therefore functionalities corresponding to saturated signals are expressed with higher probabilities, which reduces the complexity. This is the case for 16 and more diffusers in the figure.

There must be at least one entry in the expression table for each functionality in the

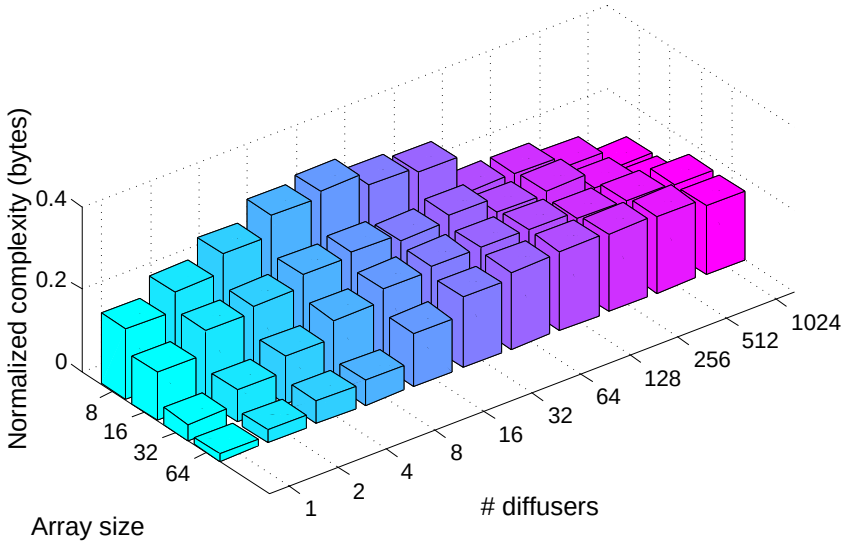

Figure C.2: Relative complexity of phenotypes in function of the number of diffusers for different array sizes. The complexity is normalized by the phenotype area for better scaling in the figure. The other parameters of the morphogenetic system are: 4 types of signals and 2 entries in the expression table (one for each cell functionality). The number of diffusers influences the phenotypic complexity: too few or too many number of diffusers lead to low complexity (i.e. cells mostly contain either uninitialized signals or signals of maximum intensity), whereas with an average number of diffusers the complexity is maximized.

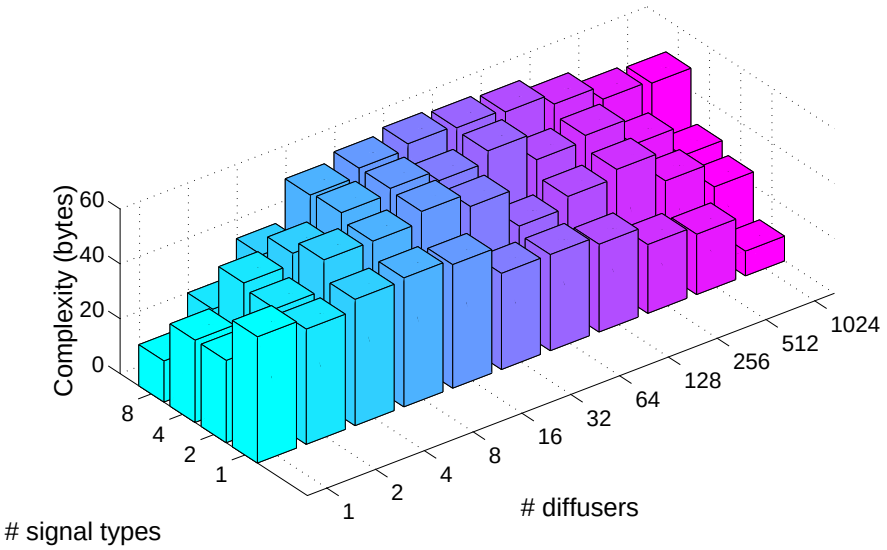

Figure C.3: Complexity in function of the number signal types and number of diffusers in a 16x16 array with two functionalities (2 entries in the expression table). With few diffusers the complexity decreases when increasing the number of signal types because cells with uninitialized signals predominate, and thereby they express the same functionality. With many diffusers increasing the number of signal types increases the complexity because more combinations of signal intensities are possible.

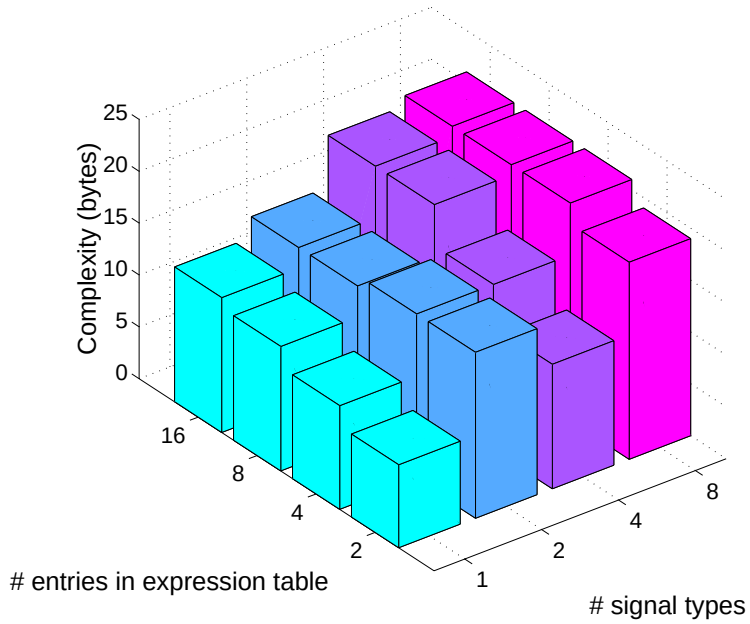

Figure C.4: Complexity in function of the number of signal types and the number of entries in the expression table. The phenotype is an array of 8x8 cells; the number of diffusers is 128. The number of entries in the expression table varies from 2 to 16 (i.e. there are from 1 to 8 entries in the expression table for each cell functionality). The phenotypic complexity tends to increase with more entries in the expression table.

family that is used, but it is possible to have several entries in the expression table that correspond to the same functionality. Figure C.4 shows that on average the complexity of the phenotypes tends to increase when several entries in the expression table map to the same functionality.

In summary, we have shown that the number of diffusers, array size, number of signal and number of entries in the expression table have a coupled effect on the complexity. Although knowing which parameters to use may still need empirical tests, we speculate that these parameters might be selected according to the foreseen complexity of the target phenotypes. The practical application of this however remains to be investigated.

D Hardware implementation of the morphogenetic system

This appendix complements chapter 6 and details the hardware implementation of the morphogenetic system on the POEtic chip for use in multi-cellular circuits.

The morphogenetic system forms the mapping layer of the cells of these circuits. We refer to the implementation of the morphogenetic system in a cell as a *morphogenetic element*. The morphogenetic element is illustrated in figure 6.19 (right). It has 4 inputs by which it receives signal intensities from neighboring morphogenetic elements, one output by which it sends its own signal intensities to neighboring morphogenetic elements, and one function output that indicates the functionality that the phenotype of the cell must take.

Section D.1 describes architectural considerations leading to the implementation of the morphogenetic system and section D.2 details the hardware implementation.

D.1 Architectural considerations

We focus on an implementation of small size, and the size depends of a number of architectural choices concerning the element interconnection, the communication format and the type of arithmetics.

Each morphogenetic element has to receive 4 signals from each of its 4 immediate neighbors. Each signal is encoded on 4 bits, with a fifth bit indicating whether the signal is valid. Therefore the state of all the signals in a morphogenetic element is represented by 20 bits (4 signals times 4 bits plus 4 valid bits). The morphogenetic element thus has to receive 20 bits from each of its four neighbors. A parallel transfer of those bits would require 80 inputs (20 bits times 4 neighbors) and 20 outputs, hence at least 100 routing units. However the POEtic chip contains 36 routing units, and therefore a more compact implementation of the signal exchange must be used. By transferring serially all the 20 bits describing the state of the signals in a morphogenetic element on a single line, only 4 inputs (1 input per neighbor) and one output are needed.

The communication format defines how data are serialized on a single line. Serial transmission is easily implemented by connecting a 16-bit shift memory molecule to the output of the morphogenetic element. To simplify the control logic to send, receive and process the signals it is better to start the transmission on multiples of 16 clock cycles.

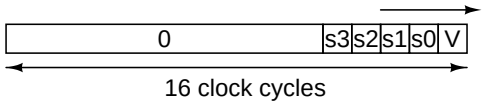

Figure D.1: Ordering of bits during the 16-clock cycle long transfer of a signal. The first bit transferred is the valid bit, afterwards the 4-bit signal intensity is transferred starting by the least significant bit. The remaining bits are 0's.

In this implementation the 4 signals are transferred in 4 16-clock cycle long sequences. Each sequence transfers one signal. During the sequence the valid bit is transferred first, followed by the 4-bit signal intensity starting from the least significant bit. The remaining bits are 0's. Figure D.1 illustrates the communication format.

Computation can be done in parallel or sequentially (serial arithmetics). The latter leads to more compact implementations but it is slower as a single bit of the result is computed at each clock cycle. The molecules of the POEtic circuit are well suited for serial arithmetic. Shift memories can be used to efficiently store 16-bit numbers and a single 3-LUT molecule can do a serial addition, subtraction or comparison. For this reason the implementation uses serial arithmetics. Although the number of clock cycles for serial arithmetics depends on the bit length of the operands, it is best to extend the operation to a multiple of 16 clock cycles because this corresponds to a complete rotation of values stored in shift memories. It is convenient to define the term *molecular cycle* to mean 16 clock cycles.

D.2 Hardware implementation

The block schematic of the morphogenetic element is illustrated in figure D.2 and the complete implementation is shown in figure D.8, D.9 and D.10. The morphogenetic element is composed of two main parts: the signaling block and the expression block.

The signaling block handles I/O, the diffusion mechanism, and provides the signal intensities for use in the expression block. Interconnections between morphogenetic elements are implemented with the dynamic routing of the POEtic chip. The `cellular input` receives the signal intensity and the valid flag from neighboring elements over the dynamic routing layer. The signal that is used by the morphogenetic process (i.e. one of the valid incoming signals) is selected by `input select` and then decremented by `decrement-compare`. `Normalize` renormalizes the signal if it is invalid or below zero. Finally the signal intensity and the valid flag is stored in `diffusion memory`. At the same time as one signal is received, the `cellular output` sends the current intensity of the corresponding signal to neighboring morphogenetic elements. Once all the 4 signals are processed, the `diffusion memory` provides the signals intensities of the morphogenetic element to the expression block.

The expression block provides the `function` output of the element by sequentially comparing the signal intensities with each of the entries of the expression table. The content of the shift memory `expression table` is compared to the signal intensities in the

cell with the `Hamming distance`. This distance is compared to the current `shortest distance` by `compare distance`. If the current distance is shorter, then the shortest distance is updated and the `best function` found until now is updated with the ever increasing value of the `function counter`, that represents the functionality that the phenotype layer of the cell must implement. The `function` output is the output of a molecule that is locally connected to the phenotype of the cell.

The morphogenetic element continuously executes the developmental program. One developmental step consists of the sequential execution of the signaling and expression blocks. Figure D.3 illustrates the execution sequence and indicates the time necessary to complete each operation. The signaling block operates in two modes. During the first 5 molecular cycles it sends/receives the signal intensities to/from the neighboring morphogenetic elements and updates the signal intensities according to the diffusion rules. In the following 11 molecular cycles, the signaling block continuously provides at each molecular cycle the four 4-bit signal intensities (i.e. a sequence of 16 bits) to the expression block that does the matching process. The expression block alternates between two phases which implement the expression mechanism of the morphogenetic system. These phases each take one molecular cycle. The sequence of operations is detailed below.

1. I/O: The signaling block sends the intensity of its signals and its valid bits to connected morphogenetic elements. At the same time it receives those of its neighbors. Each signal is sent and received during one molecular cycle according to the format shown in figure D.1.

2. DIFF: With a delay of one molecular cycle from the reception of the signals from the neighbors, the intensity of the signal in the morphogenetic element is updated according to the diffusion rules. After 4 molecular cycles all the 4 signals are updated. In the following 11 molecular cycles, the newly computed signal intensities are provided to the expression block for the expression phase.

3. EXPR: The expression phase is executed to find the index of the entry in the expression table best matching the intensities of signals in the morphogenetic element. Expression consists of two molecular cycles for each entry in the expression table, plus two molecular cycles to initialize the process. Those molecular cycles are EXPR-0 and EXPR-1. During phase EXPR-0 the Hamming distance is compared with the shortest stored distance. During phase EXPR-1 the Hamming distance is computed, the shortest distance and the best entry are updated if needed, and the index in the expression table is incremented. EXPR-0-R and EXPR-1-R correspond to the initialization of the expression phase.

4. EXPREND: The index is transferred outside of the morphogenetic element for use by the phenotype layer of the cell.

A complete developmental step takes 16 molecular cycles, with 5 molecular cycles used for I/O and diffusion, 10 molecular cycles used for expression (4 entries in the expression table), and the last molecular cycle outputs the functionality of the cell.

Complete development, which requires 16 developmental steps, takes $16 \cdot 16 \cdot 16 = 4096$ clock cycles, regardless of the size of the circuit.

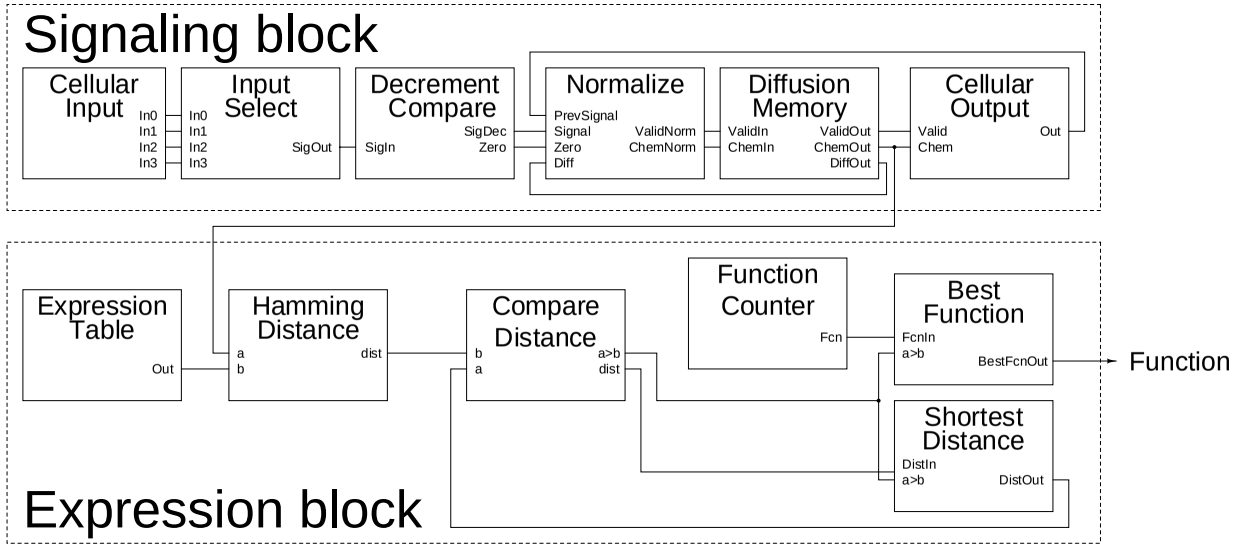

Figure D.2: The morphogenetic element is composed of two main blocks: the signaling block takes care of the inputs and outputs, and of the diffusion mechanism. The expression block finds the functionality to express in the cell according to the signal intensities.

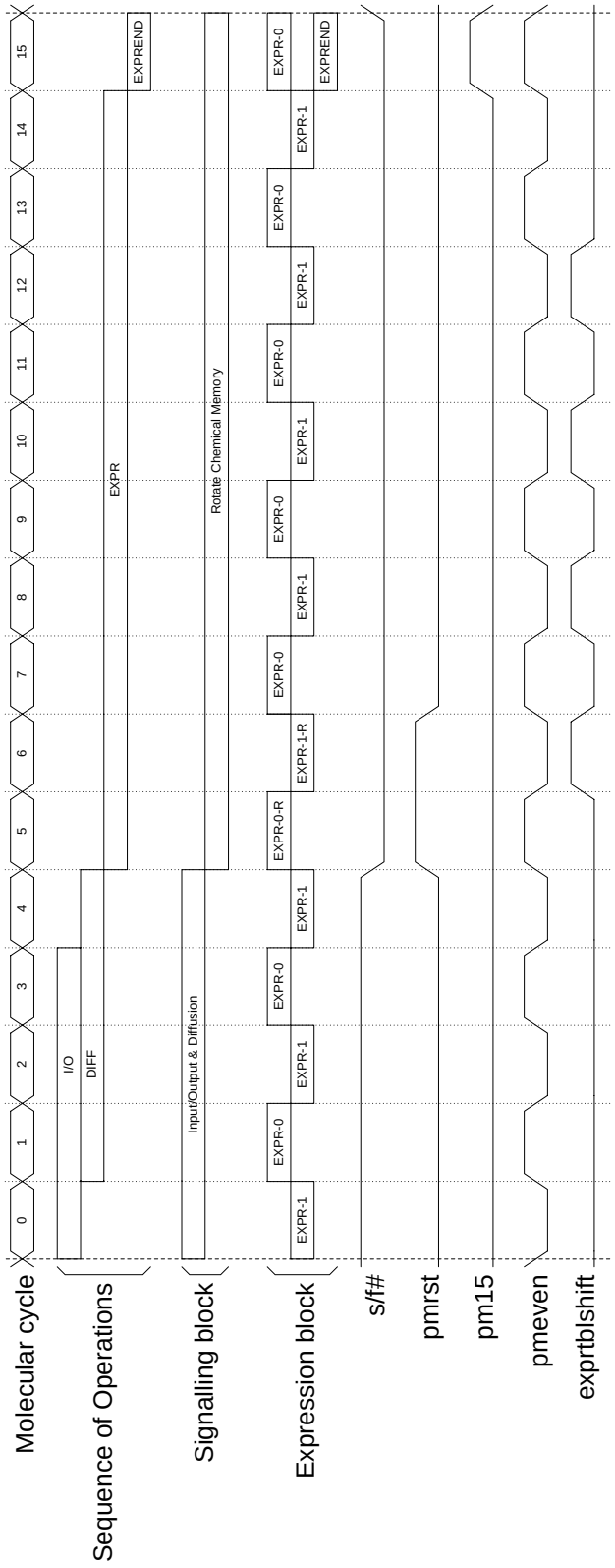

Figure D.3: Sequence of operations for one developmental step, with the corresponding control signals in the lower part.

Several control signals (see fig. D.3) manage the sequencing of the signaling and expression blocks. Control signals are efficiently implemented by memory molecules configured as rotating memories.

The molecular layout of the POEtic implementation is illustrated in figure 6.20. The morphogenetic element is implemented in 56 molecules. They are distributed as follows: 12 implement control signals (one more molecule than the minimum required is used to implement the same control signal to reduce the routing resources), 25 implement the signaling block (5 are used as inputs and outputs via the dynamic routing layer) and 19 implement the expression block (4 of them are memories to store the expression table).

A 3 by 3 array of cells containing only the morphogenetic element are is illustrated in figure D.4 after interconnection of these morphogenetic elements with the dynamic routing mechanism of the POEtic chip.

D.2.1 Description of the functional blocks

The functionality of the blocks composing the morphogenetic element are described below. All the functional blocks, including the sub-blocks forming the signaling and expression blocks (see figure D.2), perform the same sequence of operations periodically. This period is a multiple of a molecular cycle. In particular every sub-block has a periodicity of one molecular cycle. This means that, control signals being equal, each sub-block performs the same sequence of operation in all molecular cycles. Blocks can have a delay (latency) before data at their output reflects data at the input. Latency is also a multiple of a molecular cycle. A latency of 0 means that the block implements only combinatorial logic. When operations occur at specific clock cycles (e.g. a flip-flop is reset at clock cycle 0), the clock cycle is always relative to the beginning of the molecular cycle.

Data are exchanged among blocks serially during molecular cycles. Thus data exchanges consist of 16 bit packets, although only some of those bits may be significant. The serial format of the data is indicated as a sequence of 16 bits. Bits are exchanged on the rising clock edge, starting from the rightmost bit in the sequence.

The following notation is used to describe the bits exchanged among blocks. `si` is the `i`th bit of a signal intensity. `ti` is the `i`th bit of an entry of the expression table. `hi` is the `i`th bit of the Hamming distance. `Ii` is the `i`th bit of the index of an entry in the expression table. `v` indicates whether the signal is valid (1) or not (0). `D` is a bit indicating whether the morphogenetic element is a diffuser (1) or not (0). `0`, `1` and `X` represent the binary values 0, 1 and undefined respectively.

Signaling block

The signaling block handles the input/output and the diffusion mechanism. As illustrated in figure D.3 it operates in two modes. During the first 5 molecular cycles it sends and receives the signals and updates the signal intensities of the element according to the diffusion mechanism. In the 11 following molecular cycles, it sends continuously the chemical intensities to the expression block. The diffusion block is composed of the sub-blocks shown in figure D.2.

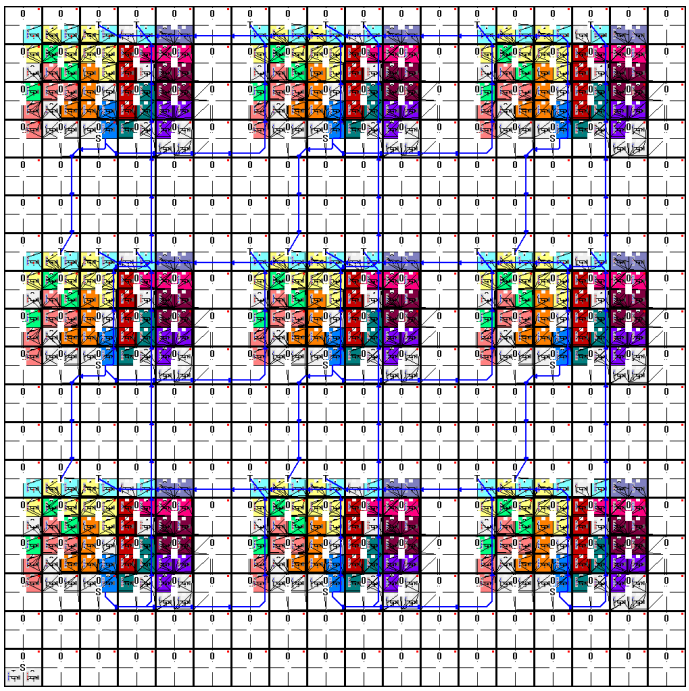

Figure D.4: A 3x3 array of cells containing the morphogenetic elements. The interconnections between the morphogenetic elements are done at run-time with the dynamic routing mechanism of the POEtic chip.

`Cellular input` receives the signals from neighboring morphogenetic elements. The valid bit indicating whether the signal is valid is received first. This bit is used by `Input Select` to select one of the valid incoming signal during the first clock cycle. `Input Select` outputs serially whether one of the inputs was valid and the corresponding signal intensity (`SigOut`). `Decrement & Compare` then in parallel compares the incoming signal to 0 (`Zero`), and decrements it by one (`SigDec`). `Normalize` provides the signal intensity (`ChemNorm`) and the valid bit (`ValidNorm`) for storage in `Diffusion Memory`. If the signal currently processed was already set (the valid bit of `PrevSignal` is 1), or if the molecular element is a diffuser for that signal (`Diff` is 1), then `Normalize` forwards that previously stored signal `PrevSignal` (i.e. once a signal intensity is set, it is not changed anymore), or the value of 15 (i.e. the maximum signal intensity if the element diffuses this signal). `Diffusion Memory` stores the signal intensities and the corresponding valid bits during the first 5 molecular cycles. In the following 11 molecular cycles it rotates the content of the signal memory to provide a 16-bit number (`ChemOut`) which is the concatenation of the four 4-bit signals. This number is used during the expression phase.

Figure D.5 summarizes the operations of the sub-blocks during a molecular cycle. Details regarding each sub-blocks are given below.

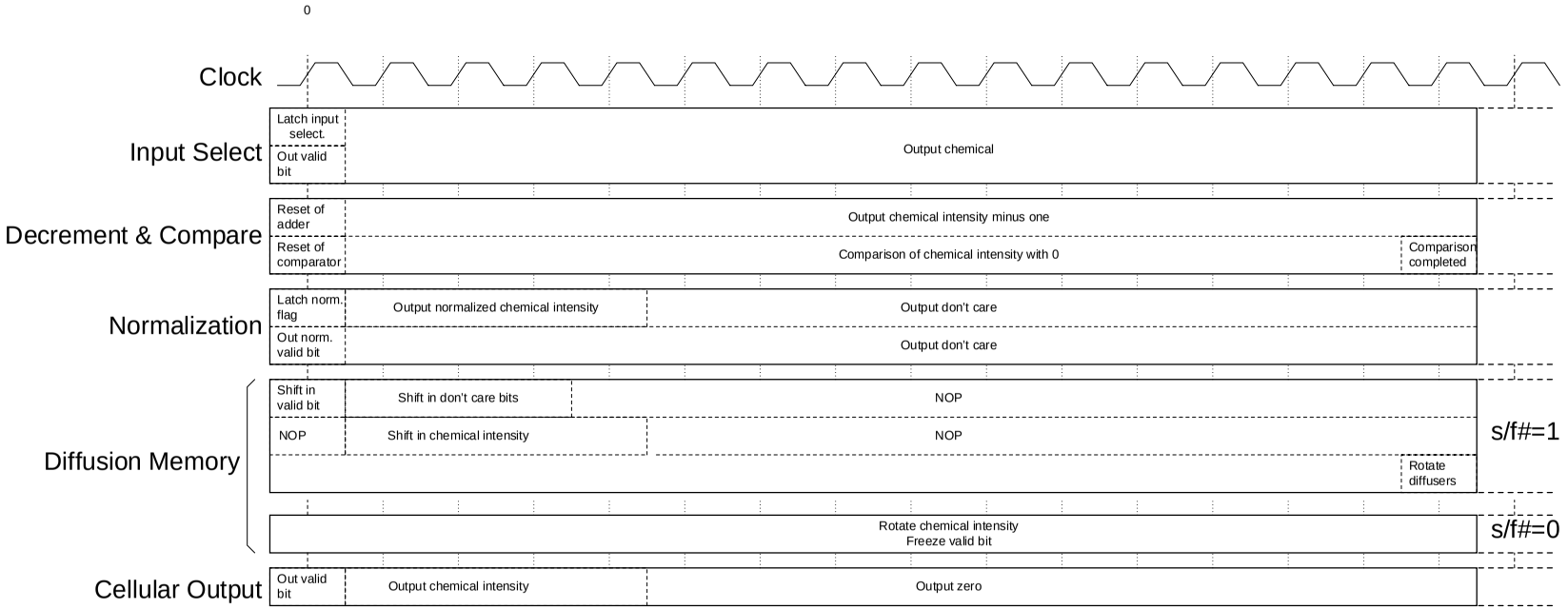

Figure D.5: The figure indicates what the sub-block composing the signaling block do during the 16 clock cycles of a molecular cycle. The sub-blocks are `Input Select`, `Decrement & Compare`, `Normalization`, `Diffusion Memory`, and `Cellular Output`. The `Diffusion Memory` sub-block operates differently depending on the signal `s/f#` shown on the right. Function NOP means no operation, i.e. the content of the registers of the block does not change.

Cellular Input

In	-															
Out	Inx:	0	0	0	0	0	0	0	0	0	0	0	s3	s2	s1	s0	v
Latency	0															

Four INPUT molecules provide the signals of other morphogenetic elements obtained through the dynamic routing layer. Those signals are sent the `Input Select` element.

Input Select

In	Inx:	0	0	0	0	0	0	0	0	0	0	0	s3	s2	s1	s0	v
Out	SigOut:	0	0	0	0	0	0	0	0	0	0	0	s3	s2	s1	s0	v
Latency	0															

This block acts as a multiplexer: it selects one of the valid inputs and directs it to the output. Molecules MuxA-1, MuxA-2 and MuxB act as a 3-input multiplexer. Molecules Sel0 and Sel1 generate the select signal for the multiplexer and molecules Sel0Latch and Sel1Latch store it during clock cycle 0 for the rest of the molecular cycle. The first output bit `v` indicates whether there is at least one input which has a valid signal. If not, this bit is cleared and the remaining output bits are meaningless.

Decrement and Compare

In	SigIn:	0	0	0	0	0	0	0	0	0	0	0	s3	s2	s1	s0	v
Out	SigDec:	0	0	0	0	0	0	0	0	0	0	0	s3	s2	s1	s0	v
	Zero:	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	Z
Latency	1															

The input signal is decremented by one by molecule SUB and sent to the `SigDec` output. At the same time the input signal is compared to 0 by molecule CMPZ: the `Zero` output indicates whether the input signal is equal to zero ($Zero = 0$) or not ($Zero = 1$). The latency of the comparison is 1 molecular cycle as all the input bits need to be processed before the `zero` output can be updated, whereas there is no latency for the subtraction. Therefore a delay is introduced in the `SigDec` output by molecule InputBuffer to ensure that this output is available at the same time as the `Zero` output.

Normalize

In	PrevSignal:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	v
	Signal:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	v
	Zero:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	Z
	Diff:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D
Out	ValidNorm:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	v
	ChemNorm:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	X
Latency	0															

This block provides the Diffusion Memory block with the new intensities of signals and the new valid bits of the cell that must be stored. This is done on two seperatate outputs: `ValidNorm` and `ChemNorm` respectively. `PrevSignal` is the signal intensity and valid bit that is currently in memory. The valid bit of `PrevSignal` (i.e. the first incoming bit) is stored by molecule ValidLatch in clock cycle 0. This bit controls the output molecule ChemMemMux1 for the rest of the molecular cycle: if the previous signal is valid, then the output signal is identical to the previously stored signal (i.e. the memory content

does not change). Otherwise the outputs are set by molecule Normalize and the memory content may change.

The output valid bit (`ValidNorm`) is set if `Signal` is valid or if the morphogenetic element is a diffuser (`Diff=1`) for that signal. The output signal `ChemNorm` is either the input signal or 15 if the morphogenetic element diffuses that signal or if the input signal is below zero `Zero=1` (normalization). Molecule ChemNorm stores at clock cycle zero whether normalization should occur. Molecule SignalDelay delays by one molecular cycle the signal coming from the Diffusion Memory block to compensate for the latency of the Decrement and Compare block.

Diffusion Memory

In	ValidIn:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	v
	ChemIn:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	X
Out	S/F#=0															
	ChemOut:	s15	s14	s13	s12	s11	s10	s9	s8	s7	s6	s5	s4	s3	s2	s1	s0
	ValidOut:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	DiffOut:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
	S/F#=1															
	ChemOut:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	X
	ValidOut:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	v
	DiffOut:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	D
Latency	4/0																

This blocks stores the four signal intensities, the corresponding valid bits and whether the morphogenetic element is a diffuser for any of the signals. Molecules ChemMem, ValidMem and DiffuserMem are used for this purpose respectively. Upon start of a developmental step, data in DiffuserMem is organized as follows:
`X X X X X X X X X X D3 D2 D1 D0 X` (rightmost bit is the MSB). `D`i indicates whether the morphogenetic element is a diffuser for signal i.

Data in ValidMem is organized as follows:
`X X X V3 X X X V2 X X X V1 X X X V0`. `V`i indicates whether signal i is valid.

Data in ChemMem is the concatenation of the four 4-bit numbers indicating the intensity of each signal.

This block is at the interface between the signaling block and the expression block and has two different functionalities depending on control signal `s/f#`. When `s/f#=1` (first five molecular cycles of a developmental step) it updates the content of the signal intensity and valid bit memories from the `ChemIn` and `ValidIn` inputs while providing the previously stored signal intensity and valid bit at its outputs. This is done by shifting the registers of the ChemMem and ValidMem molecules during 4 clock cycles in each of the first 5 molecular cycles.

When `s/f#=0` (remaining molecular cycles of a developmental step) the content of the ChemMem register rotates at each clock cycle to provide the expression block with the 16 bits representing the intensity of the four signals. Molecule ChemMemMux2 is used to feed the output of molecule ChemMem back to its input to do this rotation.

Latency of the block is 4 molecular cycle when `s/f#=0` (i.e. 4 molecular cycles are required to fill the Diffusion Memory) and 0 molecular cycles when `s/f#=1`.

Cellular Output

In	Valid:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	v
	Chem:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	X
Out	Out:	X	X	X	X	X	X	X	X	X	X	X	s3	s2	s1	s0	v
Latency	0															

An OUTPUT molecule is used to send signals to the neighboring morphogenetic elements through the dynamic routing layer. Molecule OutMux is used to direct to the `Out` the `Valid` input in clock cycle 0 and the `Chem` input in the remaining clock cycles. `Out` is the signal sent to the dynamic routing layer.

Expression block

The expression block looks for the index of the entry in the expression table which has the smallest Hamming distance to the signals in the morphogenetic element. This is a sequential process which iterates through all the entries of the expression table. For each entry in the expression table the following sequence of operations is executed:

1. Compute the Hamming distance between the expression table entry n and the signals in the morphogenetic element
2. Compare the distance with the shortest stored distance
3. Update the shortest distance with the new distance if it is shorter; at the same time update the index of the best entry with n
4. Increment the index n

Each of these operations takes one molecular cycle to execute, however some operations can be done in parallel, thus reducing the number of molecular cycles to 2 per entries in the expression table. Those two cycles are called EXPR-0 and EXPR-1 phases in the implementation. During phase EXPR-0 the Hamming distance is compared with the shortest stored distance. During phase EXPR-1 the Hamming distance is computed, the shortest distance and the best entry are updated if needed, and the index in the expression table is incremented.

The total execution time in molecular cycles for the expression block is 2 plus twice the number of entries in the expression table. The first two molecular cycles are necessary to reset the expression block. In the current implementation there are 4 entries in the expression table which gives 10 molecular cycles to complete the expression. Figure D.3 shows the expression block alternating between EXPR-0 and EXPR-1 phases. Expression is initiated by a reset (control signal `pmrst` in molecular cycles 5 and 6 (EXPR-0-R and EXPR-1-R in the figure) and at molecular cycle 15 the expression is completed and the best matching entry is provided (EXPREND in the figure).

The operation of the sub-blocks composing the expression block during the molecular cycles corresponding to phase EXPR-0 and EXPR-1 are illustrated in figure D.6. Details regarding each sub-blocks are given below.

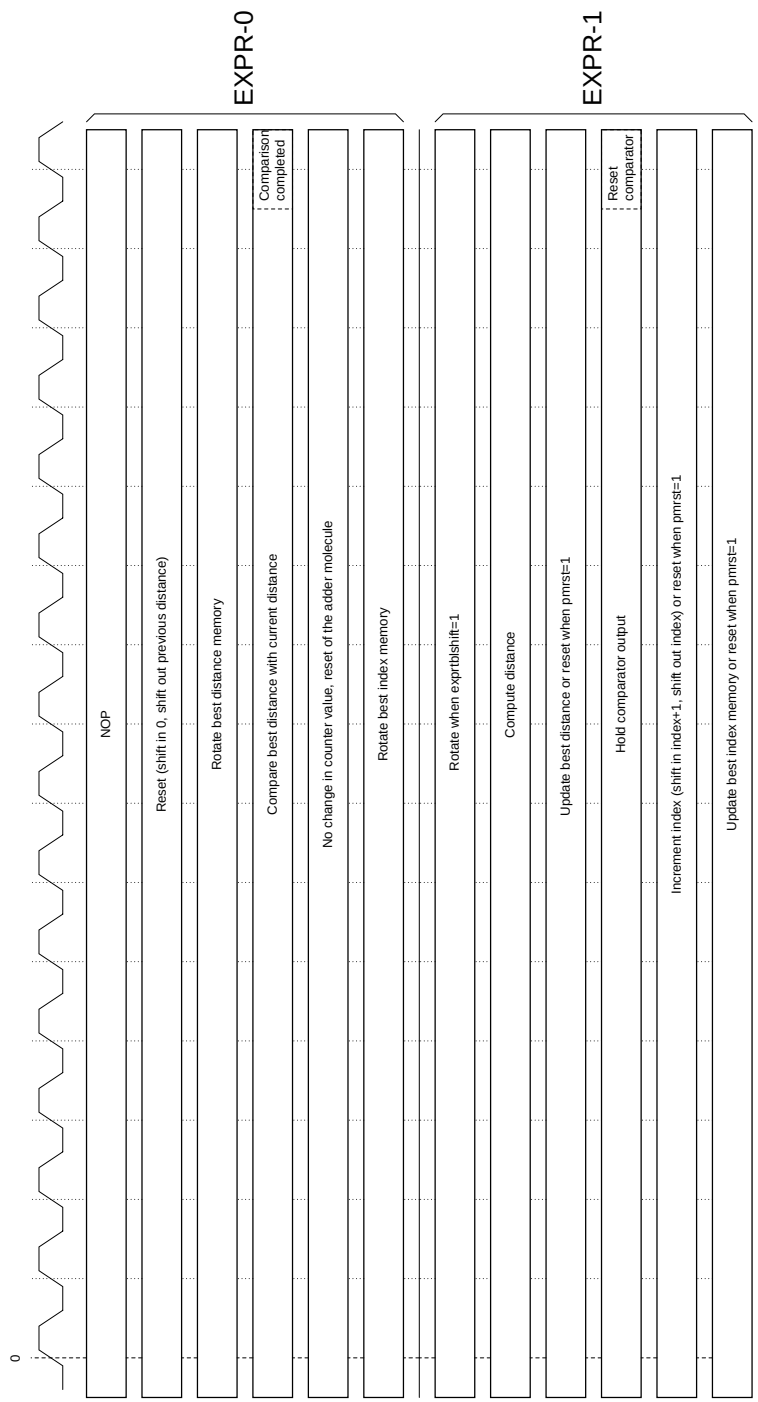

Figure D.6: Sequence of operations during the EXPR-0 and EXPR-1 molecular cycles for the sub-blocks of the expression block. Function NOP means no operation, i.e. the content of the registers of the block does not change.

Expression Table

In	-															
Out	Out:	t15	t14	t13	t12	t11	t10	t9	t8	t7	t6	t5	t4	t3	t2	t1	t0

Every entry of the expression table is composed of 16 bits and can therefore can be stored in a single MEMORY molecule. The entire expression table is stored in 4 molecules set-up as rotating memories (the output of last molecule is the input of the first one). Each time an entry of the expression table is required to compute the Hamming distance, the content of the expression table is rotated during a molecular cycle. Therefore at each clock cycle one bit of the expression table is sequentially provided at the output of the block. This allows efficient access to the expression table without any decoding logic. Signal `pmxtblshift` controls when the expression table needs to be rotated.

Hamming Distance

In	a:	s15	s14	s13	s12	s11	s10	s9	s8	s7	s6	s5	s4	s3	s2	s1	s0
	b:	t15	t14	t13	t12	t11	t10	t9	t8	t7	t6	t5	t4	t3	t2	t1	t0
Out	dist:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
Latency	1																

The MEMORY molecule Dist is used to store the Hamming distance between `a` and `b`. During phase EXPR-0 zeros are shifted in Dist, and the previously computed Hamming distance is shifted out. Then during phase EXPR-1 the Hamming distance is computed. Molecule XOR performs the exclusive or of `a` and `b`. Whenever its output is 1, a 1 is shifted in molecule Dist, otherwise no shift occur. At the end of the EXPR-1 cycle, the Dist memory contains a number of 1 equal to the Hamming distance between `a` and `b`.

Compare Distance

In	a:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
	b:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
Out	dist:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
	a>b:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	a>b
Latency	1																

During cycle EXPR-0 the values `a` and `b` are compared by molecules CMPMOL1 and CMPMOL2. `a>b` holds the result of the comparison at the end of the molecular cycle. Output `dist` reflects input `b` with a delay of one molecular cycle introduced by Dist-Buffer. This way both the result of the comparison and the distance are available in the same molecular cycle. During cycle EXPR-1 the output `a>b` keeps the result of the last comparison during the whole molecular cycle. The comparator is reset in clock cycle 15 of the EXPR-1 phase.

Function Counter

In	-															
Out	Fcn:	I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0

The function counter indicates which entry of the expression table is currently used for matching. MEMORY molecule IDXMem holds the counter and molecule IDXINC performs the serial increment. In the EXPR-0 phase the counter is not updated. In the

EXPR-1 phase IDXMem shifts out the counter value and shifts in the incremented value, or in case of reset (`pmrst=1`) shifts in zeros.

Best Function

In	FcnIn:	I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0
	a>b:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	a>b
Out	BestFcnOut:	I15	I14	I13	I12	I11	I10	I9	I8	I7	I6	I5	I4	I3	I2	I1	I0

MEMORY molecule BestIDX holds the index of the entry matching best so far the signal intensities in the morphogenetic element. In phase EXPR-0 the content of BestIDX is rotated and available on `BestFcnOut`. In phase EXPR-1 BestIDX shifts in a new index. This index is either 0 when `pmrst=1`, the output of BestIDX when `a>b=0` (no change in the index) or otherwise `FcnIn` (the index coming from the Function Counter block). Molecules BestIDXMUX1 and BestIDXMUX2 provide the appropriate new index.

Shortest Distance

In	DistIn:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0
	a>b:	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	a>b
Out	DistOut:	h15	h14	h13	h12	h11	h10	h9	h8	h7	h6	h5	h4	h3	h2	h1	h0

This block functions as the Best Function block. MEMORY molecule BestDist holds the shortest distance found so far. In phase EXPR-0 the content of BestDist is rotated and available on `DistOut`. In phase EXPR-1 the BestDist shifts in a new distance. This distance is either all ones when `pmrst=1` (longest distance possible) or the output of BestDist when `a>b=0` (no change in the distance) or otherwise `DistIn` (the distance coming from the Compare Distance block). Molecules BestMUX1 and BestMUX2 provide the appropriate new distance.

D.3 Control signals

A number of control signals are necessary to sequence the operations of the morphogenetic element. Those are provided by MEMORY molecules configured as rotating memories. Figure D.7 illustrates the control signals which are used in the morphogenetic element. There are two categories of control signals: those with a periodicity of one molecular cycle (upper half of the figure) and those which have a periodicity of 16 molecular cycles (a complete developmental step). The former are used sequence operations within a molecular cycle, for example to reset the flip-flop of adder or comparator molecules, while the latter are used to sequence the operations of the expression and diffusion blocks (e.g performing 5 molecular cycles executing the diffusion block, resetting the expression block or indicating a that the function output of the morphogenetic element is valid).

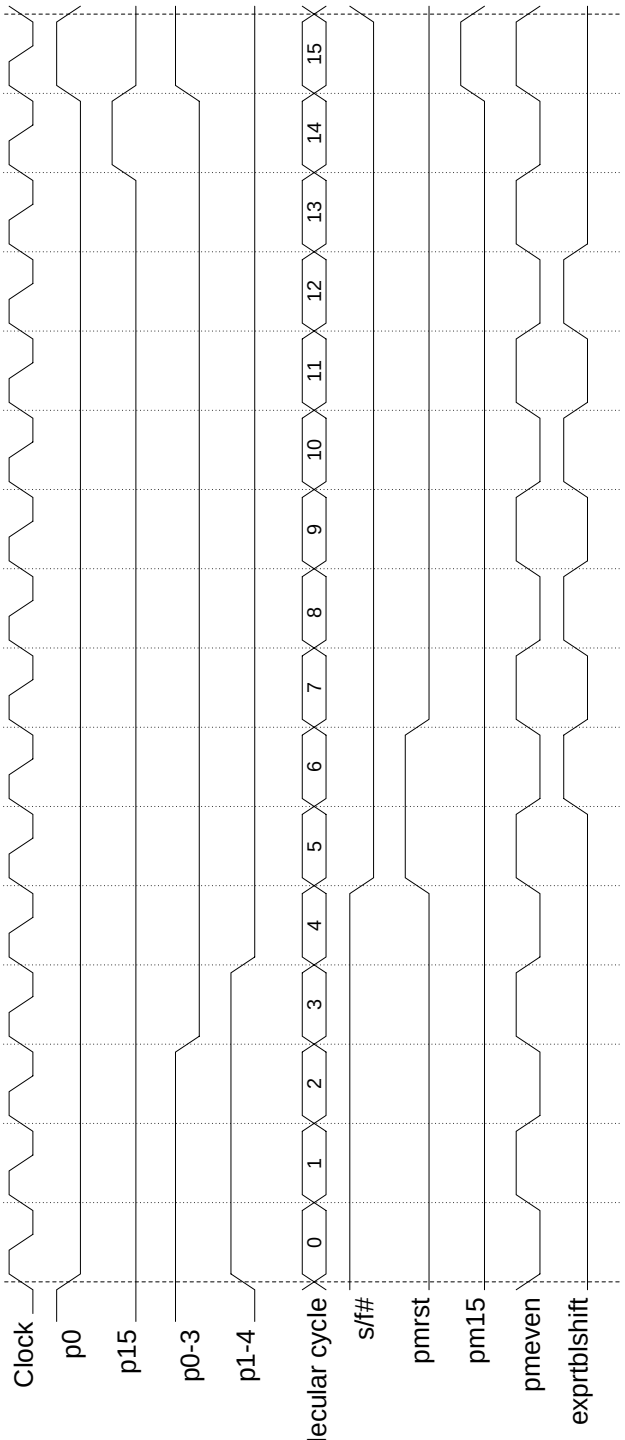

Figure D.7: Control signals used to sequence the operations of the morphogenetic element. The upper half of the figure shows control signals which have a periodicity of one molecular cycle. The lower half of the figure shows the control signals which have a periodicity of a developmental step (16 molecular cycle).

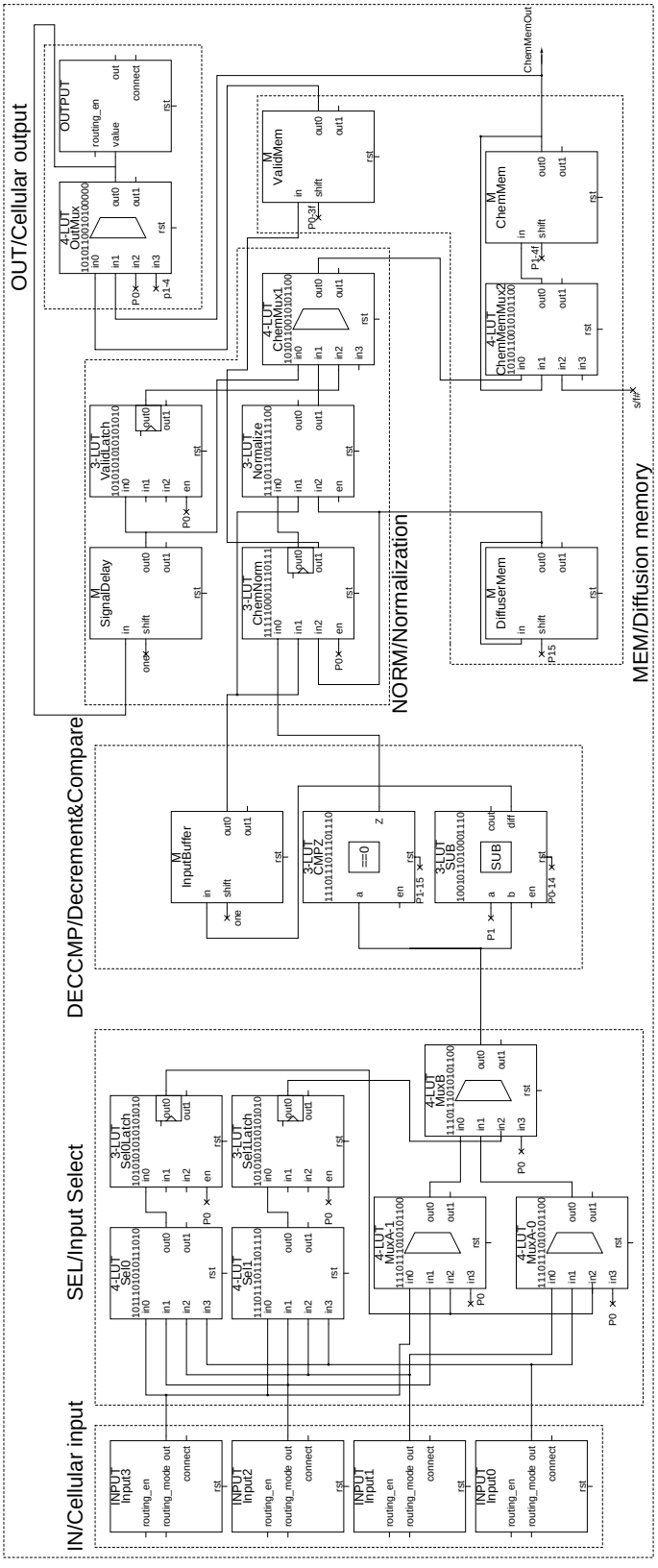

Figure D.8: Schematic of the molecular implementation of the signaling block.

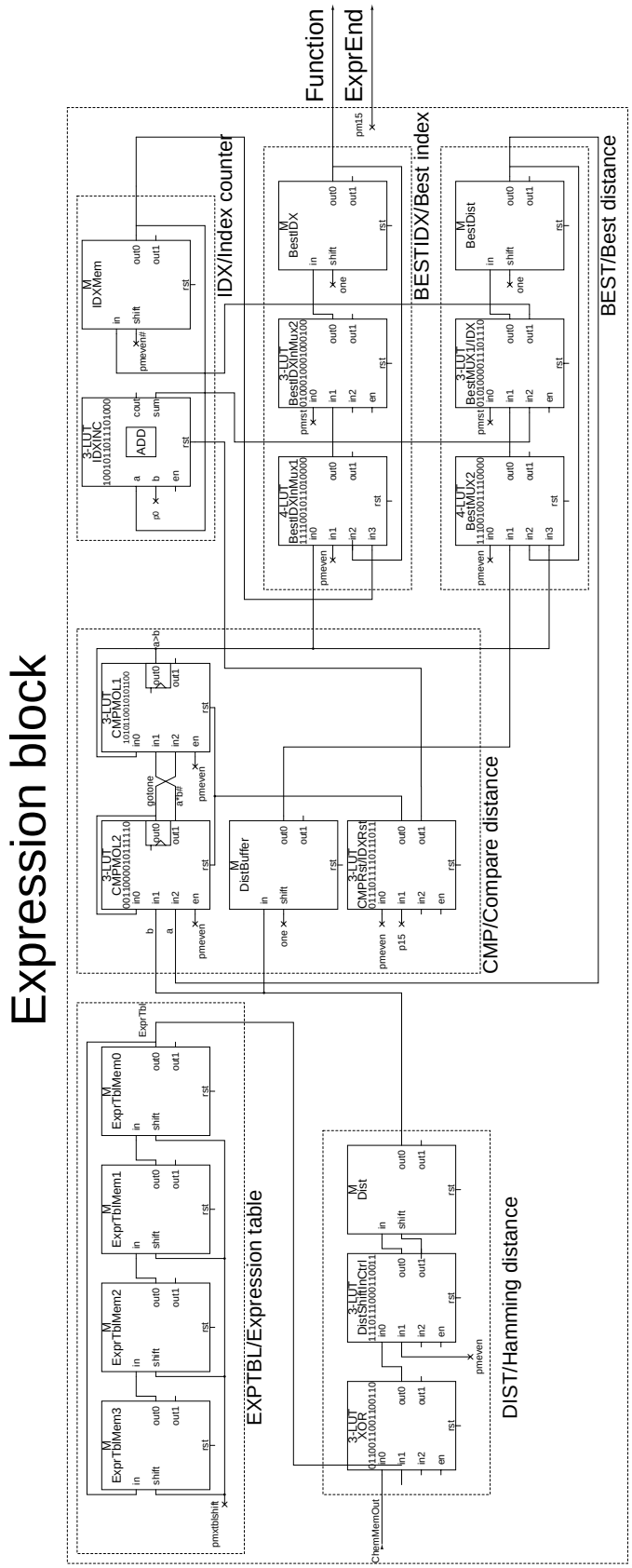

Figure D.9: Schematic of the molecular implementation of the expression block.

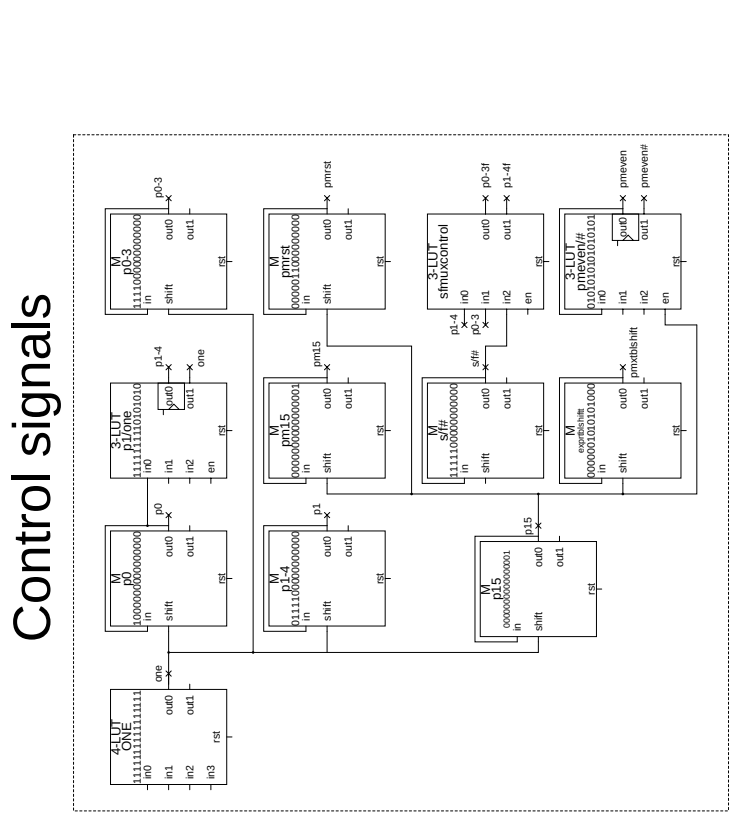

Figure D.10: Schematic of the molecular implementation of the control signals.

E Implementation of the obstacle avoidance robot controller

This appendix complements chapter 7. Section E.1 describes the FPGA module which is used to implement the multi-cellular spiking neural network to control the navigation of the mobile Khepera robot. Section E.2 details the hardware implementation of the multi-cellular spiking network. Further information about the FPGA module and the implementation may be found in [136].

E.1 FPGA module

The FPGA module is based on the APEX20K200E FPGA from Altera. Its architecture is illustrated in figure E.1 and table E.1 lists the features of the module. The FPGA module contains the FPGA together with memories and several connectors to interface with a desktop computer and user extensions. The architecture of the module is similar to that of the Excalibur development board [2], modified to perform as a fully Khepera-compatible extension. Therefore the module is also compatible with the Excalibur development tools. These tools allow to design systems on a chip that combine hardware and software. A 16- and 32-bit CPU (called Nios) and several peripherals (e.g. communication peripherals, timers) are provided for implementation in the FPGA by the manufacturer of the device. The main parts of the architecture are described below.

Memories. The non-volatile Flash memory contains the configuration of the FPGA, and user data or CPU programs. Two SRAM chips can be used to store data. These memories are fully accessible from the FPGA, while the configuration controller has partial access only to the address bus and control signals of these memories.

Configuration controller. The FPGA is SRAM-based and its configuration is volatile, i.e. its configuration needs to be downloaded after each power-up or reset. The configuration controller (EPM7064) takes care of the download procedure by transferring the configuration file from the Flash to the FPGA. The configuration controller is a non-volatile device and is programmed only once. In normal use this device is never reconfigured.

User interfaces. The module provides buttons and LEDs that can be used in applications. Configuration switches allow to select configuration options of the module: for instance self-reconfiguration can be enabled, an alternate FPGA configuration can be selected, and in a multi-module system the clock source can be selected. In addition, several

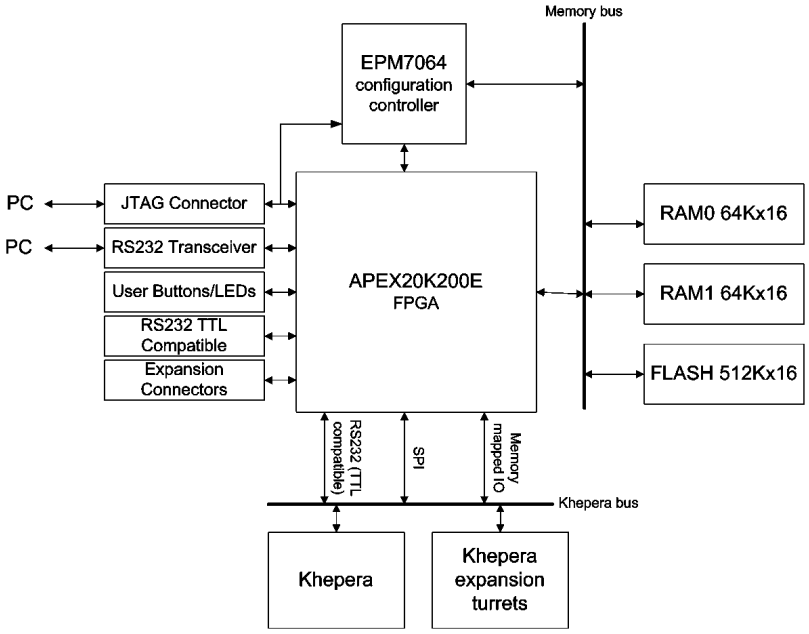

Figure E.1: Architecture of the FPGA module: in addition to the FPGA there are RAM and Flash memories to store programs and data, and connectors to interface with external devices.

• APEX20K200E-2X FPGA with 200'000 gates, 106'496 RAM bits (8320 logic elements)
• 1 MByte Flash memory (512Kx16)
• 256 KByte SRAM (two 64Kx16 chips)
• 2 user and 2 system push-buttons, 3 user LEDs, configuration switches
• 26 3.3V user I/O, 2 5V-compatible user I/O (e.g. TTL serial line)
• 26 5V-compatible I/O for the Khepera bus or user I/O
• Stackable modules (multi-FPGA system sharing a single clock)
• RS232 connector with transceiver
• JTAG connector for Altera ByteBlasterMV and MasterBlaster programmers
• Compatible with Excalibur board software development Kit
• Supply voltage: 4.5V to 25V
• Power board generates 1.8V and 3.3V (1.4A each voltage)
• On-board logic for configuring the FPGA from Flash
• Self-reconfiguration may be triggered by the FPGA
• Oscillator (33MHz) and zero skew clock distribution
• Power-on reset circuitry
• Clock and power pins available for user extensions

Table E.1: Features of the FPGA module

extension connectors are available to connect external devices.

Programming interfaces. A JTAG (Joint Test Access Group, a standardized type of programming interface) is used to program the configuration controller or the FPGA with a particular circuit. A serial interface with a RS232 transceiver (that adapts the serial interface voltage to that used in desktop computers) allows a desktop computer to communicate with the FPGA and to download programs in the CPU memory.

Khepera bus. All the digital I/O pins of the Khepera robot expansion bus are connected to the FPGA. When the module is used as a standalone device all the pins normally connected to the Khepera become available for general purpose I/O.

The module is composed of two stacked printed circuit boards (or turrets), one that contains the FPGA and the other that contains the power supply, as illustrated in figure 7.7. In addition several modules can be stacked sharing the same clock. This allows more complex circuits to be implemented in several communicating FPGAs.

E.2 Details of the hardware robot controller

The architecture of the multi-cellular robot controller implemented in the FPGA is illustrated in figure 7.8. In this section we describe how the functional part of the cells (i.e. the spiking neuron) is implemented, and how the processor in the environment subsystem configures the multi-cellular spiking network (i.e. programs its functionality according to the genetic string of the circuit).

E.2.1 Spiking neuron

Cells of the multi-cellular circuit are illustrated in figure 7.9. The functional part (phenotype layer) of the cell implements a spiking neuron. The spiking neuron has inputs that are used to receive incoming spikes from neighboring neurons, and outputs that are used to send outgoing spikes. To allow the circuit functionality to change when the genetic string of the circuit is modified (i.e. to provide run-time reconfigurability) the functional part of the cells have a function input which indicates the type of neuron (excitatory or inhibitory) and connectivity pattern that should be implemented.

To translate this in hardware, the cells are designed to be totipotent: they implement all the possible functionalities (combinations of connectivity pattern and neuron type), and the appropriate functionality is selected at run-time according to the input. In particular the cells are connected at design-time to 25 neighboring cells so that they can express all the required connectivity patterns at run-time. A 26th input is used as an external input.

Figure E.2 shows the architecture of the neuron within the cell. Control inputs (`clock`, `enable`, `reset`) allow to run, pause and reset the neuron. The inputs from connected neurons are `in1`..`in`n, which convey the incoming spikes, and `ins1`..`ins`n which indicate the type of connected neurons (spike "sign"). The `function` input tells the neuron which connectivity pattern to use and what is the type of the neuron. The output `out` sends outgoing spikes, and `outs` sends the type (sign) of the current neuron. This type is defined by the `function` input.

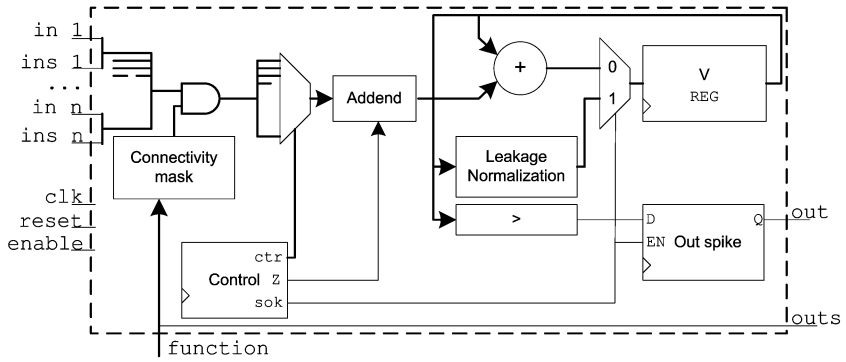

Figure E.2: Functional part of the cell that implements a spiking neuron. The main parts are a register holding the membrane potential value, a connectivity mask block, an addend block, a leakage and normalization block, a comparison block and a control unit.

The spiking neuron contains a 7-bit register (V) holding the membrane potential.[1] To reduce the size of the implementation, the inputs are processed sequentially (time-multiplexed implementation). The number of clock cycles for one network update is equal to $n + 1$, where n is the total number of inputs ($n = 26$ here).

The *control* unit generates the necessary sequence to multiplex the inputs in clock cycles 1 to n to update the register V with the contribution of the inputs (`sok=0` during the first n cycles). At clock cycle $n + 1$ (`sok=1`) the output is updated (emission of a spike if the register is above or equal to the firing threshold) and the register V is loaded with the content of the *leakage and normalization* block. This block decrements the value of the register if it is below the threshold (leakage) or resets the register to 0 if its value is above the threshold or below 0.

The *connectivity mask* unit generate the adequate run-time connectivity according to the `function` input by providing a "connectivity mask" with a look-up table. This connectivity mask is then ANDed with the inputs of all the neighboring cells to give the effective input spikes. Those are, together with the signs, multiplexed and sent to the *addend* block. This block provides the value to add to the register V. Signal `Z` of the *control* unit indicates whether the addend block handles input 1 (external input) which has a fixed weight of 10, or inputs 2 to n which have a weight of +2 or -2, depending on the sign of the connected neuron.

E.2.2 Configuration of the multi-cellular circuit

A 16-bit Nios processor is implemented in the FPGA to interface the multi-cellular network with the robot, run the evolutionary algorithm and the morphogenetic system, and eventually configure the multi-cellular circuit according to the genetic string.

[1] The register size is defined by the number of inputs and by their maximum weights. Since there are 25 inputs with a weight of +/-2 and one external input with a weight of +10, the maximal value of V is $3 + 25 \cdot 2 + 10 = 63$ (the number 3 is the maximum value of V before a spike is emitted), and the minimum value is $0 + 25 \cdot -2 = -50$. Therefore 7 bits are sufficient to hold the values of V.

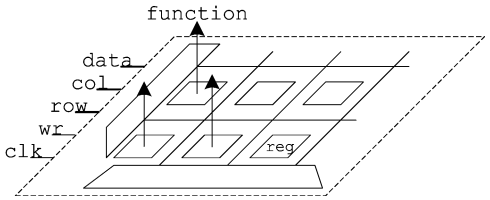

Figure E.3: The configuration layer holds the functionality of each cell. It is implemented as a writable array of registers. Each register is connected to the `function` input of the corresponding cell.

Entity	Resource use (LE)	Longest delay
• Complete system	8222	23.3 ns
• CPU (incl. UART and I/O)	2121	17.9ns
• Network interface	163	7.83ns
• Net. and config. layer (64 cells)	5939	21.7ns
• Single neuron	109	13.3ns

Table E.2: Number of logic elements and longest register to register delay after place and route for the complete system and its main parts (compiled individually). The network interface is composed of 8 spike generators and 2 activity measurement units which are required for the robotic application.

Configuring the multi-cellular circuit consists in setting all the `function` input of the spiking neurons. The processor configures the multi-cellular circuit by writing in a memory called the configuration layer. The configuration layer is illustrated in figure E.3. It is an array array of registers, with one register per cell, that holds the value of the `function` input of the spiking neurons. The `row` and `col` inputs allow to select a specific register, and the `wr` (write signal) stores the value on the `data` input in the selected register.

E.2.3 Implementation results

A multi-cellular network of 64 neurons is implemented in the FPGA module. Since the FPGA runs at 33 MHz and a network update requires 27 clock cycles, the neural network can be updated 1.2 million times per second, although the theoretical maximum frequency of the system is about 42 MHz.

Table E.2 summarizes the number of logic elements (LEs, elementary functional blocks in the FPGA) and the longest register to register delay for the complete system and its main parts. Data for the main parts are obtained after a standalone compilation of these parts. When the entire system is synthesized, optimizations may reduce the resources required by some of the parts (e.g. by removing common blocks in these parts). This is the case for the neuron: it uses 109 LEs alone but when part of a network of 64 neurons the resources are of about 90 LEs per neuron. The network and configuration layer take most of the space (5939 LEs of the 8320s LE available in the FPGA).

F Stimuli parameters and alternate learning measures

This appendix complements chapter 8 that describes how a multi-cellular network of spiking neurons can be used to learn and discriminate the direction of motion of a stimulus applied to the network. In section F.1 we show the influence of the stimulus parameters on the learning performance. In section F.2 we show some alternate metrics that may be used to measure the learning performance of the network

F.1 Stimuli parameters

In this section we discuss how to select the parameters of the stimulus (noise, speed, amplitude) and we show their influence on the learning performance in the task described in chapter 8. The learning performance is the function F_{act} defined in section 8.4.

During learning, the stimulus parameters are selected so that neurons stimulated with the bump of the stimulus are likely to fire. As the stimulus moves, the neurons which fire tend to follow the bump of the stimulus. As a consequence the synaptic activation of the connections oriented along the direction of the stimulus tend to increase (post-synaptic neurons tend to fire after pre-synaptic neurons), whereas the synaptic activation of connections oriented along the opposite direction of the stimulus tend to decrease (pre-synaptic neurons tend to fire after post-synaptic neurons)

During recall, the stimulus parameters are selected so that the neurons fire under the cumulated contribution of the noise and of the stimulus, but not with the contribution of the noise or the stimulus alone. When this condition is not respected the learning performance is affected (i.e. the network is not capable of discriminating the stimulus direction after learning). Either the neurons may not be stimulated enough (e.g. too low stimulus amplitude or level of noise) and therefore there is no activity in the network regardless of the stimulus direction. Alternatively the neurons may be overstimulated (e.g. too high stimulus amplitude or level of noise), and in this case the neurons always fire regardless of the stimulus direction.

The noise, speed and amplitude of the stimulus used during recall have coupled influence on the learning performance. This is illustrated in figure F.1 that shows the learning performance F_{act} for various stimulus parameters during recall. In particular the network activity induced by a moving stimulus is function not only of its amplitude and noise but

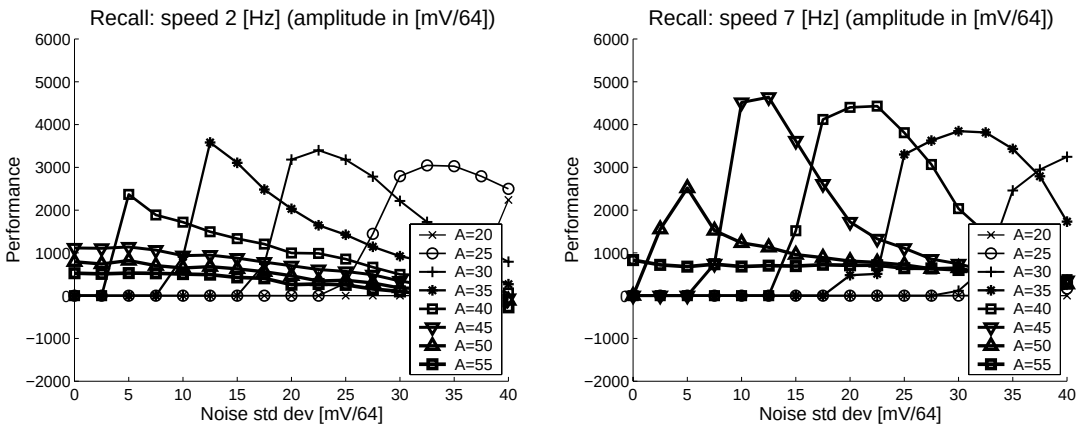

Figure F.1: Learning peformance F_{act} for a stimulus moving at 2 Hz (left plot) and 7 Hz (right plot) for different parameters of the stimulus used during recall. Learning lasts $TLearn = 10$ seconds with the stimulus parameters of section 8.4. Noise and stimulus speed and amplitude have a coupled influence on the learning performance. Consider the stimulus at 7 Hz with amplitude of 40 mV/64. Below a level of noise of 20 mV/64 the performance drops because the cumulated contribution of the stimulus and of the noise is insufficient to generate network activity. Above this noise level the performance drops because noise tends to decorrelate the network activity from the stimulus by generating activity which is independent of the stimulus direction. Since the contribution of the stimulus to the membrane potential is function of the stimulus speed, slower or higher stimulus speeds require a different combination of noise and amplitude to achieve optimal learning performance.

also of its speed: fast stimuli contribute during a shorter time to the membrane potential of the neurons, and therefore they increase the membrane potential less than slower stimuli.

In a real-world application, for instance a robot with a camera navigating in an arena with different colored stripes on the walls, the distinction between stimuli parameters used during learning and recall may not be appropriate. For instance if a robot learns during its lifetime (online learning), "learning" and "recall" occur at the same time with the same stimuli. Learning is however possible even if the stimulus parameters are identical during learning and recall. This is illustrated in figure F.2 which shows a color map of F_{act} in function of the stimulus amplitude and noise level for a stimulus moving at 6 Hz. The left plot shows the values of F_{act} above 0 and the right plot shows the values of F_{act} below 0. Lighter areas on the map indicate combinations of amplitude and noise where learning induces a change of network activity in function of the stimulus direction (i.e. they correspond to parameters where learning is possible).

Learning tends to be irreversible when identical stimuli parameters are used for learning and recall: once a stimulus is learned, the synaptic connections tend not to change even if a stimulus moving in the opposite direction is applied to the network, unless the stimulus amplitude or the noise is increased. This is caused by the relatively low activity of the network when the stimulus moves backward after learning: there are not enough connected neurons firing within the temporal learning window to reverse the synaptic activations.

Stimuli may not only move to the left or right at a fixed speed during recall: the speed

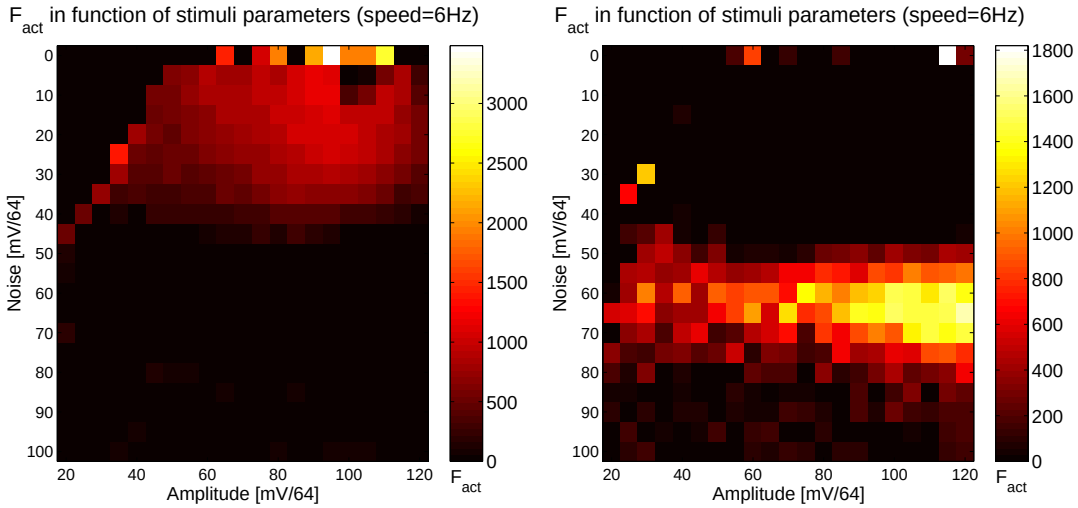

Figure F.2: Color map of F_{act} when learning and recall are done with the same stimuli parameters. The stimulus speed is 6 Hz. To clarify the view the results are represented on two maps: the left map indicates the values of F_{act} above 0 (values below 0 are clipped to 0); the right map indicates the absolute of F_{act} when it is below 0 (values above 0 are clipped to 0). Lighter areas correspond to parameters where learning is possible.

may vary, especially in a real-world application (e.g. the robotic application described in chapter 9). The influence of the stimulus speed during recall on the learning performance can be represented by tuning curves. Tuning curves describe the response of a network (here the activity of the network) in function of a parameter that characterizes the stimulus (here the stimulus speed) [22]. Figure F.3 illustrates these tuning curves for various learning speeds. The stimulus amplitude and noise, and the learning and recall duration are indicated in section 8.4.

With slow stimuli speeds during learning (e.g. 2 Hz) the tuning curves tend to be symmetrical, that is the network reacts in the same way to a forward and backward moving stimulus: the network is unable to learn slowly moving stimuli if they induce activity in connected neurons after a time longer than the learning window defined by the network time constants. As expected, with higher learning speeds the tuning curves indicate a higher network activity when the stimulus moves forward than when it moves backward.

For speeds during recall below approximately -5 Hz or above +8Hz the network acts as a bandpass filter and the network activity drops. Indeed above a certain speed defined by the network time constants, the peak of the stimulus may not stay long enough on any neuron for the membrane potential of these neurons to reach the firing threshold. The peak of the tuning curves (i.e. the recall speed at which the network activity is highest) is given by the time constants of the neural model; i.e. these time constants determine the range of speeds in which learning is possible.

In summary it is necessary to select an appropriate combination of stimuli parameters (noise, speed and amplitude) and/or time constants of the neurons so that learning and successful discrimination of the direction of motion of the stimulus is possible.

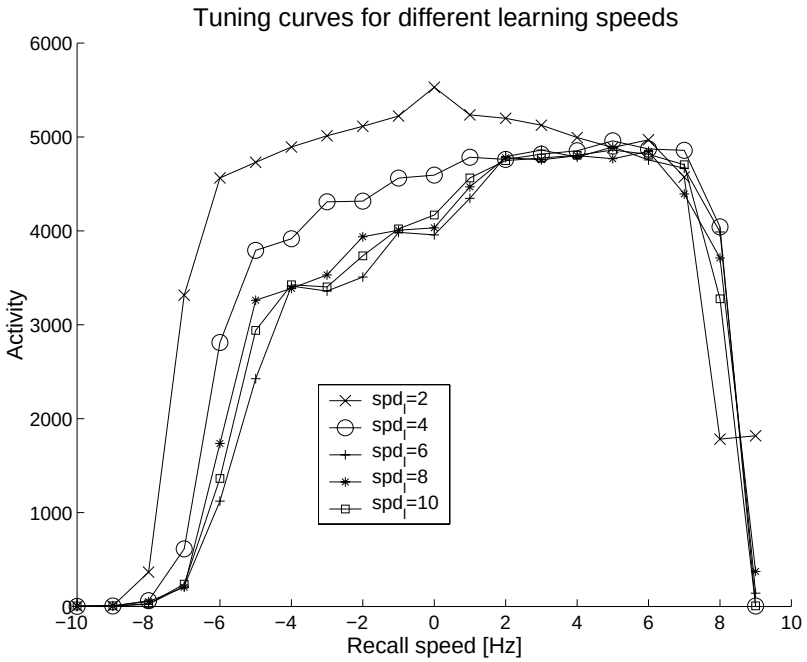

Figure F.3: Tuning curves showing the network activity in function of the stimulus speed during recall for different speeds used during learning. Each curve corresponds to a different stimulus speed during learning, indicated in the legend of the plot. Negative and positive speeds on the horizontal axis correspond to stimuli moving in the backward respectively forward direction during recall.

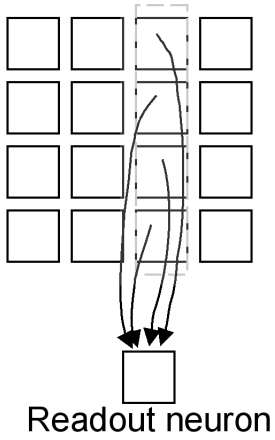

Figure F.4: A readout neuron connected to the neurons in the measurement column can be used to detect the synchronous activity of those neurons.

F.2 Alternate measures of learning

In this section we describe two alternate measures of the learning performance for the network described in chapter 8. One is based on the firing synchrony and the other is based on the activity correlation. The parameters of the stimulus used during learning and recall are indicated in section 8.4.

F.2.1 Firing synchrony

Figure 8.7 indicates that there are more neurons firing in synchrony when the stimulus moves forward than when it moves backward. A measure of learning may be devised from this observation. Synchronous activity may be detected by a readout neuron which is connected to the neurons of the measurement column, as illustrated in figure F.4. By selecting appropriate time constants this neuron can be made to fire when a predefined number of incoming spikes are synchronous. In this case its activity would be higher when the stimulus moves in the forward direction (synchronous spikes) than when it moves in the backward direction.

Alternatively the average number of neurons that fire synchronously $Sync$ in the measurement column during a time T can be determined directly:

$$Sync = \frac{\sum_{t=0}^{T} \rho(t)}{N},$$

where $\rho(t)$ is the number of neurons firing in a column of neurons situated in the middle of the network at time t, and N is the number of network steps where the value $\rho(t)$ is non-null for t in the interval $[0; T]$. In other words N is the number of steps (milliseconds) when there is non-zero activity in the measurement column.

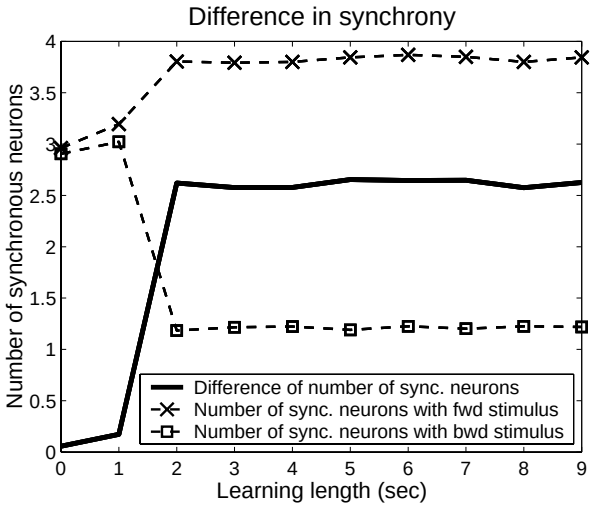

Figure F.5: Difference in average number of synchronous neurons F_{sync} in function $TLearn$. The dashed lines represent $Sync_{fw}$ and $Sync_{bw}$.

The learning performance is defined as the difference between the number of neurons firing synchronously when the stimulus moves forward ($Sync_{fw}$) and backward ($Sync_{bw}$):

$$F_{sync} = Sync_{fw} - Sync_{bw}.$$

Figure F.5 shows $Sync_{fw}$, $Sync_{bw}$ and F_{sync} in function of the learning time (the recall time $TRec$ is 50 seconds, recall is repeated 10 times and the results averaged). As expected for a measure of the learning performance, F_{sync} is close to 0 when the learning time is small and increases with longer learning periods: as the learning time increases, $Sync_{bw}$ tends to lower while $Sync_{fw}$ remains relatively constant.

This measure is however less precise than the one devised in chapter 8 since it fails to show a difference in learning performance after more than 2 seconds of learning (the measure shown in chapter 8 continuously shows an increase of learning performance up to about 15 seconds of learning).

F.2.2 Autocorrelation length

Another way to measure learning is to exploit the correlation between the activity in the measurement column and the stimulus. The autocorrelation indicates the degree of similarity of a signal compared to itself but shifted by a delay τ [22]. The autocorrelation of $\rho(t)$ is:

$$Corr(\tau) = \int_{-\infty}^{+\infty} \rho(t)\rho(t+\tau)dt.$$

The autocorrelation with the forward and backward moving stimulus is noted $Corr_{fw}$ and $Corr_{bw}$. The autocorrelation is illustrated in figure F.6 for the forward and backward moving stimuli. As expected the autocorrelation drops faster with increasing values of τ

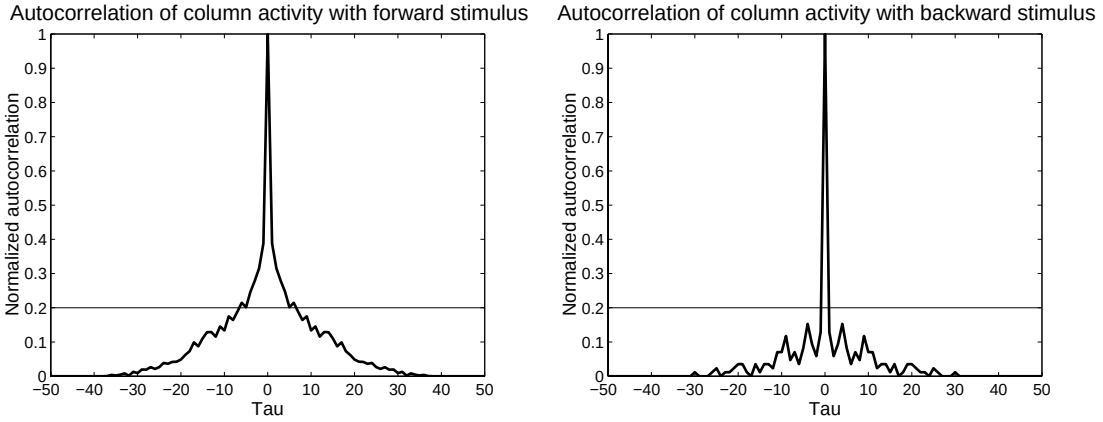

Figure F.6: Autocorrelation of the activity of one column with a forward moving stimulus (left plot) and backward moving stimulus (right plot). Horizontal axis is τ. The autocorrelation is computed on 50 seconds of network activtiy after 10 seconds of learning. The horizontal line indicates where the correlation length is measured.

when the stimulus moves in the backward direction than when it moves in the forward direction.

The learning performance can be defined in function of the autocorrelation length of the activity $\rho(t)$ in the measurement column. The autocorrelation length is the value of τ for which $Corr(\tau)$ drops to some lower value than $Corr(0)$. Here we consider the correlation length as the value of τ for which $Corr(\tau) = Corr(0)/5$. The learning performance F_{corr} is defined from the autocorrelation length τ_{fw} and τ_{bw} as follows:

$$F_{corr} = \tau_{fw} - \tau_{bw}.$$

Figure F.7 shows τ_{fw}, τ_{bw} and F_{corr} in function of $TLearn$. F_{corr} is low for short learning time and increases with longer learning time. It therefore satistifes the criteria for a measure of the learning performance.

The autocorrelation length is however impractical to detect the stimulus direction in real-time since computing it accurately requires to store the time of occurrence of spikes during a relatively long period (here we measured the autocorrelation after recording $\rho(t)$ for 50 seconds), and the correlation operation is relatively intensive computationally.

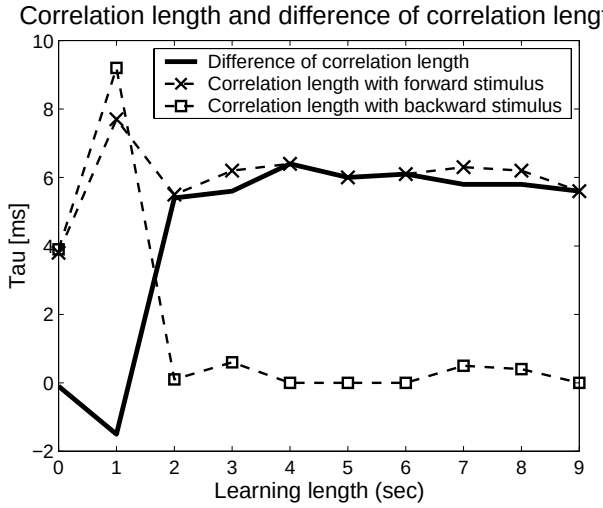

Figure F.7: Difference of the autocorrelation length of the activity of the network between the forward and backward stimulus F_{corr} in function of $TLearn$. The dashed lines represent τ_{fw} and τ_{bw}.

G DCAM Khepera camera module

None of the camera modules available for the Khepera robot filled the requirement of high-speed (50 Hz) image acquisition, pre-processing and transmission over a serial line. Therefore a custom digital camera interface module was developed for the Khepera robot for the application described in chapter 9. It uses the commercially available OV5017 monochrome 384x288 CMOS camera chip from OmniVision [127].

In addition to acquiring images, the camera module also centralizes communication with the Khepera robot. Therefore vision, sensory data and motor commands can all be exchanged over a single serial communication line which is compatible with the POEtic chip or a desktop computer.

G.1 Hardware

The high-level architecture of the camera is illustrated in figure G.1. The camera and its PCB are shown in figure G.2. The camera module contains a 16 MHz 8-bit RISC CPU (Atmel ATmega64L), the camera itself (OV5017), a 512KB SRAM, and a CPLD (Altera EPM7256) which is used to implement interfacing logic. Those elements are detailed below.

CPU The CPU supervises the operation of the camera module. It controls the acquisition of video frames and pre-processes them. In particular it does digital image stabilization which is required by the rocking motion of the robot. It also communicates with the Khepera robot via a serial line to send motor commands and read the robot sensors. Finally it sends the data of the various sensors to the POEtic chip (vision, infra-red proximity sensors, floor sensors and wheel speeds) and it receives the motor commands in return, via a serial line operating at 115'200 bps.

CPLD The CPLD serves as an interface for the CPU, the camera and the RAM. The CPLD operates in two modes: *normal* or *acquisition*. In normal mode the RAM is mapped in the memory space of the CPU. Therefore the CPU can access the external RAM with normal memory read and write operations. In acquisition mode the CPLD takes ownership of the RAM bus to store the video data (this process

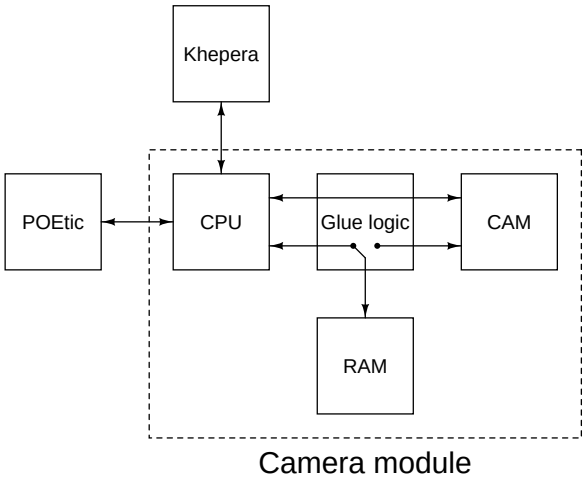

Figure G.1: Architecture of the camera module (inside the dashed block). The module is composed of a CPU, the camera and a RAM which serves as a frame buffer. Glue logic implemented in a CPLD is used to grab the pixels coming from the camera and store them in the RAM. The RAM is mapped in the address space of the CPU, therefore the CPU can access transparently the whole grabbed image. The CPU interfaces with the Khepera robot (e.g. to send motor commands or read sensors) and with the POEtic chip or a desktop computer (e.g. to send images and receive robot commands) over two serial lines.

is described afterwards). In both modes the CPU can access the camera registers and the CPLD registers which are also mapped in the memory space of the CPU. The CPLD registers relate to image acquisition and low-level image preprocessing. Registers specify the starting and ending line of the image to grab, and a downsampling factor can be specified. Finally the CPLD can do on-the-fly binarization of the grabbed image with a programmable threshold and pack 8 pixels in a single byte.

RAM The RAM is used to store the video frames (frame buffer) and can also be used to store data of CPU programs.

Camera The OV5017 is a digital camera chip. It provides a continuous stream of bytes (pixels) and control signals. Control signals include the pixel clock (indicating when a new byte is available for reading), and signals indicating the start of a frame and the start of a line. Those signals are used by the CPLD during acquisition. A number of registers which are accessible by the CPU control parameters of the camera such as the brightness, contrast, frame rate, gamma correction [127].

Image acquisition starts when the CPU writes a particular register of the CPLD. Once written, the CPLD goes in the acquisition mode and takes control of the RAM address and data bus. It then waits for the beginning of a new video frame. Once the start of the frame is detected it stores each incoming byte from the camera in sequential addresses in the RAM. Once the entire frame is acquired, it sets a bit in a register indicating the end of the acquisition and triggers an optional CPU interrupt. At the same time it goes in normal

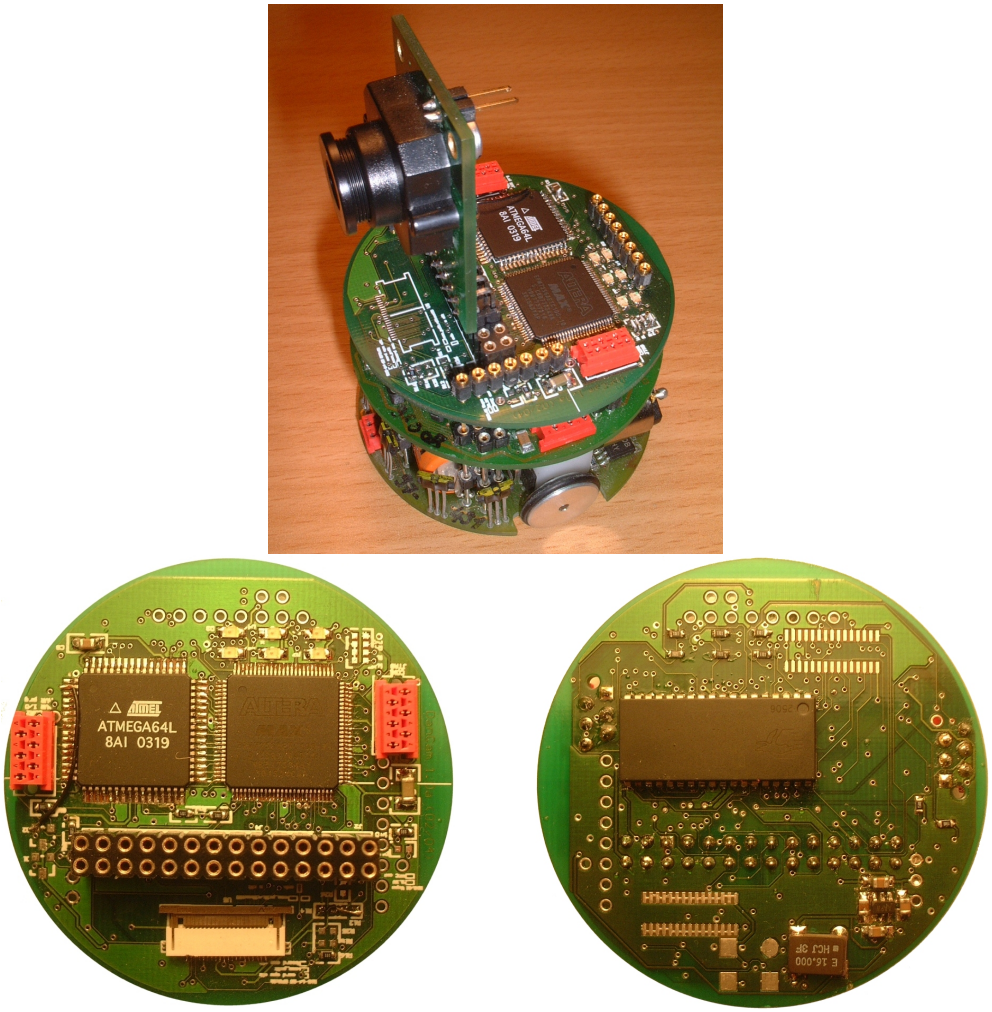

Figure G.2: Top: camera module mounted on the Khepera robot. Bottom-left: top-side of the camera module PCB with the CPU (left) the CPLD (right) and the camera connector (bottom). Bottom-right: bottom side of the PCB with the RAM (middle-left).

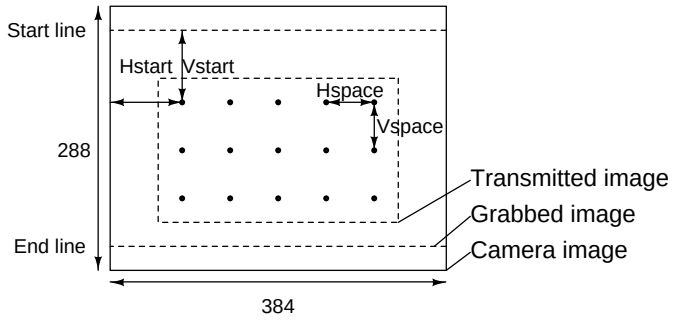

Figure G.3: The frame provided by the camera is a 384x288 image (camera image). The CPLD grabs a part of this image from the start line to the end line (grabbed image). The image which is eventually transmitted by the camera module is selected inside the grabbed image according to the start offset (Hstart and Vstart) and the spacing between columns and rows (Hspace and Vspace).

mode and releases control of the RAM bus which can now be accessed by the CPU to read the video frame.

The camera provides an image every 20 ms. As the image cannot be acquired and read from the frame buffer by the CPU at the same time, the maximum frame period is 40 ms when full scale images are grabbed. It is however possible to grab a subset of the image by specifying the starting and ending lines to grab with the corresponding CPLD registers. In this case it may be possible to grab and access to images with a 20 ms period, e.g. by acquiring only half-height images. Eventually the CPU processes and transmits a subset of the grabbed image. This allows software subsampling or to select regions of interest. The subset is selected by setting the appropriate offset to the beginning of the image to transmit (Hstart, Vstart) and by setting the spacing between rows and columns of pixels (Hspace, Vspace). These parameters are set by sending the corresponding commands to the module. Figure G.3 illustrates the image transmitted by the camera (camera image), the part which is acquired by the CPLD (grabbed image) and eventually the part that is transmitted (transmitted image).

G.2 Software

The task of the operating system (OS) of the camera is to transmit over the serial line the camera images and the sensory information of the Khepera. At the same time it listens for commands received over the serial line.

Commands received over the serial line can be motor orders for the Khepera robot, or changes of parameters of the module (e.g. size and position of the frame to transfer). Commands consist of packets of 5 bytes where the first byte indicates the operation (e.g. motor command) and the following bytes are command-dependent parameters (e.g. motor speeds).

The operating system continuously reads the sensors of the Khepera robot in background (proximity and floor sensors, wheel speeds), and mirrors their state in local vari-

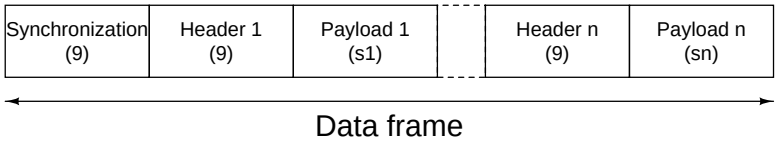

Figure G.4: Structure of the data frame sent by the camera module and size of the respective fields. The start of a data frame is indicated by a synchronization marker. Data is follows in different payloads which are indicated by a preceding header.

ables. Each time a new frame is acquired, the operating system streams a data frame containing the desired part of the image and the mirrored state of the Khepera sensors on the serial line.

The format of the data frame is described in figure G.4. It starts by a 9 byte long synchronization marker. It is used to synchronize the receiver on the start of the frame. Afterwards the data is transferred in different payloads. The start of the payload is identified by a 9 byte long header. The length of the payload is not encoded as it is known to end when a new header or a synchronization marker is encountered. This allows data of unpredictable length to be transferred directly on the serial line without the need to do an intermediate buffering, thereby reducing the computational requirements. Data of unpredictable length can occur in the case of image compression.

In the application described in chapter 9 the data frame consists of two payloads. The first is fixed length and contains the data relative to the Khepera sensors. The second contains the video data and it may be of variable length, depending on the image transmission mode or image size. In particular the horizontal and vertical image size, as well as the horizontal pixel interval and vertical line interval are programmable. The image transmission modes are described hereafter.

Raw mode Images are acquired and streamed without pre-processing: for each pixel a byte is transferred.

Raw filtered This mode differs from the raw mode in that subsampled images are filtered along the horizontal axis before being transferred to reduce image noise. Filtering is done by sending the average of n pixels, where n corresponds to the horizontal subsampling factor.

Temporal DPCM When there are relatively small changes from image to image, it is worth sending the difference between the current and the previous image with a more coarse quantization. Here this difference is encoded in 4 bits, therefore reducing the frame size by a factor two. This mode is however only suited for quasi-static images, or when objects move slowly, as no adaptive quantization is yet implemented.

JPEG-LS The speed of the serial line limits the maximum frame rate. The size of images may be reduced by compression, at the expense of higher CPU usage on the camera module.

Image	Raw		JPEG-LS-2		JPEG-LS-5		JPEG-LS-10		JPEG-LS-15	
	bytes	ΔT	bytes	ΔT	bytes	ΔT	bytes	ΔT	bytes	ΔT
384x288	110592	9.4	27000	7.8	18000	6.1	12000	4.3	9500	3.6
192x144	27648	2.4	8500	2.0	6000	1.8	4000	1.4	3500	1.2
96x72	6912	0.62	2700	0.56	1900	0.50	1400	0.42	1200	0.38
48x36	1728	0.20	950	0.16	650	0.16	450	0.14	400	0.14
24x18	432	0.06	300	0.06	200	0.06	150	0.06	120	0.06
10x8	80	0.02	-		-		-		-	

Table G.1: Size of the transmitted frames in bytes and frame period in seconds for different image size and compression ratios. The number after JPEG-LS indicates the maximum allowed pixel error. For the 10x8 image only half of the camera frame was grabbed, to demonstrate that the camera module can provide images at 50 frames per seconds. Image size and frame rate in JPEG-LS mode are based on several measures in a typical office environment.

Performing image compression on a 16 MHz 8-bit microcontroller requires a low complexity compression algorithm. The JPEG-LS low-complexity coder for still images with lossless and near-lossless compression [185] is used to compress the images with an adjustable error level before streaming the video frame over the serial line. The maximum pixel error is set by sending a corresponding command to the camera module.

In practice compression is beneficial for large images, where the bottleneck is the transmission speed. For smaller images the bottleneck is the processing time and in this case it is more advantageous to stream the images without compression.

Table G.1 illustrates the size of the video frame and the corresponding frame period for different image sizes and compression ratios. Figure G.5 illustrates how the images acquired by the camera look like.

Stabilized image This mode is used in the robotic application of chapter 9. The OS does vertical image stabilization which is based on the assumption that the bright areas in the image move slowly in the vertical axis. This assumption is verified in the environment in which the camera is used. Stabilization is done by aligning the images on their brightest line with a low pass temporal filter which limits the maximum change of alignment from frame to frame. The image is streamed without compression.

In the robotic application described in chapter 9 the stabilized image mode is used. In this application the camera module acquires images every 20 ms and the video payload is of fixed length. The size of the data frame is 77 bytes and it contains a 24x1 image and all the sensors of the Khepera robot.

Raw

JPEG-LS-2

JPEG-LS-5

JPEG-LS-10

JPEG-LS-15

Figure G.5: Images provided by the camera module in 384x288 with different compression ratios.

H Optimization of the retina size and connectivity

In this appendix we describe the optimization of the size and connectivity of the multi-cellular network or *retina* that is used in the robotic application of chapter 9. The mapping of the vision on the retina is done as described in section 9.3.

This retina is based on the multi-cellular network used in chapter 8. In that chapter we used a learning network composed of an array of 20 by 20 neurons, with each neuron locally connected to their 24 neighbors in a 5 by 5 area. Simulations of this network were however slower than real-time. When a real robot and real stimuli are used the network must operate in real-time and therefore the size and connectivity of the retina must be optimized to reduce the computational requirements. These optimizations also minimize the resources (i.e. the number of POEtic chips) required for a hardware implementation of the network.

In chapter 8 we showed that the activity of the retina was higher when facing the stimulus which had been learned than when facing the stimulus which moved in the opposite direction (opposite stimulus). The activity of the retina is thus an information that may be exploited by an evolved network to control the navigation of the mobile robot.

We therefore want to find the network size and neural connectivity which maximizes the sensitivity of the retina to the learned stimulus, that is the one with the highest learning performance. We use the measure of the learning performance introduced in section 8.4. The learning performance F_{act} is defined as the difference between the activity of the retina when the robot faces the learned stimulus $Fforward$ and when the robot faces the opposite stimulus $Fbackward$:

$$F_{act} = Fforward - Fbackward.$$

Since we not only want to maximize the difference of activity, but also minimize the number of neurons and the size of the connectivity neighborhood, we normalize F_{act} by the number of synaptic connections:

$$F_{act,norm} = \frac{F_{act}}{M \cdot N \cdot neighbors},$$

where M and N are the size of the retina and $neighbors$ is the size of the connectivity neighborhood.

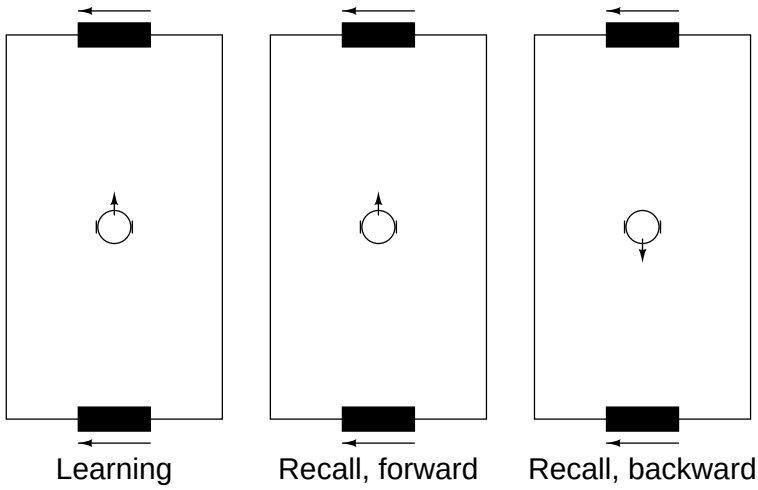

Figure H.1: Process used to measure the performance of the retina. First the robot faces the stimulus to learn for some $TLearn$ time with the learning mechanism activated. Afterwards the synaptic weights are frozen and the robot faces the learned (forward) stimulus for $TRec$ time. The total recorded network activity during this time is $Fforward$. Eventually the robot faces the opposite (backward) stimulus for the same time. The total recorded network activity during this time is $Fbackward$.

Figure H.1 illustrates the process which is used to measure the learning performance in the robotic context. The robot is always static, in the center of the arena. It first faces the stimulus to learn with the learning mechanism activated. The learning time $TLearn$ is 30 seconds. The amplitude and level of noise are given in section 9.3. Afterwards the synaptic weights are frozen. The amplitude and the noise level is set to those used during homing (section 9.3). The activity $Fforward$ is measured during $TRec = 3$ seconds with the robot facing the learned stimulus. Eventually $Fbackward$ is measured for the same duration with the robot facing the opposite stimulus. The stimulus is always a 6 Hz moving bar.

Retinas with sizes ranging from 1x1 up to 20x20 and that are wider than higher are considered. Neurons are connected to their local neighbors. Connectivity neighborhoods of 5x1, 5x3, 5x5, 7x1, 7x3 and 9x1 are considered (the first number is the width of the neighborhood). The neurons in the retina consist of regularly placed excitatory and inhibitory neurons, with the excitatory neurons representing 80 % of the total number of neurons. The distribution of these neurons is identical to that indicated in section 8.3.

Figure H.2 illustrates $F_{act,norm}$ for the different settings. The settings which maximize the normalized performance are an 8x8 network with a local connectivity of 5x3. Compared to the network used in chapter 8 with 9'600 synapses (20x20 network with 5x5 connectivity[1]) the number of synapses is reduced by a factor 10 (the 8x8 network with 5x3 connectivity has 896 synapses) and the learning performance is substantially identical. With this new retina real-time simulation of the network is possible on a 2.2 GHz AMD Athlon XP computer.

[1]Neurons are not self-connected

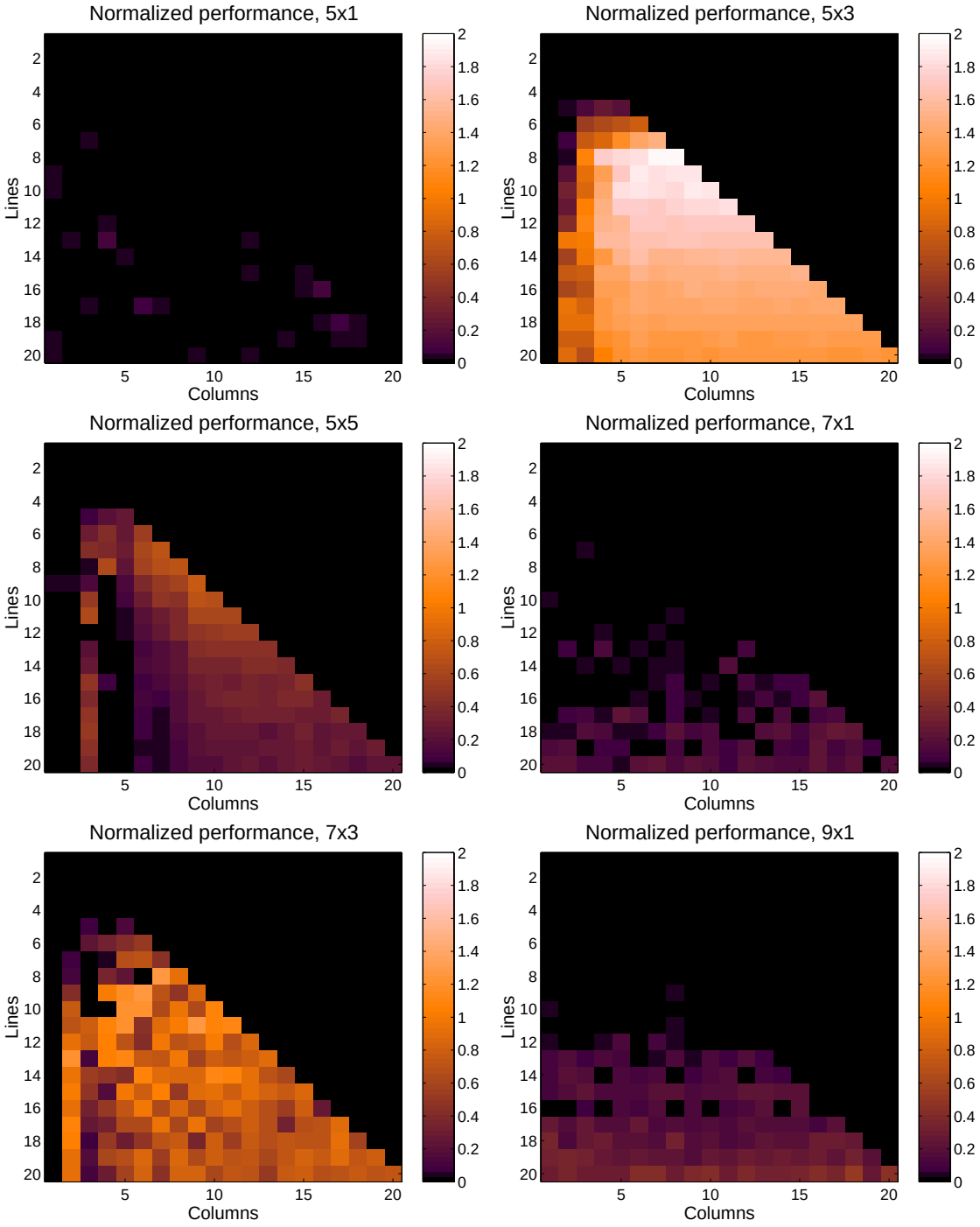

Figure H.2: Normalized performance $F_{act,norm}$ of the different retina settings (average of 50 measures). Brighter areas indicate higher performance settings. The network settings leading to the highest performance correspond to a network of 8x8 neurons with a 5x3 connectivity (top right plot).

Glossary

Application Specific Integrated Circuit (ASIC) ASICs are custom integrated circuit designed for a specific application. They are cheaper than FPGAs when manufactured in large quantities. They may also offer better protection against reverse engineering than FPGAs.

Cell In the POEtic terminology the cell is the basic functional element of POEtic circuits. Cells have external inputs and outputs, and take a functionality specified by the genetic string of the circuit. The architecture of a cell is composed of three layers: a phenotype layer, that is the functional part of the cell; a genotype layer, that contains the genetic code of the entire circuit; and a mapping layer, that decodes the genetic string to configure the phenotype layer. To allow dynamic reorganization, cells are totipotent: they can take any of the predefined functionalities that may be used in POEtic circuits. When POEtic circuits are implemented in the POEtic chip, the cells are implemented in its organic subsystem and they are built from logic elements called molecules.

Chromosome See genetic string

Dynamic routing Process by which routing units in the organic subsystem of the POEtic chip interconnect at run-time in a fully distributed way. This process relies on identifiers marking routing units as sources or targets of connections. The dynamic routing process interconnects sources and targets which have the same identifier.

Environment subsystem The environment subsystem is the part of the POEtic chip that interfaces the organic subsystem with the outside of the chip (i.e. its environment). It is composed of a custom 32-bit CPU and communication peripherals for interconnection with sensors or actuators located outside of the chip.

Evolution (of a system) Automatic creation of a system by using an evolutionary algorithm.

Evolutionary algorithm (EA) Family of search and optimization algorithms that are inspired by the evolution of biological organisms (fit organisms tend to have a higher

probability to transmit their genetic material). Evolutionary algorithms typically operate on a population of candidate solutions that are points in the search space (also called chromosomes, genetic strings or individuals). The candidate solutions are reproduced according to their performance or fitness at optimizing the problem at hand, and variations are introduced by randomly mutating and recombining their elements.

Fitness In an evolutionary algorithm, the performance of candidate solutions is measured according to their ability to optimize the problem at hand. The performance of a candidate solution is known as its fitness.

Fitness landscape The fitness of genetic strings can be represented by a surface in a high dimensionality space known as the fitness landscape. Genetic strings are coordinates in the landscape, and the fitness of these genetic strings represents the height of the surface at the corresponding coordinates.

Field-Programmable Analog Array (FPAA) Reconfigurable device that can be programmed to implement analog circuits.

Field-Programmable Gate Array (FPGA) Reconfigurable device that can be programmed to implement digital circuits. FPGAs are used for prototyping before producing application specific integrated circuits (ASICs), or in small to medium scale series when the costs of designing ASICs is prohibitive.

Gene In biology a gene is a segment of DNA on a chromosome. It contains a coding region and a control (or regulatory) region. The coding region encodes the sequence of amino-acids to build a protein. The control region controls the expression of the gene according to transcription factors that bind on this region.

In evolutionary computation, a gene is a part of the genetic string that encodes a parameter which is evolved.

Gene expression In biology a gene is said to be expressed if it is decoded to build the corresponding protein. Genes that are expressed thus produce proteins. The expression of genes is regulated by transcription factors, that can be produced by other genes. Therefore complex patterns of gene activation and repression may arise in biological cells.

Genetic operators Operators that are used in evolutionary algorithms to mimic the processes of natural evolution. For instance the selection operator selects individuals for reproduction according to their fitness, the crossover operator exchanges genetic material between two individuals, and the mutation operator introduces random variations in individuals.

Genetic string The genetic string represents the parameters that are optimized by an evolutionary algorithm. The genetic string is a string of digits or bits (if the digits are in base 2). It is also referred to as a chromosome, or as an individual in the population on which evolutionary algorithms operate.

Genotype The genotype contains all the genetic material or "instructions" necessary to build a biological organism or an artificial system.

Genotype-phenotype mapping Process by which the genotype is decoded into the corresponding biological organism or artificial system.

Hardware Description Language (HDL) Family of languages that describe textually the behavior of a circuit. Circuits can be synthesized from such a description using appropriate tools. VHDL and Verilog are two of the most common HDL in use.

Individual A member of the population of candidate solutions or genetic strings on which evolutionary algorithms operate.

Molecule Elementary logic element in the organic subsystem of the POEtic chip. It contains a flip-flop, a switch box for local routing, and a 16-bit register that can be used as a shift memory or as a lookup table to implement Boolean functions.

Organic subsystem The organic subsystem is a part of the POEtic chip composed of reconfigurable logic (molecules and routing units) where organisms, or multi-cellular circuits, are implemented.

Organism In the POEtic terminology an organism is a multi-cellular electronic circuit implemented in the POEtic chip.

Programmable Array Logic (PAL) Programmable electronic circuit suited to implement Boolean functions that can be expressed as sum-of-product terms.

Phenotype The phenotype is the observable part of a biological organism or of an artificial system that results from the decoding of the genotype.

POE Model of bio-inspiration that includes evolution (Phylogeny), development (Ontogeny) and learning (Epigenesis).

POEtic chip A custom reconfigurable device designed to implement bio-inspired mechanisms in hardware. In particular it may be used to implement POEtic circuits.

POEtic circuit Name given to multi-cellular circuits capable of evolution, development, and learning. POEtic circuits are composed of identical cells that contain a phenotype, mapping and genotype layer.

Reconfigurable device An integrated circuit that can be programmed after manufacturing to implement electronic circuits. Reconfigurable devices contain logic elements and routing resources. Logic elements are used to implement various Boolean functions, or memory elements. Routing resources are used to interconnect the logic elements. Reconfigurable devices are programmed by downloading a configuration string that describes the functionality of the logic elements and their interconnections.

Routing unit Element used in the organic subsystem of the POEtic chip to implement long distance connections (typically inter-cellular connections) or inter-chip connections. Routing units are capable of creating connections dynamically at run-time (dynamic routing).

Schematic A drawing or sketch that shows how the components of a system are connected together, for example in an electrical circuit.

Transcription factor Name given to proteins regulating the expression of genes (i.e. activating or repressing their expression).

Bibliography

[1] P. Agarwal, "The cell programming language," *Artificial Life*, vol. 2, no. 1, pp. 37–77, 1994.

[2] Altera, *Nios Embedded Processor Development Board Data Sheet Ver. 1.1*, Altera Corporation, San Jose, CA, March 2001.

[3] Anadigm, *Anadigm AN10E40 datasheet*, Anadigm Ltd., Anadigm Web Site (http://www.anadigm.com), 2000.

[4] P. J. Angeline, "Morphogenic evolutionary computations: Introduction, issues and examples," in *The Fourth Annual Conference on Evolutionary Programming*, J. R. McDonnel, R. G. Reynolds, and D. B. Fogel, Eds. Cambridge, MA: MIT Press, 1995, pp. 387–401.

[5] J. C. Astor and C. Adami, "A developmental model for the evolution of artificial neural networks," *Artificial Life*, vol. 6, no. 3, pp. 189–218, 2000.

[6] F. Aubret, R. Shine, and X. Bonnet, "Evolutionary biology: Adaptive developmental plasticity in snakes," *Nature*, vol. 431, pp. 261–262, 2004.

[7] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization," *Evolutionary Computation 1(1)*, pp. 1–23, 1993.

[8] A. V. Badyaev, G. E. Hill, M. L. Beck, A. A. Dervan, R. A. Duckworth, K. J. McGraw, P. M. Nolan, and L. A. Whittingham, "Sex-Biased Hatching Order and Adaptive Population Divergence in a Passerine Bird," *Science*, vol. 295, pp. 316–318, 2002.

[9] P. Bentley and S. Kumar, "Three ways to grow designs: A comparison of embryogenies for an evolutionary design problem," in *Proc. of Genetic and Evolutionary Computation Conference*, W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, Eds. San Francisco, CA: Morgan Kaufmann, 1999, pp. 35–43.

[10] E. J. W. Boers and H. Kuiper, "Biological metaphors and the design of modular artificial neural networks," Master's thesis, Department of Computer Science and Experimental and Theoretical Psychology, Leiden University, the Netherlands, 1992.

[11] J. C. Bongard, "Evolving modular genetic regulatory networks," in *Congress on Evolutionary Computation (CEC2002)*. IEEE Press, 2002, pp. 1872–1877.

[12] J. C. Bongard and R. Pfeifer, "Evolving complete agents using artificial ontogeny," in *Morpho-functional Machines: The New Species (Designing Embodied Intelligence)*, F. Hara and R. Pfeifer, Eds. Heidelberg: Springer-Verlag, 2003, pp. 237–258.

[13] S. Boshy and E. Ruppin, "Small is beautiful: Near minimal evolutionary neurocontrolllers obtained with self-organizing compressed encoding," in *Proceedings of the Seventh International Conference on Simulation of Adaptive Behaviour*, B. Hallam, D. Floreano, J. Hallam, G. Hayes, and J.-A. Meyer, Eds. Cambridge, MA: MIT Press-Bradford Books, 2002, pp. 345–346.

[14] D. Bradley, C. Ortega-Sanchez, and A. Tyrrell, "Embryonics + immunotronics: A bio-inspired approach to fault tolerance," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 215–223.

[15] J. P. Brockes, "Amphibian Limb Regeneration: Rebuilding a Complex Structure," *Science*, vol. 276, no. 5309, pp. 81–87, 1997.

[16] S. Brown and J. Rose, "FPGA and CPLD architectures: a tutorial," *IEEE Design & Test of Computers*, vol. 13, no. 2, pp. 42–57, 1996.

[17] T. Chen, H. L. He, and G. M. Church, "Modeling gene expression with differential equations," in *Proc. of Pacific Symposium on Biocomputing*, 1999, pp. 29–40.

[18] C. A. Coello, A. H. Aguirre, and B. P. Buckles, "Evolutionary multiobjective design of combinational logic circuits," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 161–170.

[19] E. Coen, *The art of genes*. Oxford: Oxford University Press, 1999.

[20] E. Damiani, V. Liberali, and A. Tettamanzi, "FPGA-Based Hash Circuit Synthesis with Evolutionary Algorithms," in *IEICE Trans. Fundamentals, Vol. E82-A, No. 9*, September 1999.

[21] R. Dawkins, *The selfish gene*. Oxford: Oxford University Press, 1976.

[22] P. Dayan and L. F. Abbott, *Theoretical Neuroscience*. Cambridge, MA: MIT Press, 2001.

[23] H. de Garis, "Genetic programming: Gennets, artificial nervous systems, artificial embryos," Ph.D. dissertation, Brussels University, 1992.

[24] ——, "Growing an artificial brain with a million neural net modules inside a trillion cell cellular automaton machine," in *Proceedings of the Fourth International Symposium on Micro Machine and Computer Science*, 1993, pp. 211–214.

[25] H. de Garis, L. de Penning, A. Buller, and D. Decesare, "Early experiments on the CAM-brain machine (CBM)," in *1st NASA/DoD Workshop on Evolvable Hardware*, A. Stoica *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 211–219.

[26] K. A. De Jong, "Genetic algorithms are not function optimizers," in *Proceedings of the Second Workshop on Foundations of Genetic Algorithms*, L. D. Whitley, Ed. San Mateo, CA: Morgan Kaufmann, 1992, pp. 5–17.

[27] F. Dellaert and R. Beer, "Toward an evolvable model of development for autonomous agent synthesis," in *Proc. of Artificial Life IV*, R. Brooks and P. Maes, Eds. Cambridge, MA: MIT Press, 1994, pp. 246–257.

[28] ——, "A developmental model for the evolution of complete autonomous agents," in *Proc. of the 4th Int. Conf. on Simulation of Adaptive Behavior*, P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. Wilson, Eds. Cambridge, MA: MIT Press-Bradford Books, 1996, pp. 393–401.

[29] E. A. Di Paolo, "Spike timing dependent plasticity for evolved robots," *Adaptive Behavior*, vol. 10, no. 3–4, pp. 243–263, 2002.

[30] J. Dubnau and T. Tully, "Gene discovery in *drosophila*: New insights for learning and memory," *Annu. Rev. Neurosci.*, vol. 21, pp. 407–444, 1998.

[31] P. Eggenberger, "Cell interactions as a control tool of developmental processes for evolutionary robotics," in *Proc. of the 4th Int. Conf. on Simulation of Adaptive Behavior*, P. Maes, M. J. Mataric, J.-A. Meyer, J. Pollack, and S. Wilson, Eds. Cambridge, MA: MIT Press-Bradford Books, 1996, pp. 440–448.

[32] ——, "Creation of neural networks based on developmental and evolutionary principles," in *Proc. of the International Conference on Artificial Neural Networks ICANN'97*, W. Gerstner, A. Germond, M. Hasler, and J.-D. Nicoud, Eds. Heidelberg: Springer-Verlag, 1997, pp. 337–342.

[33] ——, "Evolving morphologies of simulated 3d organisms based on differential gene expression," in *Proc. of the 4th European Conference on Artificial Life (ECAL97)*, P. Husbands and I. Harvey, Eds. Cambridge, MA: MIT Press, 1997, pp. 205–213.

[34] J. Eriksson, O. Torres, A. Mitchell, G. Tucker, K. Lindsay, D. Halliday, J. Rosenberg, J.-M. Moreno, and A. E. P. Villa, "Spiking Neural Networks for Reconfigurable POEtic Tissue," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 165–173.

[35] D. Federici, "Evolving a neurocontroller through a process of embryogeny," in *Proc. of the 8th Int. Conf. on Simulation of Adaptive Behavior*, S. Schaal, A. Ijspeert, A. Billard, S. Vijayakumar, J. Hallam, and J.-A. Meyer, Eds. Cambridge, MA: MIT Press-Bradford Book, 2004, pp. 373–382.

[36] ——, "Personal communication," August 2004.

[37] ——, "Using embryonic stages to increase the evolvability of development," *Proceedings of WORLDS workshop at GECCO 2004*, 2004.

[38] S. J. Flockton and K. Sheehan, "Intrinsic circuit evolution using programmable analogue arrays," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 144–153.

[39] ——, "Behaviour of a building block for intrinsic evolution of analogue signal shaping and filtering circuits," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 117–123.

[40] D. Floreano and C. Mattiussi, "Evolution of spiking neural controllers for autonomous vision-based robots," in *Evolutionary Robotics IV*, T. Gomi, Ed. Heidelberg: Springer-Verlag, 2001, pp. 38–61.

[41] D. Floreano and F. Mondada, "Automatic creation of an autonomous agent: Genetic evolution of a neural-network driven robot," in *Proc. of the 3rd Int. Conf. on Simulation of Adaptive Behavior*, D. Cliff, P. Husbands, J. Meyer, and S. W. Wilson, Eds. Cambridge, MA: MIT Press-Bradford Books, 1994, pp. 421–430.

[42] D. Floreano, D. Roggen, Y. Thoma, B. Schraner, J. M. Moreno, O. Torres, D. Halliday, A. Mitchell, and K. Wadwell, "POEtic Deliverable 14,17,20: POEtic Configuration for PO, PE and OE circuits," 2003.

[43] D. Floreano, N. Schoeni, G. Caprari, and J. Blynel, "Evolutionary bits'n'spikes," in *Proc. of Artificial Life VIII*, R. K. Standish, B. M. A., and A. H. A., Eds. Cambridge, MA: MIT Press, 2002, pp. 335–344.

[44] D. Floreano and J. Urzelai, "Evolution of plastic control networks," *Autonomous Robots*, vol. 11, pp. 311–317, 2001.

[45] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence Through Simulated Evolution*. John Wiley, Chichester, UK, 1966.

[46] S. Fusi, M. Annunziato, D. Badoni, A. Salamon, and D. J. Amit, "Spike-Driven Synpatic Plasticity: Theory, Simulation, VLSI Implementation," *Neural Computation*, vol. 12, pp. 2227–2258, 2000.

[47] J. C. Gallagher, "A neuromorphic paradigm for extrinsically evolved hybrid analog/digital device controllers: Initial explorations," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 48–55.

[48] C. Gershenson, "Classification of random boolean networks," in *Artificial Life VIII: Proceedings of the Eight International Conference on Artificial Life*, R. K. Standish, M. A. Bedau, and H. A. Abbass, Eds. MIT Press, 2002, pp. 1–8.

[49] W. Gerstner and W. Kistler, *Spiking Neuron Models*. Cambridge University Press, 2002.

[50] D. E. Goldberg, *Genetic Algorithms in Search Optimization & Machine Learning*. Reading, MA: Addison-Wesley, 1989.

[51] R. C. Gonzalez and R. E. Woods, *Digital Image Processing*. Addison-Wesley, 1993.

[52] T. Gordon, "Exploring models of development for evolutionary circuit design," in *Congress on Evolutionary Computation (CEC2003)*. IEEE Press, 2003, pp. 2050–2057.

[53] T. Gordon and P. Bentley, "Towards development in evolvable hardware," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2002, pp. 241–250.

[54] F. Gruau, "Neural network synthesis using cellular encoding and the genetic algorithm." Ph.D. dissertation, Laboratoire de l'Informatique du Parallélisme, Ecole Normale Supérieure de Lyon, France, 1994.

[55] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Proceedings of the First Genetic Programming Conference*. MIT Press, 1996, pp. 81–89.

[56] P. C. Haddow and G. Tufte, "An evolvable hardware fpga for adaptive hardware," in *Proceedings of CEC'00*, 2000, pp. 553–560.

[57] ——, "Bridging the Genotype-Phenotype Mapping for Digital FPGAs," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 109–123.

[58] P. C. Haddow, G. Tufte, and P. van Remortel, "Shrinking the Genotype: L-systems for EHW?" in *Proc. of the 4th Int. Conf. on Evolvable Systems (ICES 2001)*, Y. Liu *et al.*, Eds. Heidelberg: Springer-Verlag, 2001, pp. 128–139.

[59] P. C. Haddow and P. van Remortel, "From Here to There: Future Robust EHW Technologies for Large Digital Designs," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 232–239.

[60] A. Hamilton, K. Papathanasiou, M. R. Tamplin, and T. Brandtner, "Palmo: Field programmable analog and mixed-signal vlsi for evolvable hardware," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 335–344.

[61] G. Hartmann, G. Frank, G. Schäfer, and M. Wolff, "SPIKE128K-An Accelerator for Dynamic Simulation of Large Pulse-Coded Networks," in *Proceedings of MicroNeuro 97*, Dresden, 1997, pp. 130–139.

[62] P. Hasler, B. A. Minch, and C. Diorio, "Floating-gate devices: They are not just for digital memories anymore," *Proc. of the 1999 IEEE Int. Symp. on Circuits and Systems*, vol. 2, p. 388Ű391, 1999.

[63] H. Hemmi, J. Mizoguchi, and K. Shimohara, "Evolving large scale digital circuits," in *Proc. of Artificial Life V*, C. Langton and K. Shimohara, Eds. Cambridge, MA: MIT Press, 1996, pp. 213–218.

[64] T. Higuchi *et al.*, "Evolving hardware with genetic learning: A first step towards building a darwin machine," in *Proc. of the 2nd Intl. Conf. on Simulation of Adaptive Behaviour*, J.-A. Meyer, H. Roitblat, and S. Wilson, Eds. Cambridge, MA: MIT Press-Bradford Books, 1993, pp. 417–424.

[65] T. Higuchi, H. Iba, and B. Manderick, "Evolvable hardware," in *Massively Parallel Artificial Intelligence*, H. Kitano and J. A. Hendler, Eds. MIT Press, 1994, ch. 12, pp. 399–421.

[66] T. Higuchi, M. Iwata, I. Kajitani, M. Murakawa, S. Yoshizawa, and F. T., "Hardware evolution at gate and function levels," in *Proceedings of Biologically Inspired Autonomous Systems: Computation, Cognition and Action*, Durham, North Carolina, March 1996.

[67] T. Higuchi, M. Iwata, D. Keymeulen, *et al.*, "Real-world applications of analog and digital evolvable hardware," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 220–235, september 1999.

[68] T. Hikage, H. Hemmi, and K. Shimohara, "Comparison of evolutionary methods for smoother evolution," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 115–124.

[69] S. L. Hill and A. E. P. Villa, "Dynamic transitions in global network activity influenced by the balance of excitation and inhibition," *Network: Comput. Neural Syst*, vol. 8, pp. 165–184, 1997.

[70] J. H. Holland, *Adaptation in natural and artificial systems*. Ann Arbor: The University of Michigan Press, 1975.

[71] G. S. Hornby and J. B. Pollack, "Evolving l-systems to generate virtual creatures," *Computers and Graphics*, vol. 25, no. 6, pp. 1041–1048, 2001.

[72] ——, "The advantages of generative grammatical encodings for physical design," in *Proceedings of CEC 2001, 600–607*, 2001.

[73] L. Huelsbergen, R. E., and R. Slous, "Evolution of astable multivibrators in silico," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 66–77.

[74] G. Indiveri and R. Douglas, "ROBOTIC VISION: Neuromorphic Vision Sensors," *Science*, vol. 288, pp. 1189–1190, 2000.

[75] N. Jakobi, "Half-baked, ad-hoc and noisy: Minimal simulations for evolutionary robotics," in *Proceedings of the 4th European Conference on Artificial Life*, P. Husbands and I. Harvey, Eds. Cambridge, MA: MIT Press, 1997, pp. 348–357.

[76] ——, "Harnessing morphogenesis," School of Cognitive and Computing Sciences, University of Sussex, Tech. Rep. CSRP 423, 1995.

[77] T. Jones, "One operator, one landscape," Santa Fe Institute, Santa Fe NM 87501, Tech. Rep. 95-02-025, 1995.

[78] I. Kajitani *et al.*, "A Gate-Level EHW Chip: Implementing GA Operations and Reconfigurable Hardware on a Single LSI," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 1–12.

[79] T. Kalganova, "Bidirectional incremental evolution in extrinsic evolvable hardware," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 65–74.

[80] T. Kalganova and J. F. Miller, "Evolving more efficient digital circuits by allowing circuit layout evolution and multi-objective fitness," in *1st NASA/DoD Workshop on Evolvable Hardware*, A. Stoica *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1999, pp. 54–63.

[81] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets," *Journal of Theoretical Biology*, vol. 22, pp. 437–467, 1969.

[82] S. Kazadi, Y. Qi, I. Park, N. Huang, P. Hwu, B. Kwan, W. Lue, and H. Li, "Insufficiency of piecewise evolution," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 223–231.

[83] D. Keymeulen, R. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 305–316, September 2000.

[84] H. Kitano, "Designing neural networks using genetic algorithms with graph generation system," *Complex Systems*, vol. 4, no. 4, pp. 461–476, 1990.

[85] ——, "Challenges of evolvable systems: Analysis and future directions," in *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, T. Higuchi *et al.*, Eds. Berlin: Springer-Verlag, 1996, pp. 125–135.

[86] H. Kitano, S. Hamahashi, J. Kitazawa, K. Takao, and S. Imai, "The virtual biology laboratories: A new approach of computational biology," in *Proceedings of the Fourth European Conference on Artificial Life*, P. Husbands and I. Harvey, Eds. MIT Press, 1997, pp. 274–283.

[87] J. Kodjabachian and J.-A. Meyer, "Evolution and development of control architectures in animats," *Robotics and Autonomous Systems*, vol. 16, no. 2–4, pp. 161–182, 1995.

[88] A. Koopman and D. Roggen, "Evolving Genetic Regulatory Networks for Hardware Fault Tolerance," in *Proc. of Parallel Problem Solving from Nature VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, Eds. Heidelberg: Springer-Verlag, 2004, pp. 561–570.

[89] J. R. Koza, *Genetic Programming*. Cambridge, MA: MIT Press, 1992.

[90] J. R. Koza, F. H. Bennet III, D. Andre, M. A. Keane, and F. Dunlap, "Automated synthesis of analog electrical circuits by means of genetic programming," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 2, pp. 109–128, 1997.

[91] J. R. Koza, F. H. Bennett III, D. Andre, and M. A. Keane, "Reuse, parameterized reuse, and hierarchical reuse of substructures in evolving electrical circuits using genetic programming," in *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, T. Higuchi *et al.*, Eds. Berlin: Springer-Verlag, 1996, pp. 312–326.

[92] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, , and G. Lanza, *Genetic Programming IV: Routine Human-Competitive Machine Intelligence*. Kluwer Academic Publishers, 2003.

[93] S. Kumar and P. J. Bentley, "Biologically inspired evolutionary development," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 57–68.

[94] J. Langeheine, J. Becker, S. Fölling, K. Meier, and J. Schemmel, "A CMOS FPTA chip for intrinsic hardware evolution of analog electronic circuits," in *3rd*

NASA/DoD Workshop on Evolvable Hardware, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 172–175.

[95] Lattice, *GAL16V8 Datascheet*, Lattice Semiconductor Corporation, Lattice Web Site (http://www.latticesemi.com/), August 2004.

[96] P. Layzell, "A new research tool for intrinsic hardware evolution," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 47–56.

[97] ——, "Reducing Hardware Evolution's Dependency on FPGAs," in *Proc. 7th Int. Conf. on microelectronics, fuzzy and bio-inspired systems (MicroNeuro '99)*, 1999, pp. 171–178.

[98] P. Lehre and P. C. Haddow, "Developmental mappings and phenotypic complexity," in *Congress on Evolutionary Computation (CEC2003)*. IEEE Press, 2003, pp. 62–68.

[99] D. Levi, "Hereboy: A fast evolutionary algorithm," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 17–23.

[100] A. Lindenmayer, "Mathematical models for cellular interactions in development," *Journal of Theoretical Biology*, vol. 18, pp. 280–299, 1968.

[101] H. Liu, J. F. Miller, and A. M. Tyrrell, "An Intrinsic Robust Transient Fault-Tolerant Developmental Model for Digital Systems," in *Proceedings of WORLDS workshop at GECCO 2004*, K. Deb *et al.*, Eds. Heidelberg: Springer-Verlag, 2004.

[102] J. D. Lohn and S. P. Colombano, "Automated Analog Circuit Synthesis using a Linear Representation," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 125–133.

[103] ——, "A circuit representation technique for automated circuit design," *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 205–219, 1999.

[104] S. Luke and L. Spector, "Evolving graphs and networks with edge encoding: Preliminary report," in *Late Breaking Papers at the Genetic Programming 1996 Conference*, J. R. Koza, Ed., 1996, pp. 117–124.

[105] W. Maas and C. Bishop, *Pulsed Neural Networks*. Cambridge, MA: MIT Press / Bradford Books, 1998.

[106] W. Maas, "Networks of spiking neurons: the third generation of neural network models," *Transactions of the Society for Computer Simulation International*, vol. 14, no. 4, pp. 1659–1671, 1997.

[107] B. Manderick, M. de Weger, and P. Spiessens, "The genetic algorithm and the structure of the fitness landscape," in *Proceedings of the Fourth International Conference on Genetic Algorithms*, 1991, pp. 143–150.

[108] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand, "Embryonics: A new family of coarse-grained field-programmable gate array with self-repair and self-reproducing properties," in *Towards Evolvable Hardware*, E. Sanchez and M. Tomassini, Eds. Heidelberg: Springer-Verlag, 1996, pp. 197–220.

[109] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti, "Toward robust integrated circuits: The embryonics approach," *Proceedings of the IEEE*, vol. 88, no. 4, pp. 516–541, 2000.

[110] P. Marchal, C. Piguet, D. Mange, A. Stauffer, and S. Durand, "Embryological development on silicon," in *Proc. of Artificial Life IV*, R. Brooks and P. Maes, Eds. MIT Press, 1994, pp. 365–370.

[111] C. Mattiussi and D. Floreano, "Evolution of analog networks using local string alignment on highly reorganizable genomes," in *2004 NASA/DoD Conference on Evolvable Hardware*, R. Zebulum *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 30–37.

[112] C. Mead, *Analog VLSI and Neural Systems*. Reading, MA: Addison-Wesley, 1991.

[113] J. F. Miller, "On the filtering properties of evolved gate arrays," in *1st NASA/DoD Workshop on Evolvable Hardware*, A. Stoica *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1999, pp. 2–11.

[114] J. F. Miller, D. Job, and V. K. Vassilev, "Principles in the evolutionary design of digital circuits - part i," *Genetic Programming and Evolvable Machines*, vol. 1, no. 1-2, pp. 7–35, 2000.

[115] J. F. Miller and P. Thomson, "Cartesian genetic programming," in *Proceedings of EuroGP'2000*, R. Poli, W. Banzhaf, W. B. Langdon, J. F. Miller, P. Nordin, and T. C. Fogarty, Eds. Heidelberg: Springer-Verlag, 2000, pp. 121–132.

[116] ——, "A Developmental Method for Growing Graphs and Circuits," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 93–104.

[117] J. Miller, "Evolving developmental programs for adaptation, morphogenesis, and self-repair," in *Proc. of 7th European Conf. on Artificial Life*, W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, Eds. Heidelberg: Springer-Verlag, 2003, pp. 256–265.

[118] M. Mitchell, *An introduction to genetic algorithms*. Cambridge, Massachusetts: MIT Press, 1996.

[119] J. Mizoguchi, H. Hemmi, and K. Shimohara, "Production genetic algorithms for automated hardware design through an evolutionary process," in *Proceedings of the First IEEE Conference on Evolutionary Computation (ICEC'94)*. IEEE Press, 1994, pp. 661–664.

[120] F. Mondada, E. Franzi, and P. Ienne, "Mobile robot miniaturisation: A tool for investigation in control algorithms," in *Proc. of the 3rd Int. Symp. on Experimental Robotics*. Heidelberg: Springer-Verlag, 1994, pp. 501–513.

[121] G. E. Moore, "Cramming more components onto integrated circuits," *Electronics*, vol. 38, no. 8, April 1965.

[122] J. M. Moreno, Y. Thoma, E. Sanchez, and G. Tempesti, "Hardware realization of a bio-inspired poetic tissue," in *2004 NASA/DoD Conference on Evolvable Hardware*, R. Zebulum *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 237–244.

[123] Motorola, *Motorola Semiconductor Technical Data: Advance Information Field Programmable Analog Array 20 Cell Version, MPAA020*, Motorola Inc., 1997.

[124] J. D. Murray, *Mathematical Biology*. Heidelberg: Springer-Verlag, 1993.

[125] S. Nolfi and D. Parisi, "Growing neural networks," Institute of Psychology, Rome, Tech. Rep., 1992.

[126] ——, "Learning to adapt to changing environments in evolving neural networks," *Adaptive Behavior*, vol. 5, no. 1, pp. 75–98, 1997.

[127] Omnivision, *OV5017 datasheet, version 1.6*, OmniVision Technologies, Inc., Sunnyvale, CA, USA, 1997.

[128] A. Pérez-Uribe and E. Sanchez, "The FAST Architecture: A Neural Network with Flexible Adaptable-Size Topology," in *Proc. of the 5th Int. Conf. on Microelectronics for Neural Networks and Fuzzy Systems (MicroNeuro'96)*. IEEE Computer Society Press, 1996, pp. 337–340.

[129] T. Quick, C. L. Nehaniv, K. Dautenhahn, and G. Roberts, "Evolving embodied genetic regulatory network-driven control systems," in *Proc. of the 7th European Conference on Artificial Life (ECAL2003)*, W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kim, and J. Ziegler, Eds. Heidelberg: Springer-Verlag, 2003, pp. 266–277.

[130] I. Rechenberg, *Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution*. Stuttgart: Fromman-Holzboog Verlag, 1973.

[131] T. Reil, "Dynamics of Gene Expression in an Artificial Genome - Implications for Biological and Artificial Ontogeny," in *Proc. of the 5th European Conference on Artificial Life*, D. Floreano, F. Mondada, and J. Nicoud, Eds. Heidelberg: Springer-Verlag, 1999, pp. 457–466.

[132] ——, "Models of Gene Regulation - A Review," in *Artificial Life 7 Workshop Proceedings*, C. Maley and E. Boudreau, Eds. MIT Press, 2000, pp. 107–113.

[133] D. N. Reznick, F. H. Shaw, F. Helen Rodd, and R. G. Shaw, "Evaluation of the Rate of Evolution in Natural Populations of Guppies *Poecilia reticulata*," *Science*, vol. 275, pp. 1934–1937, 1997.

[134] D. Roggen and D. Federici, "Multi-cellular development: is there scalability and robustness to gain?" in *Proc. of Parallel Problem Solving from Nature VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, Eds. Heidelberg: Springer-Verlag, 2004, pp. 391–400.

[135] D. Roggen, D. Floreano, and C. Mattiussi, "A Morphogenetic Evolutionary System: Phylogenesis of the POEtic Tissue," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 153–164.

[136] D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware spiking neural network with run-time reconfigurable conectivity in an autonomous robot," in *2003 NASA/DoD Conference on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2003, pp. 189–198.

[137] D. Roggen, Y. Thoma, and E. Sanchez, "An Evolving and Developing Cellular Electronic Circuit," in *Proceedings of Artificial Life IX*, J. Pollack, M. Bedau, P. Husbands, T. Ikegami, and R. A. Watson, Eds., 2004, pp. 33–38.

[138] D. Salomon, *Data Compression: The Complete Reference*. Heidelberg: Springer-Verlag, 2004.

[139] E. Sanchez and D. Mange, "Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware," in *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, T. Higuchi *et al.*, Eds. Berlin: Springer-Verlag, 1996, pp. 35–54.

[140] C. C. Santini, R. Zebulum, M. A. Pacheco, R. Vellasco, and M. H. Szwarcman, "PAMA - Programmable Analog Multiplexer Array," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 36–43.

[141] M. Schmitt, "On computing boolean functions by a spiking neuron," *Annals of Mathematics and Artificial Intelligence*, vol. 24, no. 1-4, pp. 181–191, 1998.

[142] T. Schnier, X. Yao, and P. Liu, "Digital filter design using multiple pareto fronts," *Soft Computing*, vol. 8, no. 5, pp. 332–343, April 2004.

[143] T. Schoenauer, S. Atasoy, N. Mehrtash, and H. Klar, "NeuroPipe-Chip: A Digital Neuro-Processor for Spiking Neural Networks," *IEEE Transactions on Neural Networks*, vol. 13, no. 1, pp. 205–213, 2002.

[144] H. P. Schwefel, "Kybernetische evolution als strategie der experimentellen forschung in der strömungstechnik," Master's thesis, Technische Universität Berlin, Berlin, 1965.

[145] K. Sims, "Evolving 3D Morphology and Behavior by Competition," in *Proc. of Artificial Life IV*, R. Brooks and P. Maes, Eds. Cambridge, MA: MIT Press, 1994, pp. 28–39.

[146] M. Sipper, E. Sancher, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer, "A phylogenetic, ontogenetic, and epigenetic view of bio-inspired hardware systems," *IEEE Transactions on Evolutionary Computation*, vol. 1, no. 1, 1997.

[147] T. Smith, P. Husbands, and M. O'Shea, "Not measuring evolvability: Initial investigation of an evolutionary robotics search space," in *Congress on Evolutionary Computation (CEC2001)*. IEEE Press, 2001, pp. 9–16.

[148] K. Stanley and R. Miikkulainen, "A taxonomy for artificial embryogeny," *Artificial Life*, vol. 9, no. 2, pp. 93–130, 2003.

[149] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher, "A self-repairing and self-healing electronic watch: The biowatch," in *Proc. of the 4th Int. Conf. on Evolvable Systems (ICES 2001)*, Y. Liu *et al.*, Eds., Heidelberg, 2001, pp. 112–127.

[150] A. Stoica, "Toward evolvable hardware chips: experiments with a programmable transistor array," in *Proceedings of 7th International Conference on Microelectronics for Neural, Fuzzy and Bio-Inspired Systems*, 1999.

[151] A. Stoica, D. Keymeulen, R. Tawel, R. Salazar-Lazaro, and W. Li, "Evolutionary experiments with a fine-grained reconfigurable architecture for analog and digital CMOS circuits," in *1st NASA/DoD Workshop on Evolvable Hardware*, A. Stoica *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 1999, pp. 76–84.

[152] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, G. Klimeck, Y. Jin, R. Tawel, and V. Duong, "Evolution of analog circuits on field programmable transistor arrays," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 99–108.

[153] A. Stoica, R. Zebulum, and D. Keymeulen, "Mixtrinsic evolution," in *Proc. of the 3rd Int. Conf. on Evolvable Systems (ICES 2000)*, J. Miller *et al.*, Eds. Heidelberg: Springer-Verlag, 2000, pp. 208–217.

[154] ——, "Progress and challenges in building evolvable devices," in *3rd NASA/DoD Workshop on Evolvable Hardware*, D. Keymeulen *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2001, pp. 33–35.

[155] F. Streichert, C. Spieth, H. Ulmer, and A. Zell, "How to evolve the head-tail pattern from reaction-diffusion systems," in *2004 NASA/DoD Conference on Evolvable Hardware*, R. Zebulum *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 261–268.

[156] I. Tagkopoulos, C. Zukowski, G. Cavelier, and D. Anastassiou, "A custom FPGA for the simulation of gene regulatory networks," in *Proceedings of the 13th ACM Great Lakes symposium on VLSI*. New York, NY, USA: ACM Press, 2003, pp. 132–135.

[157] L. F. Tammero and M. H. Dickinson, "The influence of visual landscape on the free flight behavior of the fruit fly drosophila melanogaster," *The Journal of Experimental Biology*, vol. 205, pp. 327–343, 2002.

[158] G. Tempesti, D. Mange, E. Petraglio, A. Stauffer, and Y. Thoma, "Developmental processes in silicon: An engineering perspective," in *2003 NASA/DoD Conference on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2003, pp. 255–264.

[159] C. Teuscher, *Turing's Connectionism. An investigation of neural network architectures*. Heidelberg: Springer-Verlag, 2002.

[160] Y. Thoma, *Description of the Organic Subsystem of the POEtic Tissue*, Swiss Federal Institute of Technology, Logic Systems Laboratory, Lausanne, Switzerland, 2004.

[161] Y. Thoma, E. Sanchez, J.-M. Moreno Arostegui, and G. Tempesti, "A dynamic routing alogrithm for a bio-inspired reconfigurable circuit," in *Proceedings of the 13th International Conference on Field Programmable Logic and Applications (FPL'03)*. Heidelberg: Springer-Verlag, 2003, pp. 681–690.

[162] Y. Thoma, E. Sanchez, D. Roggen, C. Hetherington, and J.-M. Moreno, "Prototyping with a bio-inspired reconfigurable chip," in *Proc. of the 15th IEEE Int. Workshop on Rapid System Prototyping (RSP'04)*. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 239–246.

[163] Y. Thoma, G. Tempesti, E. Sanchez, and J.-M. Moreno Arostegui, "POEtic: An electronic tissue for bio-inspired cellular applications," *BioSystems*, vol. 76, pp. 191–200, 2004.

[164] A. Thompson and P. Layzell, "Evolution of robustness in an electronics design," in *Proc. of the 3rd Int. Conf. on Evolvable Systems (ICES 2000)*, J. Miller *et al.*, Eds. Heidelberg: Springer-Verlag, 2000, pp. 218–228.

[165] A. Thompson, "An evolved circuit, intrinsic in silicon, entwined with physics," in *Proc. of the 1st Int. Conf. on Evolvable Systems (ICES 96)*, T. Higuchi *et al.*, Eds. Berlin: Springer-Verlag, 1996, pp. 390–405.

[166] ——, "Silicon evolution," in *Proceedings of Genetic Programming 96*, J. R. Koza *et al.*, Eds. MIT Press, 1996, pp. 444–452.

[167] A. Thompson, I. Harvey, and P. Husbands, "Unconstrained evolution and hard consequences," in *Towards Evolvable Hardware*, E. Sanchez and M. Tomassini, Eds. Heidelberg: Springer-Verlag, 1996, pp. 136–165.

[168] K. Tomita, S. Murata, H. Kurokawa, E. Yoshida, and S. Kokaji, "Self-assembly and self-repair method for a distributed mechanical systems," *IEEE Trans. on Robotics and Automation*, vol. 15, no. 6, pp. 1035–1045, 1999.

[169] O. Torres, "Personal communication," April 2004.

[170] O. Torres, J. Eriksson, J. M. Moreno, and A. Villa, "Hardware Optimization and Serial Implementation of a Novel Spiking Neuron Model for the POEtic Tissue," in *BioSystems Journal: proceedings of IPCAT'03*. Elsevier Science, 2003.

[171] ——, "Hardware Optimization of a Novel Spiking Neuron Model for the POEtic tissue," in *Proc. of IWANN'03*, J. Mira, Ed. Heidelberg: Springer-Verlag, 2003, pp. 113–120.

[172] O. Torres, "POEtic Deliverable 21: Demonstration of a sample application on an OE hardware prototype," 2004.

[173] J. Torresen, "Possibilities and limitations of applying evolvable hardware to real-world applications," in *10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, 2000.

[174] ——, "Scalable evolvable hardware applied to road image recognition," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 245–252.

[175] G. Tufte and P. C. Haddow, "Biologically-Inspired: A Rule-Based Self-Reconfiguration of a Virtex Chip," in *Proc. of the 4th Int. Conf. on Computational Science (ICCS 2004)*, M. Bubak, G. D. van Albada, P. M. A. Sloot, *et al.*, Eds. Heidelberg: Springer-Verlag, 2004, pp. 1249–125.

[176] ——, "Building knowledge into developmental rules for circuit design," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 69–80.

[177] A. M. Turing, "The chemical basis of morphogenesis," *Philosophical Transactions of the Royal Society of London*, vol. B, 237, no. 641, pp. 37–72, 1952.

[178] A. M. Tyrrell, E. Sanchez, D. Floreano, G. Tempesti, D. Mange, J.-M. Moreno, J. Rosenberg, and A. Villa, "POEtic Tissue: An Integrated Architecture for Bio-Inspired Hardware," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 129–140.

[179] J. Vaario, S. Ohsuga, and K. Hori, "Connectionist modeling using Lindenmayer systems," in *Information Modeling and Knowledge Bases: Foundations, Theory, and Applications*, Ohsuga *et al.*, Eds. IOS Press, 1991, pp. 496–510.

[180] V. K. Vassilev, D. Job, and J. F. Miller, "Towards the automatic design of more efficient digital circuits," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 151–160.

[181] V. K. Vassilev and J. F. Miller, "The advantages of landscape neutrality in digital circuit evolution," in *Proc. of the 3rd Int. Conf. on Evolvable Systems (ICES 2000)*, J. Miller *et al.*, Eds. Heidelberg: Springer-Verlag, 2000, pp. 252–263.

[182] ——, "Scalability problems of digital circuit evolution: Evolvability and efficient designs," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 55–64.

[183] M. S. Vickery and J. B. McClintock, "Regeneration in metazoan larvae," *Nature*, vol. 394, p. 140, 1998.

[184] B. Webb and T. Scutt, "A simple latency-dependent spiking-neuron model of cricket phonotaxis," *Biol. Cybern.*, vol. 82, pp. 247–269, 2000.

[185] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS," *IEEE Trans. on Image Processing*, vol. 9, no. 8, pp. 1309–1324, 2000.

[186] J. A. White, J. T. Rubinstein, and A. R. Kay, "Channel noise in neurons," *Trends Neurosci.*, vol. 23, no. 3, pp. 131–137, 2000.

[187] L. Wolpert, "Do we understand development?" *Science*, vol. 266, no. 28, pp. 571–572, 1994.

[188] ——, *Principles of Development*. Oxford: Oxford University Press, 1998.

[189] Xilinx, *XC6200 field programmable gate arrays. Datasheet*, Xilinx Inc., Xilinx Web Site (http://www.xilinx.com), April 1997.

[190] X. Yao, "A review of evolutionary artificial neural networks," *International Journal of Intelligent Systems*, vol. 4, pp. 203–222, 1993.

[191] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 29, no. 1, pp. 87–97, February 1999.

[192] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. on Evolutionary Computation*, vol. 3, no. 2, pp. 82–102, July 1999.

[193] R. S. Zebulum, M. A. Pacheco, and M. Vellasco, "Analog circuits evolution in extrinsic and intrinsic modes," in *Proc. of the 2nd Int. Conf. on Evolvable Systems (ICES 98)*, M. Sipper *et al.*, Eds. Heidelberg: Springer-Verlag, 1998, pp. 154–165.

[194] R. S. Zebulum, C. C. Santini, H. T. Sinohata, M. A. Pacheco, M. Vellasco, and M. H. Szwarcman, "A reconfigurable platform for the automatic synthesis of analog circuits," in *2nd NASA/DoD Workshop on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2000, pp. 91–98.

[195] Zetex, *Zetex TRAC020LH datasheet*, Zetex, Zetex Web Site (http://www.zetex.com/), March 1999.

Curriculum vitæ

Daniel Roggen studied at EPFL in microengineering where he obtained the master degree in 2001 with specialization in integrated products. His master project dealt with vision-based mobile robot navigation using evolvable hardware.

He worked during a short period as a R&D engineer at VisioWave, a company active in the field of digital video, where he performed key algorithmic and implementation optimizations of the company's wavelet-based video compression system.

He then started a PhD funded by the European project POEtic at the Autonomous Systems Laboratory of the Swiss Federal Institute of Technology in Lausanne, in the field of bio-inspired electronics. He worked on evolutionary algorithms for bio-inspired reconfigurable multi-cellular circuits, and on the applications and implementation of these circuits.

His research interests include bio-inspired electronics and robotics, signal and image processing and embedded systems.

Publications

Peer-reviewed proceedings

A. Koopman and D. Roggen, "Evolving Genetic Regulatory Networks for Hardware Fault Tolerance," in *Proc. of Parallel Problem Solving from Nature VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, Eds. Heidelberg: Springer-Verlag, 2004, pp. 561–570.

D. Roggen and D. Federici, "Multi-cellular development: is there scalability and robustness to gain?" in *Proc. of Parallel Problem Solving from Nature VIII*, X. Yao, E. Burke, J. A. Lozano, J. Smith, J. J. Merelo-Guervós, J. A. Bullinaria, J. Rowe, P. Tiňo, A. Kabán, and H.-P. Schwefel, Eds. Heidelberg: Springer-Verlag, 2004, pp. 391–400.

D. Roggen, Y. Thoma, and E. Sanchez, "An Evolving and Developing Cellular Electronic Circuit," in *Proceedings of Artificial Life IX*, J. Pollack, M. Bedau, P. Husbands,

T. Ikegami, and R. A. Watson, Eds., 2004, pp. 33–38.

Y. Thoma, E. Sanchez, D. Roggen, C. Hetherington, and J.-M. Moreno, "Prototyping with a bio-inspired reconfigurable chip," in *Proc. of the 15th IEEE Int. Workshop on Rapid System Prototyping (RSP'04)*. Los Alamitos, CA: IEEE Computer Society Press, 2004, pp. 239–246.

D. Roggen, S. Hofmann, Y. Thoma, and D. Floreano, "Hardware spiking neural network with run-time reconfigurable conectivity in an autonomous robot," in *2003 NASA/DoD Conference on Evolvable Hardware*, J. Lohn *et al.*, Eds. Los Alamitos, CA: IEEE Computer Society Press, 2003, pp. 189–198.

D. Roggen, D. Floreano, and C. Mattiussi, "A Morphogenetic Evolutionary System: Phylogenesis of the POEtic Tissue," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 153–164.

G. Tempesti, D. Roggen, E. Sanchez, Y. Thoma, R. Canham, and A. M. Tyrrell, "Ontogenetic Development and Fault Tolerance in the POEtic Tissue," in *Proc. of the 5th Int. Conf. on Evolvable Systems (ICES 2003)*, A. M. Tyrrell *et al.*, Eds. Heidelberg: Springer-Verlag, 2003, pp. 141–152.

G. Tempesti, D. Roggen, E. Sanchez, and Y. Thoma, "A poetic architecture for bio-inspired hardware," in *Proceedings of Artificial Life VIII*, R. K. Standish, B. M. A., and A. H. A., Eds. Cambridge, MA: MIT Press, 2002, pp. 111–115.

Journals

D. Roggen, D. Federici, and D. Floreano, "Evolutionary morphogenesis for multi-cellular systems," *Submitted to IEEE Trans. on Evolutionary Computation*, 2004.

Book chapters

D. Floreano, F. Mondada, A. Perez-Uribe, and D. Roggen, "Evolution of embodied intelligence," in *Embodied Artificial Intelligence*, F. Iida, R. Pfeifer, L. Steels, and Y. Kuniyoshi, Eds. Heidelberg: Springer-Verlag, 2004.