BIO-INSPIRED CELLULAR MACHINES: TOWARDS A NEW ELECTRONIC PAPER ARCHITECTURE

THÈSE N^{O} 3933 (2007)

PRÉSENTÉE LE 7 DÉCEMBRE 2007

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS
Groupe Sanchez
SECTION D'INFORMATIQUE

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

Fabien VANNEL

Ingénieur en Electronique et Electrotechnique, Ecole supérieure d'Ingénieurs (ESIEE), Paris, France
et de nationalité française

ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Lausanne, EPFL
2007

A ma grand-mère.

Version abrégée

L'informatique est une invention très récente, vieille de quelques décennies seulement. Même si les prémices du calcul automatisé sont bien plus anciens (W. Schickard construisit la première calculatrice mécanique en 1623), il faudra attendre l'invention du transistor par W. Shockley, J. Bardeen et W. Brattain en 1947 pour propulser les calculateurs hors des laboratoire et observer l'omniprésence des systèmes informatiques dans le quotidien de chaque être humain.

Les ordinateurs ne se contentent plus d'être des systèmes très performants, capables d'exécuter des milliards d'opérations par seconde, mais envahissent notre monde en se glissant dans les moindres recoins de notre environnement. De la montre à quartz à l'ordinateur de bureau, en passant par la clé de voiture, le téléphone portable, la télévision, ou la carte de crédit, les microprocesseurs sont omniprésents; il est fort probable qu'ils seront encore plus nombreux demain, et qu'ils pourront se nicher dans nos vêtements ou notre journal.

La recherche incessante de systèmes de plus en plus puissants, plus robustes, plus intelligents, ne passe pas uniquement par l'amélioration des technologies de fabrication des puces électroniques, mais également par la recherche de nouvelles architectures informatiques. Pour atteindre ces objectifs une des voies de recherche est la bio-inspiration. La nature est fascinante pour l'ingénieur: quoi de plus intelligent, adaptatif, évolutif et robuste qu'un être vivant? D'une simple cellule possédant son plan de construction sous forme d'ADN va se développer un être multi-cellulaire complet. Ces particularités de la nature ont souvent été étudiées et mimées afin de concevoir des systèmes artificiels adaptatifs, robustes et insensibles aux défaillances. Le modèle POE résume les trois grandes sources de la bio-inspiration: l'évolution des espèces (P: phylogenèse), le développement par division et différenciation d'un être multi-cellulaire (O: ontogenèse) et l'apprentissage par interaction avec l'environnement (E: épigenèse).

Cette thèse se propose d'approfondir l'étude de l'axe ontogenétique du modèle POE, grâce à l'étude de trois machines informatiques cellulaires tout à fait originales dont l'élément de base satisfait les six caractéristiques suivantes: (1) il est reconfigurable, (2) de complexité minimale, (3) présent en grand nombre, (4) interconnecté avec ses voisins de façon locale, (5) disposant d'un système d'affichage et (6) d'un capteur permettant une interaction minimale.

Le BioWall, qui est notre première réalisation, est constitué d'une surface de 4'000 éléments de base ou molécules, aptes à réaliser tous les systèmes cellulaires comportant au plus 160×25 éléments.

La seconde réalisation, le BioCube, transpose l'architecture à deux dimensions du BioWall dans un espace à trois dimensions, limitée à $4 \times 4 \times 4 = 64$ éléments de base ou sphères, cherche à préfigurer un ordinateur tridimensionnel construit à partir des nanotechnologies.

Quant à la troisième machine, le BioTissue, elle reprend les hypothèses du BioWall tout en poussant ses performances aux limites des techniques actuelles tout en lui ajoutant les atouts d'un système autonome.

La convergence de ces trois réalisations, étudiées dans le contexte des technologies émergentes, nous amène à définir et proposer l'architecture informatique de l'ordinateur du futur: le tissu ou papier électronique.

Mot-clés: système bio-inspiré, ordinateur cellulaire, système reconfigurable, conception électronique, architecture des ordinateurs, Embryonique, tolérance aux pannes, FPGA, nanotechnologies, papier électronique.

Abstract

Information technology has only been around for about fifty years. Although the beginnings of automatic calculation date from as early as the 17^{th} century (W. Schickard built the first mechanical calculator in 1623), it took the invention of the transistor by W. Shockley, J. Bardeen and W. Brattain in 1947 to catapult calculators out of the laboratory and produce the omnipresence of information and communication systems in today's world.

Computers not only boast very high performance, capable of carrying out billions of operations per second, they are taking over our world, working their way into every last corner of our environment. Microprocessors are in everything, from the quartz watch to the PC via the mobile phone, the television and the credit card. Their continuing spread is very probable, and they will even be able to get into our clothes and newspapers.

The incessant search for increasingly powerful, robust and intelligent systems is not only based on the improvement of technologies for the manufacture of electronic chips, but also on finding new computer architectures. One important source of inspiration for the research of new architectures is the biological world. Nature is fascinating for an engineer: what could be more robust, intelligent and able to adapt and evolve than a living organism? Out of a simple cell, equipped with its own blueprint in the form of DNA, develops a complete multi-cellular organism. The characteristics of the natural world have often been studied and imitated in the design of adaptive, robust and fault-tolerant artificial systems. The POE model resumes the three major sources of bio-inspiration: the evolution of species (P: phylogeny), the development of a multi-cellular organism by division and differentiation (O: ontogeny) and learning by interaction with the environment (E: epigenesis).

This thesis aims to contribute to the ontogenetic branch of the POE model, through the study of three completely original cellular machines for which the basic element respects the six following characteristics: it is (1) reconfigurable, (2) of minimal complexity, (3) present in large numbers, (4) interconnected locally with its neighboring elements, (5) equipped with a display capacity and (6) with sensor allowing minimal interaction.

Our first realization, the BioWall, is made up of a surface of 4,000 basic elements or molecules, capable of creating all cellular systems with a maximum of 160×25 elements.

The second realization, the BioCube, transposes the two-dimensional architecture of the BioWall into a two-dimensional space, limited to $4 \times 4 \times 4 = 64$ basic elements or spheres. It prefigures a three-dimensional computer built using nanotechnologies.

The third machine, named BioTissue, uses the same hypothesis as the BioWall while pushing its performance to the limits of current technical possibilities and offering the benefits of an autonomous system.

The convergence of these three realizations, studied in the context of emerging technologies, has allowed us to propose and define the computer architecture of the future: the electronic paper.

Keywords: bio-inspired hardware, cellular computing, reconfigurable computing, electronic design, computer architecture, Embryonics, fault tolerance, FPGA, nanotechnologies, electronic paper.

Chapter 1

Introduction

WHAT would the words *computer* or *computer science* have meant for our grandparents? Probably an inconceivable revolution. For younger people, these concepts are part of the fabric of our world. In less than 50 years, the invention of the transistor has projected the computer from the laboratories to the public place, where it is now found everywhere: watches, mobile phones, music players, cars, credit cards...

Computing needs are evolving towards faster microprocessors and the conquest of new targets and applications. The newspaper becoming a computer [92], and the integration of processors in clothes [79] are just two examples. Computers are also becoming smaller and thinner - less noticeable - but increasing by present in our everyday world.

1.1 Bio-inspired cellular computing

Up to now, the main research in computing has focused on increasing the computational performance of microprocessors. This has been achieved mainly through reducing the size of etching silicon technologies making it possible to increase the operating frequency of the microprocessors. Recently, the demands for ever increasing computing power have been addressed by designing architectures capable of exploiting parallelism at the instruction level through hardware mechanisms such as super-scalar execution. However, all these approaches seem to have reached their practical limits, mainly due to issues related to design complexity and cost-effectiveness.

The current trend in computer design seems to favor a switch to coarser-grain parallelization, typically at the thread level. In other words, high computational power is achieved not by a single very fast and very complex microprocessor, but through the parallel operation of several on-chip processors, each executing a single thread. This kind of approach is currently implemented commercially through multi-core processors and in the research community through the *Multi-processors Systems On Chip* (MPSoCs) term, which is itself largely based on the *Network On Chip* (NoC) paradigm [34, 33].

Extrapolating this trend to take into account the vast amount of on-chip hardware resources that will be available in the next few decades (either through further shrinkage of silicon fabrication processes or by the introduction of molecular-scale devices), together with the predicted features of such devices (i.e. the impossibility of global synchronization), this approach comes to resemble another computational paradigm, commonly known as *cellular computing*.

Loosely based on the observation that biological organisms are in fact highly complex structures, which function thanks to the parallel operation of vast numbers of relatively simple elements (the cells), this paradigm tries to draw an analogy between multi-cellular organisms and multi-processor systems. At the root of this analogy is the observation that organisms, in addition to being completely asynchronous, are built through a bottom-up self-assembly process and do not require the specification of a complete layout.

The existing interpretations and implementations of this paradigm are extremely varied, ranging from theoretical studies [115, 118] to commercial realizations (notably, the *Cell* CPU [97, 98] jointly developed by IBM®, Sony® and Toshiba®), through wetware-based systems [9], OS-based mechanisms [49] and amorphous computing approaches [2].

Depending on the authors, the *cells* may comprise different levels of complexity ranging from very simple, locally-connected, logic elements to high-performance computing units endowed with memory and complex network capabilities. However, in every case, the basic idea of two-dimensional systems composed of relatively simple connected elements, remains.

1.2 Thesis goal

The objective of this thesis is to study different systems based on a cellular architecture. From the simplest architecture, which can nonetheless do complex computation thanks to its huge size, to the most complex cellular system able to challenge today's computer performances, this thesis will discuss how cellular systems can been applied to solve various problems, and be included in future computing systems. Moreover, newcomer technologies like molecular electronics, or nanoelectronics, will probably be made up of regular arrays interconnected together locally [130]. This thesis will study an algorithm able to configure a cellular network of any size, and which includes fault tolerance capabilities. Our final goal is to propose an architecture which could be used for manufacturing a product of the future: electronic paper.

It has long been a practice of our research laboratory to attempt to physically realize, in hardware, systems in order to verify their properties and to analyze their efficiency. Considering the complexity of this kind of systems, prototyping in hardware requires vast amounts of reconfigurable resources, which for this project take the form of Field Programmable Gate Array (FPGA) components, as specified in this thesis. Our realizations are all based on a recurrent basic topology: each cell of our cellular machines is endowed with at least the following three key components: a reconfigurable computing unit, a display element and an external sensor allowing users to interact with the cell.

1.3 Thesis organization

This thesis is divided into 10 chapters. Chapters 3 to 8 present our three realizations. The presentation of each of the three realizations is structured in the same way: a first chapter describes the hardware machine, and a second chapter focuses on applications.

Chapter 2 provides a rather general overview of the world of computer hardware. It summarizes the evolution of the computer from its birth to the current models, and makes some predictions concerning its future. We also outlined the evolution of technologies and of computer hardware architectures. As the focus for the past several decades has been on speeding up processing by miniaturization, we will discuss some new research topics like nanotechnologies for technology improvements, and cellular computing for architecture improvements.

1.3.1 Three biologically-inspired machines

The next chapters describe the three bio-inspired machines developed during this thesis. They are all based on a same common cellular architecture, where several identical units are interconnected using a mesh-network topology. At the heart of each of our units, is an FPGA, which allows several experiments to be easily conducted thanks to its reconfigurable properties and computing performance.

Chapter 3 deals with our first invention: the BioWall. This bio-inspired electronic tissue was firstly developed to study and test the ontogenetic model described in the Embryonics project in hardware. The BioWall is made of 4'000 identical molecules and is also the largest of the three realizations in terms of size. The results obtained from this machine are very promising, but the BioWall needs some additional features and improvements in order to supply the autonomy needed to accomplish our final goal: the electronic paper.

Chapter 5 extends the concept of the BioWall from a wall in 2D to a cube in 3D. This second invention, called the BioCube, proposes an architecture that could be used with promising circuits of the future based on nanotechnologies. Several improvements towards autonomy have been added to this cellular system.

Chapter 7 presents our final realization: the BioTissue. It is the outcome of our current research in terms of systems conceived in hardware. This tablet, still based on a cellular architecture, similar to the BioWall global structure, is fully autonomous and can run several different applications at the same time. Its structure is a precursor of tomorrow's electronic paper architecture.

1.3.2 Two common applications

For the purpose of comparing our three architectures, we developed two different applications, that were both implemented in the three different machines. The first application, the Game of Life, will be a useful tool for comparing the performance of each machine and a good example to show how to develop an application on the three systems. The second application, the Tom Thumb replicating loop, is a perfectly dedicated application, which is used for investigating the bio-inspired capabilities of each machine. Improvements to this application are added in each new implementation (depending on the resources of each machine) and the final version implemented onto the BioTissue is the current result of our research on this topic.

Chapter 4 describes these two applications running onto the BioWall, chapter 6 does the same for the BioCube, and chapter 8 shows the evolution of these applications running on the BioTissue.

The Game of Life

Our first application is a simple Cellular Automata (CA): John Conway's Game of Life [47]. It is probably the best known CA. Chapter 4 will present the principles of this automaton. We chose this application for several reasons:

- This simple automaton is easy to implement on our three realizations.
- Since it is a cellular automaton, it is well suited to our cellular architectures which are perfectly adapted for parallel computation.
- The biological aspects found in this automaton match the bio-inspired quality of our research. "Because of Life's analogies with the rise, fall and alterations of a society of living organisms, it belongs to a growing class of what are called 'simulation games' (games that resemble real life processes)" [47].
- From these simple, local rules, some astoundingly complex global behaviors have been observed and developed, and a tissue with a huge number of automaton cells allows the Game of Life to be programmed to perform computations. Indeed, it is a universal computer [101]. The perfect example of its universality is the implementation of the Turing machine [105].
- Another capacity of this automaton is the self-reproduction of organisms [14], which is part of our research topic.

This application will be tailored to the three systems.

Self-replicating loop

The second application is a novel self-replicating loop endowed with universal construction properties, perfectly suited to the regular cellular structure of our systems. The goal of this application is to perform configuration, cloning, cicatrization and regeneration mechanisms through a single process inspired from the biological world.

Self-replicating and self-repairing machines could be used in the development of new physical architectures where defects and failures are common. This is not the case with our current silicon technologies, but could soon be used with the development of molecular electronics where manufacturing defects may be present in many parts of the component. This application proposes a new methodology for configuring a regular network of computational units, even if some units are defective. Our research focuses on a 2D and a 3D network topologies. The cicatrization and regeneration mechanisms have only been added in the last version of our application, i.e. the application running on the BioTissue, as the latter is complex enough to support these processes.

1.3.3 The electronic paper

The final chapter before the conclusion is a summary of our three machines and a proposal for a novel architecture, the electronic paper. Chapter 9 focuses on a possible future system, which would allow computing power to invade our newspapers and our clothes. Our research could be adapted for products like electronic paper. In this chapter, we will discuss how our cellular architectures could be perfectly suited for this kind of product, and for the manufacturing processes of these tissues. Primarily the research done on the BioTissue but also the research on the BioCube will help us to define an architecture specially suited for an electronic tissue, which will hopefully become common after some years once the appropriate manufacturing technologies are be ready.

Chapter 10 resumes our work and gives some ideas for future researches based on our thesis work.

1.4 Contributions

The following major contributions were important in the realization of this thesis work:

1.4.1 BioWall

The BioWall was a huge and time consuming realization which involved many researchers of the Logic Systems Laboratory (LSL). As one of the major developers of this machine, I proposed and developed several features for the control and the configuration of this wall: FPGA configuration process, clocking and synchronization of the FPGAs, system supervising, interfaces with external modules (remote control, audio, video). In addition to the hardware side of this machine, important work was done to simplify the implementation of the applications described in chapter 4; many reusable Intellectual Property (IP) cores have been developed for the SPARTAN® FPGA. The study and implementation of the applications running on this wall was conducted in collaboration with the BioWall LSL team.

1.4.2 BioCube

The BioCube is an evolution of the BioWall architecture towards the third dimension. Since the dimensions of this new machine are smaller than those of the BioWall, and although it is just as complex or more, I found myself nearly alone to define and build this new system. The definition of the new architecture, the development of the hardware devices and all stages of testing the prototypes were accomplished by myself. The analysis of the capabilities and performance of this machine was carried out via the development of applications as seen in chapter 6. The research concerning the 3D Tom Thumb loop was a collaborative work with researchers of the LSL.

1.4.3 BioTissue

The BioTissue architecture is a novel proposal inspired by the BioWall architecture, but completely revisited to match the new needs I specified to create a powerful, autonomous cellular machine whose performance is closer to that of an actual Personal Computer (PC) than to a dedicated cellular research tool. Research done on the BioTissue before its construction has always been in connection with the idea of defining and building a machine the architecture of which could fit the needs of a possible electronic paper machine of the future. As well as defining the whole architecture of the BioTissue, I built the machine. The need for an operating system led me to imagine and to implement the μOS application. The help of the LSL researchers was needed to update the Tom Thumb algorithm to create the *eSOS* application described in chapter 8.

1.4.4 Electronic paper

This thesis would not be complete if we didn't look ahead and find a common future for our three cellular machines. The predicted electronic newspaper product [96], which will be similar to the current paper version, led us to propose an architecture based on our research, which could match the new printing fabrication process of electronic components, while being perfectly adapted to any flat computer like the electronic paper.

Chapter 2

The computer through the ages

THE twenty century was a fascinating era with many memorable developments. Since then planes have developed from the first prototype made of wood to the huge Airbus A380, able to carry more than 500 passengers and automobiles have colonized the city. Another evolution, or maybe revolution, happened in less than half a century: the invention of the semiconductor and the democratization of its main application, the computer. In a few decades the computer went from a huge, expensive and slow machine dedicated to a single simple task, to incredibly powerful computers, small, inexpensive, interactive, multitasking, present in each family, and connected all together via the Internet.

This chapter will focus on the computer's history from its first prototype to the current omnipresent machines, and will make some predictions for its future evolution. This subject could fill a book, or even an encyclopedia, let alone a chapter. Studying the history of computers since the first prototype to the current powerful microprocessors is not the aim of this chapter, which will only give some major points which will help understand the research contained in this thesis paper.

Section 2.1 presents a short overview of the main events in the history of the computer and more particularly its main component, the microprocessor. Section 2.2 presents the current microprocessor architectures, and the different components allowing computation. The last section 2.3 is consecrated to our vision for improving microprocessor performance, such as new research into materials, or the introduction of bio-inspired architectures.

2.1 The birth of the computer

Originally, the term "computer" did not refer to a computing machine, but referred to a person who performed numerical calculations [50]. Such a human computer often worked with the aid of mechanical calculating devices like abacus, slide rules or astrolabes.

2.1.1 The first computing machine

If a computing machine is a device able to input data, compute it and produce a result, then the first computing machine is the *Antikythera mechanism* [109, 35] dating from around 80 BC, designed to calculate astronomical positions. Nothing so complex was subsequently developed before the beginning of the 17^{th} century, when Wilhelm Schickard conceived the first calculating machine able to accomplish addition, subtraction, multiplication and division [123]. At the same time, Gottfried Leibniz published an article describing the principles of binary arithmetic, the basis of our modern computing numeral system. Two centuries later, George Boole published in 1854 a landmark paper detailing a logic system that would become known as Boolean algebra. The next step toward the implementation of Boolean algebra was performed by Claude Shannon in 1937, who used electronic relays and switches to include binary arithmetic in hardware, the basis of modern computing and practical digital circuit design.

During the first half of the 20^{th} century, several machines were built, mainly for military purposes. The Electronic Numerical Integrator And Computer (ENIAC) [109], the construction of which started in 1943, was the first large-scale, electronic, digital computer capable of being reprogrammed (by rewiring the machine) to solve a full range of computing problems. Physically, ENIAC (Fig. 2.1) was a massive machine of 27 tons, roughly sized $30m \times 2.4m \times 0.9m$, and consuming 150 kW of power. This computer was built with 17,468 vacuum tubes, 7,200 crystal diodes, 1,500 relays, 70,000 resistors, 10,000 capacitors and around 5 million hand-soldered joints. Data to process were input using a punched card reader, while a punched card perforator wrote the results. The ENIAC machine could perform 385 multiplication operations per second, but needed to be partially rewired to update its program. Two years after the start of ENIAC, in 1945, John von Neumann, while he was working on the development of the Electronic Discrete Variable Automatic Computer (EDVAC), presented a paper describing the complete design of a stored-program computer, the architecture of which, named “von Neumann architecture”, is still used nowadays.

Most of these machines were made of vacuum tubes, which had a notable disadvantage, specially when used in huge number: they tended to leak, and the metal that emitted electrons burned out. Moreover, tubes were big and required so much power that big machines were too large and took too much energy to run. These problems led scientists and engineers to think of other ways to perform the tubes’ tasks by looking at how one might control electrons in solid materials, like metals and semiconductors.

2.1.2 The invention of the transistor

In 1947, John Bardeen and Walter Brattain, working at Bell Telephone Laboratories under the direction of William Shockley, were trying to understand the nature of the electrons at the interface between a metal and a semiconductor, when they realized that by putting two point contacts very close to each other, they could get similar results as a three-electrode tube. They had just invented the first “point contact” transistor. (Fig. 2.2)

Their first application was to build a few of these transistors and to connect them with some other components in order to develop an audio amplifier. This device was

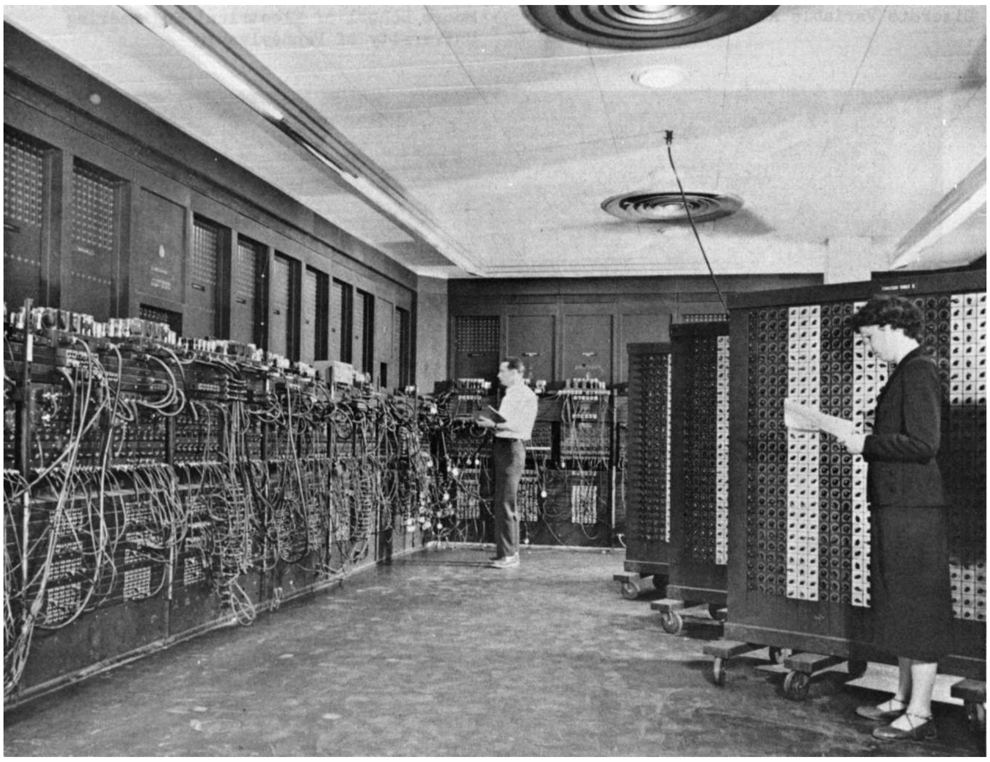


Figure 2.1: *The ENIAC machine.*

shown to chief executives at Bell Telephone Company, who were very impressed by the fact that it did not need time to warm up before use (unlike the heaters in vacuum tube circuits). The importance of this new technology was immediately realized and this invention was the spark that ignited a huge research effort in solid state electronics. William Shockley developed an improved version of this “point contact” transistor, the junction transistor, which was built on thin slices of different types of semiconductor material pressed together. The junction transistor was easier to understand theoretically, and could be manufactured more reliably. The amazing thing about the invention of this “point contact” transistor is that the first patent for the field-effect transistor principle was registered in Germany in 1928 by the physicist Julius Edgar Lilienfeld. Unfortunately, Lilienfeld did not publish any research articles about his device, and his work was ignored by industry.

For many years, transistors were manufactured as individual electronic components and replaced vacuum tubes in computers. Electronic circuits could be made more complex, with more transistors, switching faster than tubes and consuming much less power. However, it did not take a long time before the limits of this construction technique were reached. Circuits based on transistors became too large and too difficult to assemble since there were too many electronic components to deal with. Global computer performance was limited because the time taken for electric signals to propagate over a long distance could not be reduced any further. To make the circuits even faster, one needed to pack the transistors closer and closer together.

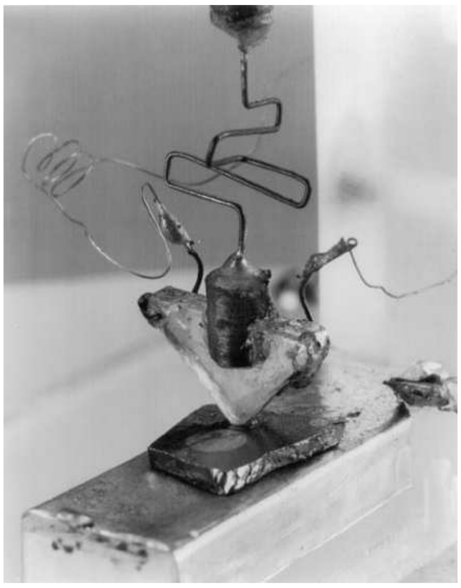


Figure 2.2: *The first point contact transistor developed by J. Bardeen and W. Brattain. Paper clips and razor blades were used to make the device.*

2.1.3 The integrated circuit

In 1958 and 1959, Jack Kilby at Texas Instruments and Robert Noyce at Fairchild Semiconductor, came up, almost simultaneously, with a solution to the problem of packaging a large number of components, and the Integrated Circuit (IC) was born (Fig. 2.3). Instead of making and assembling transistors one-by-one, several transistors could be manufactured at the same time, in the same piece of semiconductor. Not only transistors, but other electric components such as resistors, capacitors and diodes could be realized by the same process into the same material. The first IC contained only around a dozen transistors. Since this time the number of transistors inside an IC has continuously increased, along with the complexity and performance of computers build upon these ICs.

Nowadays, ICs still use the same principle as the concepts developed by J. Kilby [65] and R. Noyce [89]. The structure is still two dimensional and only differs in the substrate used, where silicon has mainly replaced germanium.

2.1.4 From the microprocessor to the Personal Computer (PC)

For years, new ICs were built for each new product. The methodology was the same; IC manufacturers got the specifications from their customer and designed a circuit able to compute the customer's needs. The story [17] of the invention of the first microprocessor starts with Busicom, a Japanese company specializing in hand-held calculators, which asked Intel® to produce a line of products with different capabilities, each with different advanced mathematical functions, aimed at a different market segment. Instead of manufacturing the multiple requested ICs, Intel® focused on looking for a way to design fewer chips with more general capabilities. In 1971, Intel® were able to

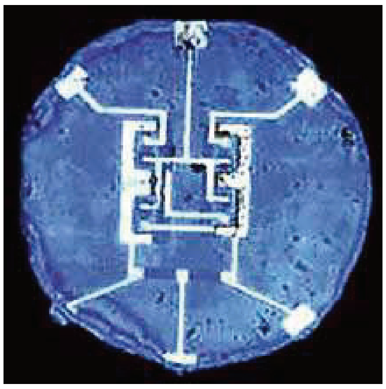


Figure 2.3: *The first IC developed by R. Noyce.*

offer Busicom a logic chip that incorporated more of the concepts of a general purpose computer, and which could be used for many other applications besides calculators. The first microprocessor was born: the Intel® 4004. With the addition of three other dedicated ICs, a simple microprogrammable computer was built. The other chips contained Read-Only Memory (ROM), Random Access Memory (RAM), and a chip to handle output functions. Together, these ICs constituted a computer as powerful as the ENIAC machine described on page 8. After this, several new more powerful models were designed and sold as general purpose microprocessors. Different products could use the same microprocessors, only software running on these microprocessors was different.

During the following decade, several computers based on microprocessors were built and sold by different companies. Software development become more and more important since it allowed the specialization of each computer to a dedicated task and made it possible to switch between programs easily. In 1981, IBM® brought out its first low-cost single-user computer that they dubbed a Personal Computer (PC). This machine came with a monochrome screen, a keyboard and an Operating System (OS). Different kinds of software, for word processing, accounting, spreadsheets and games was developed and sold for use with this kind of machine.

From the time of the first PC, relentless technological advances and innovation have put powerful PCs at the center of daily activities for people worldwide, with more than one billion computers connected to Internet in 2007. The PC has changed the way people communicate, retrieve information, go shopping, and entertain themselves.

2.2 Today's computers

The first Intel® microprocessor, the 4004, was a simple 4-bit model made of 2'300 transistors, which worked at a maximum frequency of 108 KHz. Nowadays, microprocessors are 100'000 times more complex than the 4004. They are built with 64-bit architectures, are made of millions of transistors, and have a core frequency of around 3 GHz. Today, microprocessors are available in a wide range of types, architectures, performance and also price. They range from tiny microprocessors costing less than one US dollar and suited for embedded systems requiring low processing performance,

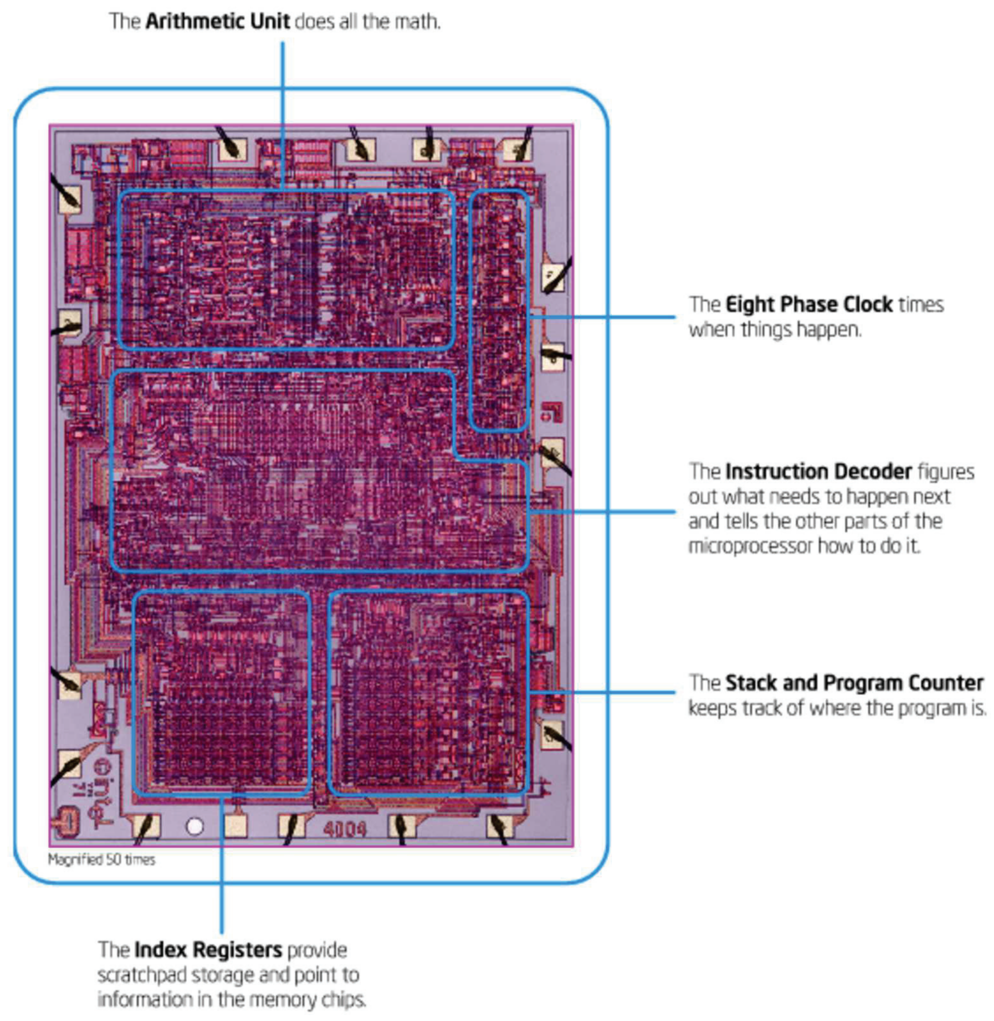

The **Arithmetic Unit** does all the math.
The **Eight Phase Clock** times when things happen.
The **Instruction Decoder** figures out what needs to happen next and tells the other parts of the microprocessor how to do it.
The **Stack and Program Counter** keeps track of where the program is.
The **Index Registers** provide scratchpad storage and point to information in the memory chips.
Magnified 50 times
4004

Figure 2.4: *The first Intel® 4004 microprocessor. Source: Intel®.*

like watches, to extremely powerful models designed for the fastest computers, but which are enormously power hungry and cost more than a thousand US dollars.

Until very recently, the main research on microprocessors has focused on increasing computational performance. The major factor for performing this task has mainly been through reducing the size of etching silicon technologies, resulting in reduction of the transistor size from 10 μm for the Intel® 4004 to 0.065 μm or 65 nm for the latest microprocessor currently on the market. A higher number of transistors can consequently fit onto the same surface, resulting in higher speeds. Another major key to these performance increases comes from the evolution of the microprocessors architecture.

2.2.1 Microprocessor architecture

From the first microprocessor, which was based on a very simple architecture with a 4-bit Arithmetic Logic Unit (ALU), to the 64-bit microprocessors available today, the principle of operation is still the same: microprocessors execute a sequence of stored instructions, or programs, sequentially to compute information coming from memory or a peripheral, and write the result in memory or send it to a device. To perform this task several microprocessor architectures have been developed over the past few decades, and several architecture models are in use today.

Single core

Since speed was the main limitation to technological developments, improvements were made by looking for methods to compute a higher quantity of data during each cycle. Complex Instruction Set Computers (CISCs) and Reduced Instruction Set Computers (RISCs) are two different Central Processing Unit (CPU) architectures which coexist for a long time. Whereas CISC based machines are mainly used in extremely fast and high performance PCs, RISC microprocessors are succeeding in performing the same tasks, at a reduced frequency. RISC architectures are thus often used in tiny or embedded microprocessors working at low power. Several other architectures, theoretically extremely powerful, such as Very Long Instruction Word (VLIW) and Explicitly Parallel Instruction Computing (EPIC) have been invented, but they are not widely used since the development of software for these kinds of CPUs is more complex and thus only used on few highly specialized computers. Aside from the internal CPU structure, several improvements have been made to performance through the addition of cache memory inside the microprocessor chip. Descriptions of all these architectures can be found in several books [91, 53]. Since our thesis work uses several RISC based microprocessors, we describe below some specifications of this architecture.

The Reduced Instruction Set Computer (RISC), is a CPU design philosophy that favors an instruction set reduced both in size and complexity of addressing modes, in order to enable easier implementation, greater instruction level parallelism, and more efficient compilers. Our example in figure 2.5 is a simple 8-bit microcontroller[1] based on a Harvard architecture, which is a computer design model, in which program and data are accessed on separate buses. This improves bandwidth over traditional von Neumann architectures where program and data are fetched on the same bus. Separating program and data memory also allows instructions to be sized differently. The Microchip® PIC10F20x uses, for example, an 8-bit wide data word, and a 12-bit wide instruction, making it possible to have the whole instruction in a single word. The advantage of a RISC structure is the possibility, by using a pipeline, to execute an instruction at each cycle with the exception of program branches, which could need more than one cycle in some cases.

Recently, the ever-increasing demand of computing power has been met by de-

[1]The difference between a Microcontroller unit (MCU) and a microprocessor resides in the additional features included in the same package. An MCU is built upon a microprocessor with embedded memory, and peripherals allowing direct reading and writing on Input/Output (I/O), or including specific communication buses, timers, display drivers...

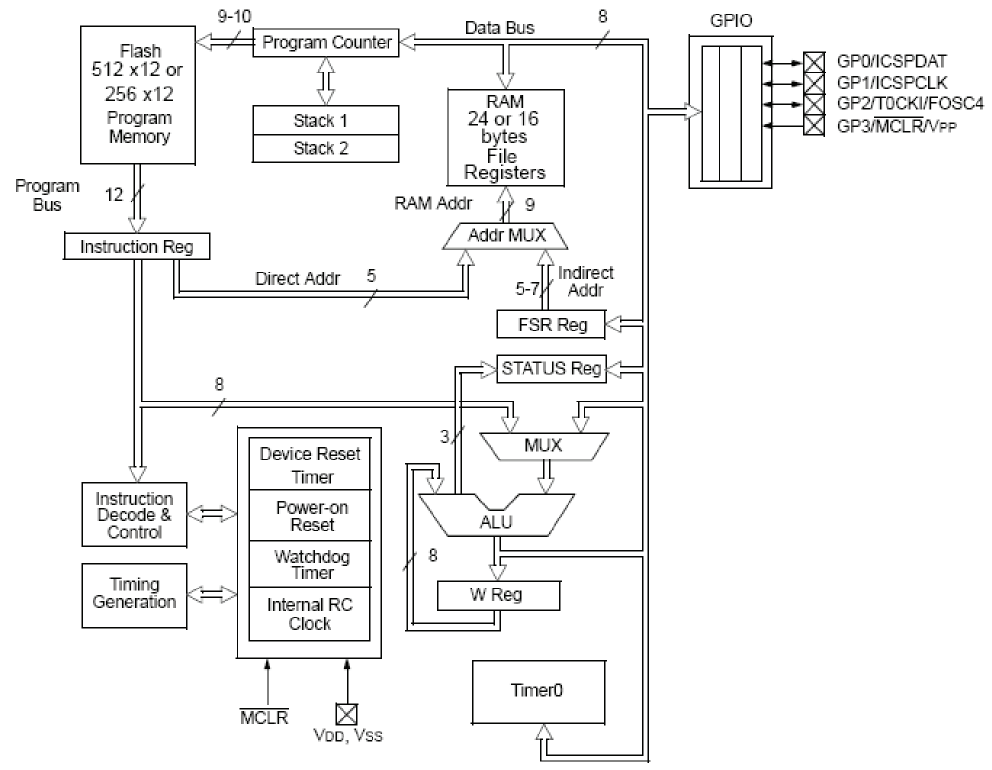


Figure 2.5: *Diagram of the Microchip® RISC PIC10F20x microcontroller based on a Harvard architecture. Source: [57]*

signing architectures capable of exploiting parallelism at the instruction level through hardware mechanisms such as VLIW super-scalar architecture. However, as previously mentioned, all these approaches have reached their practical limits, mainly due to issues related to design complexity and cost-effectiveness.

Multi-cores

For the past few years, a new trend in computer design has favored another methodology to parallelize computations. Rather than using a unique single very fast and very complex microprocessor, parallelism has been reached by the introduction of multi-core microprocessors. A multi-core microprocessor combines into a single physical package, two or more independent single CPUs similar to the one described in the previous paragraph (Fig. 2.6). Each core has its own cache memory and can share a single coherent cache at the highest on-device cache level (e.g. L2 in figure 2.6). The processors also share the same interconnection with the rest of the system. Such architecture with two cores allows concurrent execution of two threads.

Some years after the first dual-core microprocessors, new developments are moving towards processors with quad-, eight- or more cores. Research done on *Multi-processors Systems On Chip* (MPSoCs), and *Network On Chip* (NoC) systems [34,

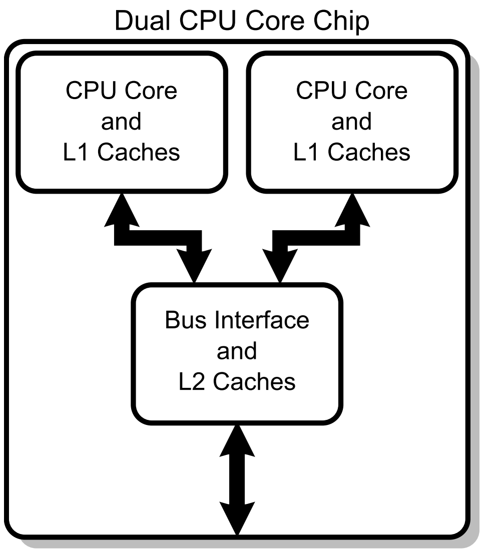


Figure 2.6: *Diagram of a dual core microprocessor, with CPU-local Level 1 caches, and a shared, on-die Level 2 cache.*

33] have, for example, been used for the development of the *Cell* processor [61, 97, 98] jointly developed by IBM®, Sony® and Toshiba®, which include up to eight identical cores (Fig. 2.7). The multi-core structure is currently used for PC microprocessors (with several dual- or quad-core CPUs), or in game consoles, which require highly powerful CPUs (the *Cell* multi-core microprocessor is used in the Sony® PS3). The general trend in this domain will be from multi-core to many-core where the number of independent cores will increase to large numbers of tens or even hundreds, thus allowing as many threads as the number of cores present in the microprocessor to be executed in parallel. The Intel® Tera-scale research project [54] tried to study how by varying the mix of functional elements (with experiment up to 100 cores), the Tera-scale architecture can allow designers to create multiple implementations that match specific market needs.

Since all cores are identical, and rather than manufacturing new chips for each new multi-core configuration, a new packaging method, based on 3D stacking, allows each individual core to be stacked in the same package.

3D electronics

Figure 2.8 represents the rendering of a small Complementary Metal-Oxide-Semiconductor (CMOS) standard cell with three layers of metal. Even if the construction seems to be in 3D, we call this kind of circuit a 2D component, since all the transistors are on the silicon substrate plane (represented in blue in the figure). The definition of 3D integrated circuit would have been to be able to place transistors inside or on top of this metal web [52]. A solution based on a laser recrystallization process has been proposed in 1983 [64]. However, at this time, the additional costs for having several transistors layers was not justified, and this technique has never been used.

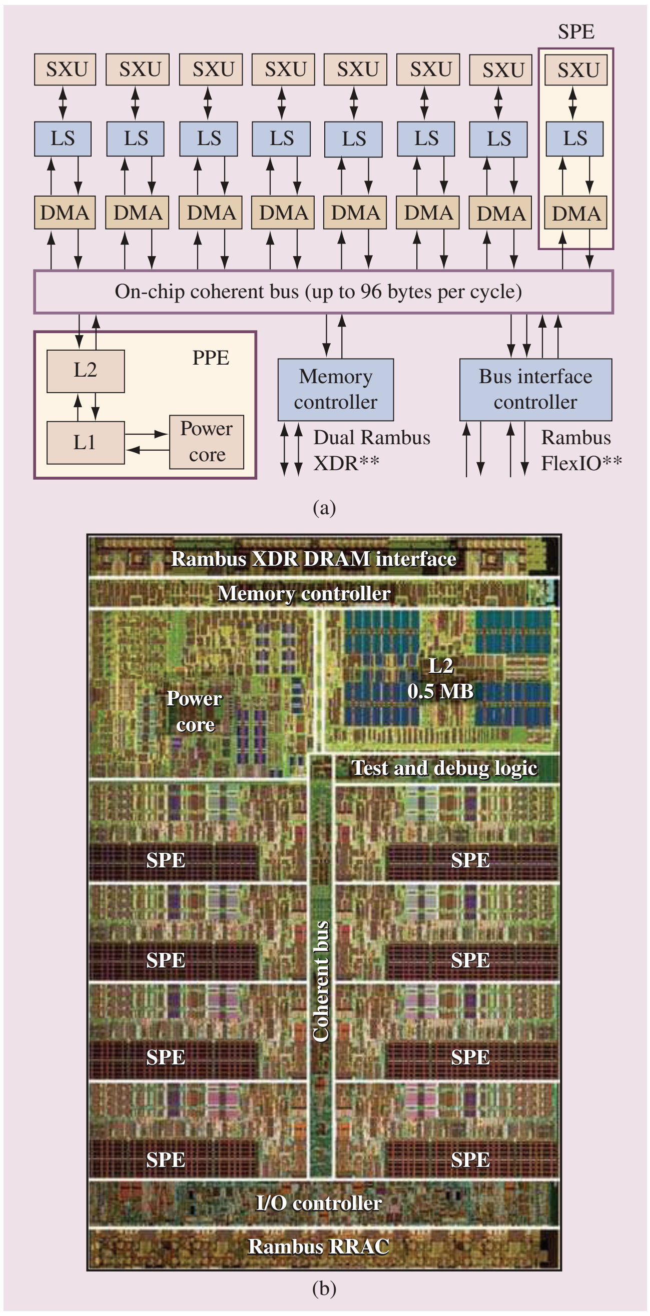


Figure 2.7: *(a)* Cell *microprocessor block diagram and (b) die photo. The first generation* Cell *processor contains a Power Processor Element (PPE) with a Power core, first- and second-level caches (L1 and L2), eight Synergistic Processor Elements (SPEs) each containing a Direct Memory Access (DMA) unit, a Local Store memory (LS) and execution units (SXUs), and memory and bus interface controllers, all interconnected by a coherent on-chip bus. Source: [61]*

Today, huge silicon surfaces are needed to meet the need for memory components with enormous capacities. The solution for having a large amount of memory within a small package has been reached through stacking several wafer layers in the same package [110]. Several low-cost methods have been developed, and are mainly used by memory chip manufacturers.

A System in Package (SiP) is a number of integrated circuits enclosed in a single package or module. The chips constituting the SiP are stacked vertically inside the package (Fig. 2.9). They are internally connected by fine wires that are buried in the package. A SiP can contain up to 50 silicon dies. Despite its benefits, this technique decreases the yield of fabrication since any defective chip in the package will result in a non-functional packaged integrated circuit, even if all other modules in that same package are functional. Other methods less expensive and simpler to implement exist, like the Stacked Package-Chip Size Package (SP-CSP) stacking where each die is connected to the others using classic bonding technology (Fig. 2.10).

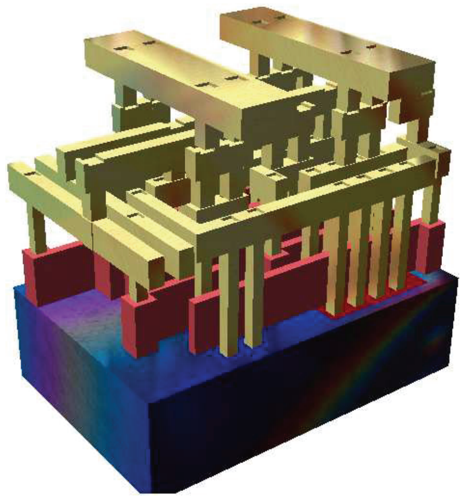


Figure 2.8: *CMOS small standard cell with three layers of metal layers. For visual purposes, the dielectric has been removed. The yellow structures are metal interconnections, with the vertical pillars being contacts. The red structures are polysilicon gates, and the blue at the bottom is the crystalline silicon bulk.*

Currently SiP are mainly used by memory manufacturers, but are also used for dedicated components used in cellphones, where telecommunication circuits lay over a microprocessor in the same package. The recent developments in multi-core microprocessors are starting to use these technique. Intel® for example includes two identical dual-core processors in the same package to produce its quad-core microprocessor.

2.2.2 Components

Nowadays, three approaches are most often used to perform a computing task. The first one consists in building a dedicated component, commonly named Application-

Figure 2.9: *Two individual ICs at the bottom-right and a stacked chip made of 19 layers of these ICs. Source: [58]*

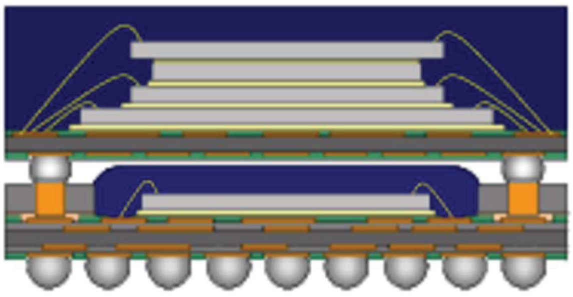


Figure 2.10: *Cross-sectional schematic of SP-CSP. Source: [62]*

Specific Integrated Circuit (ASIC), which will be suited for this specific task. The second method is to use a microprocessor, which has the advantage of being able to execute many other different tasks, but the disadvantage of not being as powerful an ASIC since its architecture allows it to processes only sequential programs. The third approach is to use a Field Programmable Gate Array (FPGA), which combines the advantages of both microprocessors and ASICs.

Table 2.1 summarizes the advantages and disadvantages of these three families, which will be described in the next paragraphs.

ASIC

An Application-Specific Integrated Circuit (ASIC) is an integrated circuit (IC) customized for a particular use, rather than for a general use. The design of an ASIC leaves for engineers the liberty to design the circuit according to their specifications. The engineer can specify the exhaustive features of the chip and optimize the functionality of the circuit, or the power consumption. It is possible to distinguish semi-custom integrated circuits, where only logic gates blocks are placed and routed, and full-custom circuits, where even the characteristics of each transistor can be specified.

Nowadays many components libraries are available for the designer, who can buy

	ASIC	Microprocessor	FPGA
Performance[a]	+	+	o
Parallelism	+	o	+
General use	-	+	+
Reconfigurability	-	+	+
Component cost[b]	+	o	-
Development costs	-	+	o
Development time	-	+	o

[a]In terms of clock speed.
[b]Indication based on high volume.

Table 2.1: *Comparison of microprocessor, ASIC and FPGA technologies. ("-" means disadvantage "0" neutral and "+" advantage).*

specific blocks, also called Intellectual Property (IP) modules. An IP can be for example a MP3 audio decoder, a memory module, an encryption system.... Since production costs are very high, particular care should be taken with verification. The whole circuit should be simulated to avoid errors, which are expensive and can cause big delays to the project. To simplify the conception, designers most often use System-On-Chip (SoC) [5] methodologies, which allow several modules to be designed in parallel, optimized, tested, and integrated together into the same circuit.

Microprocessors

The best example of an ASIC full custom design is the PC microprocessor, where the end goal is high frequency. Microprocessors are the most commonly used components, perfectly suited for computing. Their sequential programming helps to easily develop programs. The microprocessor family includes microprocessors from the well-known PC microprocessor, extremely powerful and fast, to the tiny microprocessor of your watch, which is slow and low power. MCUs and Digital Signal Processors (DSPs) are the perfect example of ICs that include one microprocessor and additional features, like memory, I/O interfaces, bus controllers, analog interfaces, display controllers in the same package...The MCU model is often chosen based on the peripherals and memory resources of the component, more than on the microprocessor features.

FPGA

A field-programmable gate array is a semiconductor device containing programmable logic components called "logic blocks", or CLB, and programmable interconnections (Fig. 2.11). Logic blocks can be programmed to perform the function of basic logic gates such as AND, XOR, or more complex combinational functions such as decoders or simple mathematical functions. In most FPGAs, the logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memories. A hierarchy of programmable connections allows logic blocks to be interconnected as needed by the system designer, somewhat like a one-chip programmable breadboard.

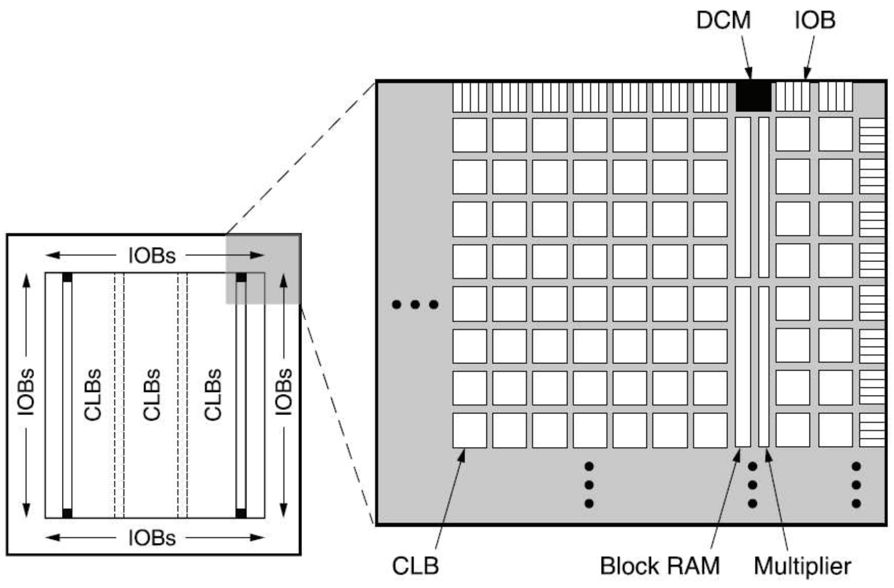


Figure 2.11: SPARTAN®*3 Family Architecture. Source: [142]*

Logic blocks and connections can be programmed by the customer or designer, after the FPGA is manufactured, to implement any logical function.

FPGAs are usually slower than their Application-Specific Integrated Circuit (ASIC) counterparts, as they handle a more complex design, and draw more power. But their advantages include a shorter time to market, ability to be re-programmed, and low development costs. The main difference between programming a standard microprocessor chip and an FPGA are into the program. Microprocessor involves the specification of a sequence of actions, or instructions, while FPGA involves a configuration of the machine itself, often at the gate level.

The FPGAs have the main advantage of being able to fulfill the same role as a MCU since they can embed microprocessor softcores in their configuration. Then, once the FPGA is configured, its microprocessor can be programmed to execute a sequential program. Advantages over MCU solution can reside in the possibility to integrate specific customs periphericals.

After the prototyping and test phases, it is still possible to migrate the developed design into a fixed version that more resembles to an ASIC [42].

2.3 The future of the computer

In 1965, a researcher named Gordon E. Moore made the empirical observation that the number of transistors on an integrated circuit doubles every 24 months: “The complexity for minimum component costs has increased at a rate of roughly a factor of two per year ... Certainly over the short term this rate can be expected to continue, if not to increase. Over the longer term, the rate of increase is a bit more uncertain, although there is no reason to believe it will not remain nearly constant for at least 10 years. That means by 1975, the number of components per integrated circuit for minimum cost will be 65,000. I believe that such a large circuit can be built on a single wafer." [82]

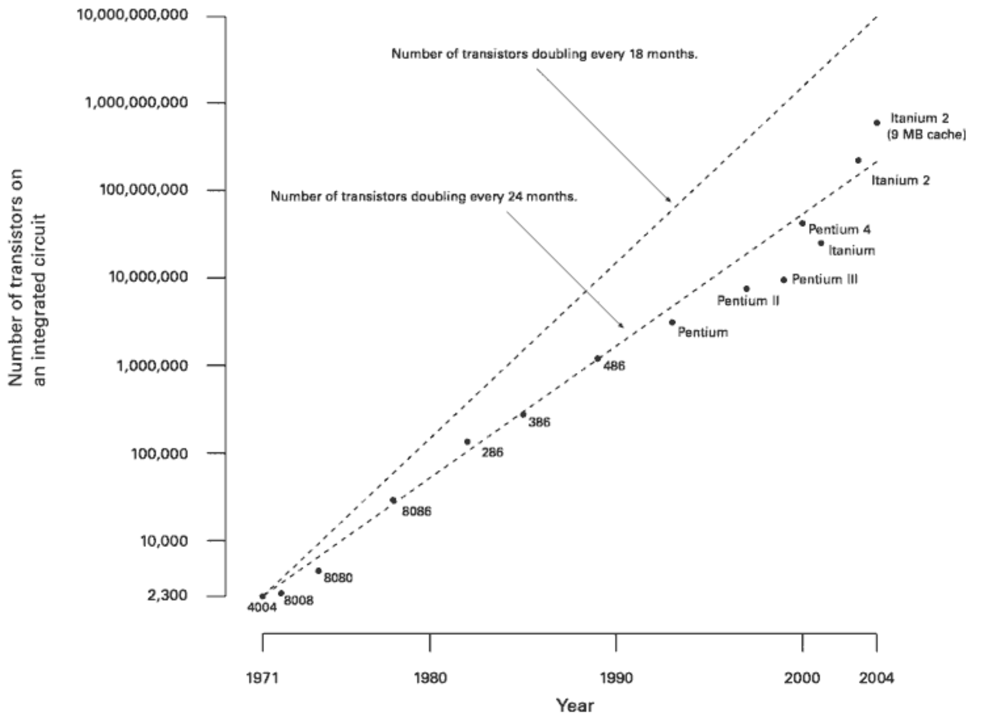


Figure 2.12: *Moore's law: growth of transistor counts for Intel® processors.*

This prediction was made a few years after the realization of the first IC; 40 years later, Moore's law is still correct [30]. Figure 2.12 shows the evolution of the number of transistors for Intel® microprocessors, with a line representing the prediction of Moore's law. Many factors are linked to the evolution of transistor number: the size decrease, the power consumption decrease, the price decrease, the frequency increase of a transistor. The observation of the evolution of all these factors also matches the prediction of Moore's law. Another factor is correlated with this law: the price of an IC foundry, which is, for a current brand new factory, about one billion US Dollars. Moore's law has been applied to several electronic domains, and states for example that RAM storage capacity increases at the same rate as microprocessor performance.

Currently, PC microprocessors are fabricated using a 65nm process. The next generation will be 45nm and will start to be commercially produced at the end of 2007 [31]. Research for building the next generation foundry, which will be 32nm, is on the way. In some years, silicon process technologies will approach the physical limits of the atomic structure. The semiconductor industry's roadmap identifies a near future where current CMOS technology will no longer be adequate, since scaling the transistor down to atomic size will become ineffective and very costly [59]. This could happen in less than a decade. Will it be the end of Moore's law? [41]

2.3.1 Nanotechnology and future architectures

Because of the need for ever better systems, and more powerful PCs with new features, it seems that the evolution of the computer will continue, at least until the singularity has been reached [68]. Predictions place this date prior to the end of our century, which mean that up to this date, new technologies need to be developed by humans. However, the CMOS silicon will reach its manufacturing limits before the ultimate computer [138] targeting the singularity is developed. Thus, new technologies such as nanotechnology[2] [103, 95, 43] will have to take over the current CMOS silicon technology. In 1959 Richard Feynman introduced this field of research in his well-known statement: "There is plenty of room at the bottom" [44]. He stated that from the point of view of physics there are not in principle any obstacles for implementing devices at the atomic level. Nowadays it is evident that nanotechnology has grown into a multidisciplinary field, drawing from chemistry, biology, computer science and mechanical engineering [112].

However, there is a lack of consensus on what type of technology and computing architecture will take over. Several research direction, all based on nanotechnology, are already on the go. Among them, the most probable break through technology successor to the current silicon technology are:

- **Nanoelectronics**, the main elements of which are constituted of nanotubes or nanowires. Nanotubes are predicted as replacements for transistors, whereas nanowires will be able to link together components at the nanometer scale.

- **Molecular electronics** aims to ultimately miniaturize logic circuits using single molecules which will act as electronic switches and storages elements [26, 135, 134].

- **Quantum computing** breaks the classic paradigm by introducing *qubits* in place of bits. Physical quantic phenomena will help solve problems exponentially faster than our classical methods [138, 24].

- **DNA computing** allows to computation to be carried out directly on molecules. All operations are performed using specific Deoxyribonucleic Acid (DNA) strands. Its main advantages reside in its massively parallel computing strength [19]. Unfortunately, the reliability of the computational process with the amount of DNA that must be supplied is not verifiable.

- **Membrane computing** is another biological field, where computations are performed inside a region defined by a membrane structure. Membranes can evolve and dissolve themselves to allow the next operations to be performed [130]. Although the formal approach to membrane computing is completely described in theory, no implementation exists yet.

[2]Nanotechnology definition (Dr. Mike Roco, National Science and Technology Council, February 2000): "research and technology development at the atomic, molecular or macromolecular levels, in the length scale of approximately 1–100 nanometer range, to provide a fundamental understanding of phenomena and materials at the nanoscale and to create and use structures, devices and systems that have novel properties and functions because of their small and/or intermediate size."

Based on current results, nanoelectronics will probably be the first breakthrough technology ready for manufacturing a future microprocessor generation. However this new revolution will not be ready for a huge production before a decade.

In parallel to these attempts to produce a mature substitute for the CMOS silicon technology, researchers are working on many other fields. Among these fields, we can mention the research for new microprocessor architectures, which may increase processor performance, using massive parallelism similar to biological cellular structure for example.

The future of the computer lies not only in increasing performance, but also the arrival of the pervasive computing, where computation capabilities are less crucial than size, or physical structure. One of the current research domains involves the realization of thousands of identical autonomous microcomputers which could be scattered in fields or included in wall paint in the aim of gathering specific information and communicating it to a user through an auto-organized network. The Smart Dust project is a perfect example of such a project, the aim of which is to build such a system in a cubic millimeter [137].

Processors are also looking to conquer new applications like the newspaper [67]. This kind of application does not require powerful system, but needs the integration of CMOS technology into new materials like flexible polymers.

The following paragraphs will be focus on two of the most promising new paradigms: nanoelectronics and bio-inspiration.

2.3.2 Nanoelectronics and nano-computer

The ultimate operating limit of the silicon transistor may be around 10 nm, or about 30 atoms long. Since it will be crucial to find alternative technologies that can further shrink computing devices, the most promising candidates are nanotubes and nanowires. Carbon nanotubes (Fig. 2.13) are cylindrical carbon molecules with novel properties that make them potentially useful in a wide variety of applications and especially in the nanoelectronics domain. A nanotube is a seamless cylinder with a diameter of around a nanometer and whose length-to-diameter ratio exceeds 10,000. Depending on its structure, a nanotube can have semiconductor properties (Fig. 2.13), or behave like a metallic element, giving it electric conductive abilities. Nanotubes

Figure 2.13: *Representation of a twisted carbon nanotube, which behaves like a semiconductor. Source: [27]*

differ from the current materials used in the electronic field in several novel properties, and especially in their capacity to carry high currents. As a comparison, the current density estimated for a nanotube is around 1 billion amps per square centimeter, whereas copper wires burn out at about 1 million $\frac{A}{cm^2}$. Researchers have succeeded in building transistor components using nanotubes [27] (Fig. 2.14), and designing a microprocessor using nanotubes could become a reality in some years [12].

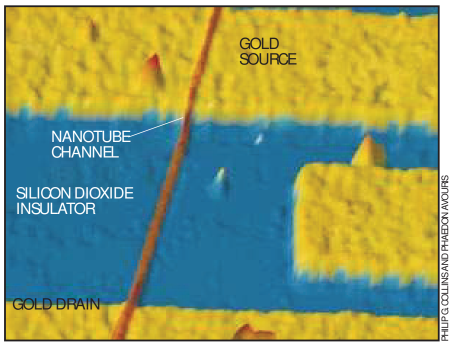

GOLD SOURCE
NANOTUBE CHANNEL
SILICON DIOXIDE INSULATOR
GOLD DRAIN
PHILIP G. COLLINS AND PHAEDON AVOURIS

Figure 2.14: *Nanotube used in an experimental field-effect transistor. Source: [27]*

For example, André Dehon has specialized his research in the development of nanoelectronic architectures toward the development of Programmable Logic Array (PLA) [36]. Figure 2.15 shows an example of a PLA structure able to realize combinational functions: the first step toward FPGAs built with carbon nanotubes and nanowires. In the nanoelectronic world, the construction of 3D structures will be much easier than with CMOS silicon technologies. Thus, creation of 3D chips could be considered once the technology is mature (Fig 2.16).

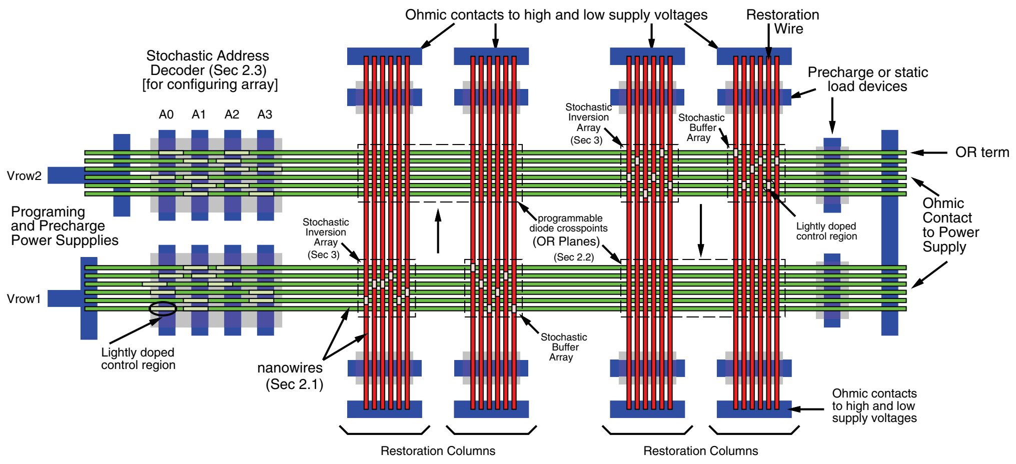
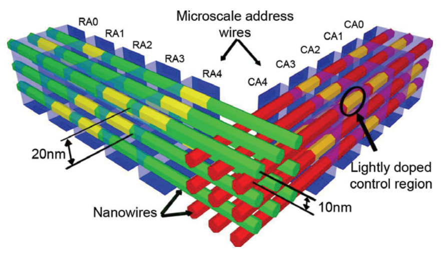

Ohmic contacts to high and low supply voltages
Restoration Wire
Stochastic Address Decoder (Sec 2.3) [for configuring array]
Precharge or static load devices
A0
A1
A2
A3
Stochastic Inversion Array (Sec 3)
Stochastic Buffer Array
OR term
Vrow2
Ohmic Contact to Power Supply
Programing and Precharge Power Suppplies
Stochastic Inversion Array (Sec 3)
programmable diode crosspoints (OR Planes) (Sec 2.2)
Lightly doped control region
Vrow1
Stochastic Buffer Array
Lightly doped control region
nanowires (Sec 2.1)
Ohmic contacts to high and low supply voltages
Restoration Columns
Restoration Columns

Figure 2.15: *Example of a nanowire PLA structure. Source: [36]*



Relative sizes and number of wires not shown to scale.

Figure 2.16: *Details of a 3D Nanowire PLA. Source: [48]*

2.3.3 Bio-inspiration

The evolution of technology is the product of human imagination. We have been able to create some really impressive and powerful systems, like the one discussed in this thesis: the microprocessor. Developing and manufacturing microprocessors is extremely expensive. The cost for building and equipping a cleanroom is currently near to a billion US dollars. Moreover, the latest PC microprocessors are certainly very powerful, but they need a lot of power, and thus need to be cooled.

On the other hand, Nature has found incredibly good solutions for building extremely small and complex living systems, moreover at an incredibly low cost and low power consumption. The human brain, for example, is able to perform extremely complex operations like object recognition. The biggest computers, the size of which is many times greater than a human brain, are not yet able to perform this task, and are limited to the recognition of some pre-learned objects.

Rather than trying to improve our current technologies by miniaturizing the transistor size, it could be interesting to look at how Nature has created the living world, and thus try to find some ideas that might be used for designing or manufacturing microprocessors. If we look closer, a living creature is composed of cells, the number of which may vary from one for a bacterium, to hundred trillion for a human being. Each cell has in common the construction map of the whole organism, the DNA. The human cell, sizing about $10\mu m$, is able to self-replicate in a few minutes, with its 3 billion DNA pairs perfectly copied. The most spectacular aspect of this process may be that this transformation requires little energy and occurs at normal temperature and pressure (unlike the IC process which needs strong acids, high temperatures and pressures and a dust free atmosphere).

It is thus very interesting to copy Nature in our research. The aim is not to use natural processes and add them to our microprocessor design, but to try to take inspiration from them and find applications where they could be used. Bio-inspiration is not a new concept and has already been introduced into computer science [78], but several research fields could still benefit from this approach to find new developments.

The POE model

Bio-inspiration encompasses a lot of different areas. To help researchers to define their bio-inspiration domain, a classification model has been proposed, the POE model [119, 111]. If we consider life on Earth since its very beginning, then the following three levels of organization can be distinguished [119, 111]:

- **Phylogeny:** The first level concerns the temporal evolution of the genetic program, the hallmark of which is the evolution of species, or phylogeny. The multiplication of living organisms is based upon the reproduction of the program, subject to an extremely low error rate at the individual level, so as to ensure that the identity of the offspring remains practically unchanged. Mutation (asexual reproduction) or mutation along with recombination (sexual reproduction) give rise to the emergence of new organisms. The phylogenetic mechanisms are fundamentally non-deterministic, with the mutation and recombination rate providing a major source of diversity. This diversity is vital for the survival of living species, for their continuous adaptation to a changing environment, and for the appearance of new species.

- **Ontogeny:** Upon the appearance of multicellular organisms, a second level of biological organization manifests itself. The successive divisions of the mother cell, the zygote, with each newly formed cell possessing a copy of the original genome, is followed by a specialization of the daughter cells in accordance with their surroundings, i.e., their position within the ensemble. This latter phase is known as cellular differentiation. Ontogeny is thus the developmental process of a multicellular organism. This process is essentially deterministic: an error in a single base within the genome can provoke an ontogenetic sequence which results in notable, possibly lethal, malformations.

- **Epigenesis:** The ontogenetic program is limited in the amount of information that can be stored, thereby rendering the complete specification of the organism impossible. A well-known example is that of the human brain with some 10^{10} neurones and 10^{14} connections: far too large a number to be completely specified in the four-character genome of length approximately 3×10^9. Therefore, upon reaching a certain level of complexity, there must emerge a different process that permits the individual to integrate the vast quantity of interactions with the outside world. This process is known as epigenesis, and primarily includes the nervous system, the immune system, and the endocrine system. These systems are characterized by the possession of a basic structure that is entirely defined by the genome (the innate part), which is then subjected to modification through lifetime interactions of the individual with the environment (the acquired part). The epigenetic processes can be loosely grouped under the heading of *learning* systems.

In analogy to nature, the space of bio-inspired hardware systems can be partitioned along these three axes: *phylogeny*, *ontogeny*, and *epigenesis*, giving rise to the POE model. Combinations of these components over these three axes give new possibilities for research fields (Fig. 2.17).

The following list gives the computer applications and implementations of these three axis and their combinations:

- **P**: If in life, *phylogeny* principally concerns the genetic evolution of species,

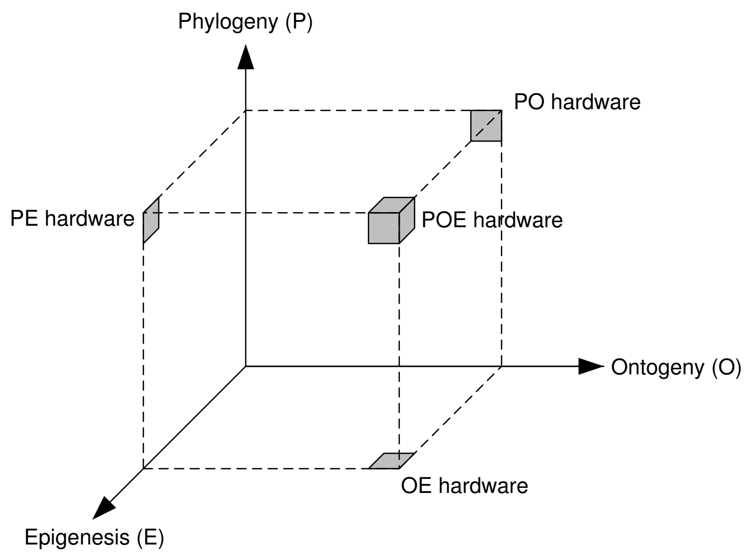


Figure 2.17: *The tri-dimensional representation of the POE model*

then in the engineering world this corresponds mainly to the panel of evolutionary algorithms, genetic algorithms and evolvable hardware [55, 112].

- **O**: The *ontogenetic* axis involves the development of a single individual from its own genetic material, essentially without environmental interactions. It involves multi-cellular organization, cellular division and differentiation from the mother cell to the daughter cells. Each daughter cell possesses a copy of the original genome. Our thesis research will be mainly based on this axis of inspiration.
- **E**: The *epigenetic* axis involves learning through environmental interactions that take place after formation of the individual. Artificial neural networks are the most common illustration of *epigenesis* [63].
- **PO**: The combination of the *phylogeny* and *ontogeny* axes exhibits the evolution of cellular structures, and the creation of self-reproducing and self-healing evolvable hardware. The Morphogenetic Evolutionary System is an example of such a PO plane [108].
- **PE**: An example of the PE plane is evolutionary artificial neural networks, where evolution allows adaptation of the network structure, the synaptic weights and/or learning rules [144].
- **OE**: This plane combines ontogenetic mechanisms (self-replication, self-repair) with epigenetic learning (neural network). The combination gives self-reproducing and self-healing learning hardware systems.
- **POE**: This space comprises systems that exhibit characteristics pertaining to all three axes, like a hardware system that exhibits learning, evolutionary diversity and a multi-cellular array [111, 120]. The POETIC project concluded the realization of IC able to support these three axes [84].

From a technological point of view we observed that many bio-inspired realizations are based on programmable circuits like FPGAs.

Of these three bio-inspired research axes, we will focus on *ontogeny* in the next paragraph, with the presentation of an original bio-inspired research project developed by the Logic Systems Laboratory (LSL) team: the Embryonics project.

The Embryonics project

In the previous paragraph we mentioned the powerful characteristics of the human being, whose hundred trillion of cells execute their genome, divide and replicate ceaselessly from conception to the death of the individual. Faults are rare and, in the majority of cases, successfully detected and repaired. This remarkable process relies on the structure of the DNA for its complexity and its precision. The Embryonics[3] project [75, 76, 78] is inspired by the basic processes of molecular biology and by the embryonic development of living beings. By adopting certain features of cellular organization, and by transposing them to the two-dimensional world of integrated circuits on silicon, the Embryonics project shows that properties unique to the living world, such as self-replication and self-repair, can also be applied to Integrated Circuits (ICs), and proposes a novel robust architecture to fulfill these objectives.

Almost every living being, with the notable exceptions of viruses and bacteria, share the same basic principles for their organization. The incredible complexity present in organisms is based on multi-cellular organization where cells having a limited function are able to behave on very complex ways by assembling themselves into specific structures and operating in parallel.

Each cell can be considered as universal since each contains the whole of the organism's genetic material, the genome. The Embryonics project is based on the same structure and features the three fundamental features of an organism:

1. **Multicellular organization** divides the organism into a finite number of cells, each charged with a unique function (neuron, muscle, intestine,...). The same organism can contain multiple cells of the same kind.

2. **Cellular division** is the process whereby each cell (beginning with the first cell or zygote) generates one or two daughter cells. During this division, all of the genetic material of the mother cell, the genome, is copied into the daughter cell(s).

3. **Cellular differentiation** defines the role of each cell of the organism, that is, its particular function (neuron, muscle, intestine,...). This specialization of the cell is obtained through the expression of part of the genome, consisting of one or more genes, and depends essentially on the physical position of the cell in the organism

When a minor (wound) or major (loss of an organ) trauma occurs, living organisms are thus potentially capable of self-repair (i.e. cicatrization) or self-replication (i.e. cloning or budding). The main goal of the Embryonics project is the implementation of the above three features of living organisms in an integrated circuit in silicon, in order to obtain the properties of self-repair and self-replication.

[3]For embryonic electronics

Multicellular organization The Embryonics project is based on a regular two dimensional grid, similar to the silicon process possibility, but different from the living world where cells are organized on a three dimensional structure.

The artificial organism is divided by the multicellular organization into a finite number of cells; each cell realizes a unique function, defined by a subprogram called the gene of the cell, which is part of the genome (Fig. 2.18). The same organism can contain multiple cells expressing the same gene. However, the state and values computed by each cell are independent from the other cells.

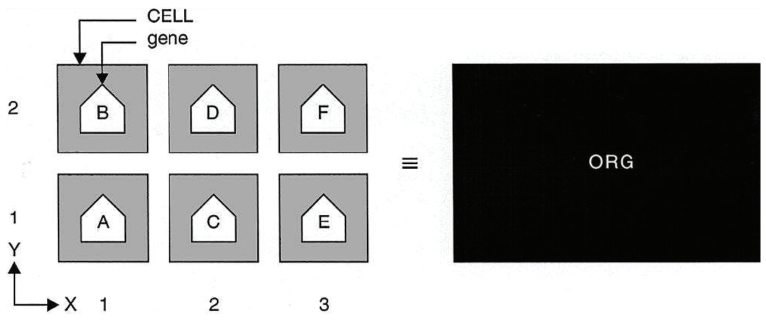


Figure 2.18: *Multicellular organization of a 6-cell organism expressing 6 different genes.*

Cellular differentiation The operative genome (OG) is the program containing all the genes of an artificial organism, where each gene (A to F) is a subprogram characterized by a set of instructions and by the cell's position (coordinates X,Y=1,1 to X,Y=3,2).

When each cell contains the entire operative genome (Fig. 2.19), depending on its position in the array, i.e., its place within the organism, each cell can interpret the operative genome and extract then execute the gene which defines its function.

In summary, storing the whole operative genome in each cell makes the cell universal: it can execute any one of the genes of the operative genome and thus implement *cellular differentiation.*

In our artificial organism, any cell CELL[X,Y] continuously computes its coordinate X by incrementing the coordinate WX of its neighbor immediately to the west. Likewise, it continuously computes its coordinate Y by incrementing the coordinate SX of its neighbor immediately to the south. At startup, the first cell, at the bottom left of the organism, is arbitrarily defined as having the coordinates X,Y=1,1, and holds the one and only copy of the operative genome OG. After time t1, the genome of the zygote (*mother* cell) is copied into the neighboring (*daughter*) cells to the east (CELL[2,1]) and to the north (CELL[1,2]). This process of cellular division continues until the six cells of the organism are completely programmed (in our example, the farthest cell is programmed after time t3).

The self-replication of organisms The *self-replication* or cloning of the organism, i.e., the production of an exact copy of the original, can be performed if empty cells exist on the silicon circuit (at least six in our organism example) and if the calculation

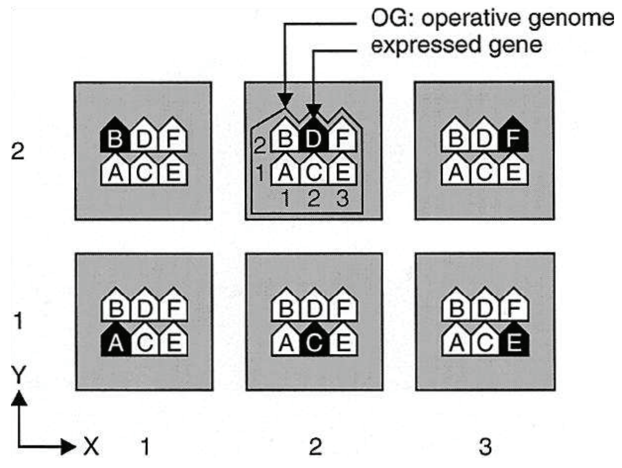


Figure 2.19: *Cellular differentiation of the organism with the operative genome and its expressed gene depending on the coordinates.*

of the coordinates produces a cycle $X = 1 \to 2 \to 3 \to 1 \ldots$ and $Y = 1 \to 2 \to 1 \ldots$ in figure 2.20, implying $X = (WX + 1)$ modulo 3 and $Y = (SY + 1)$ modulo 2.

When both conditions are present, a copy of the *mother* organism will be produced and one or many *daughter* organisms will populate the IC silicon surface.

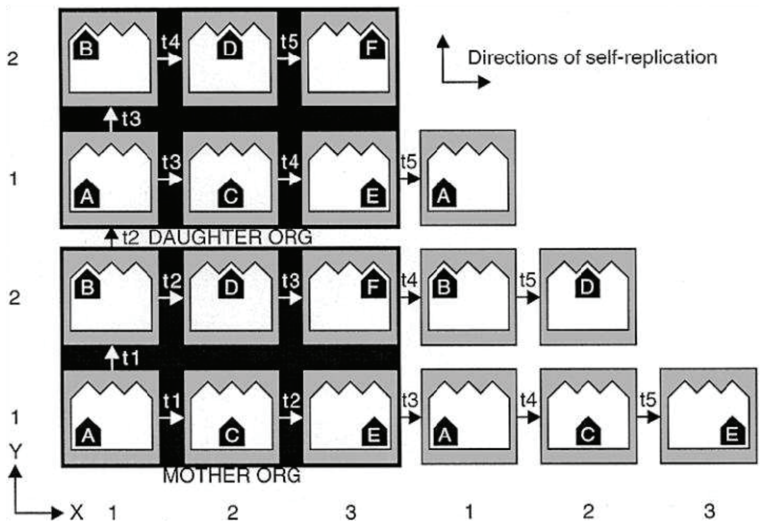


Figure 2.20: *Self-replication of a six-cell organism in a limited homogeneous array of 6×4 cells (situation after 5 cellular divisions).*

The self-repair of organisms In order to implement the *self-repair* of the organism, two columns of spare cells have been added to the right of the original organism (Fig. 2.21). The existence of a fault is detected by a KILL signal, which is produced in each cell by a built-in self-test mechanism. The state KILL=1 identifies the faulty cell, and the entire column to which the faulty cell belongs is considered faulty and is deactivated (column X=2 in figure. 2.21). All the functions (X coordinate and gene) of the cells at the right of the column X=1 are shifted by one column to the right. Obviously, this process requires as many spare columns to the right of the array as there are faulty cells or columns to repair (two spare columns, tolerating two successive faulty cells, in the example of figure. 2.21). It also implies that the cell needs to be able to bypass the faulty column and to divert to the right all the required signals (such as the operative genome and the coordinate, as well as the data buses).

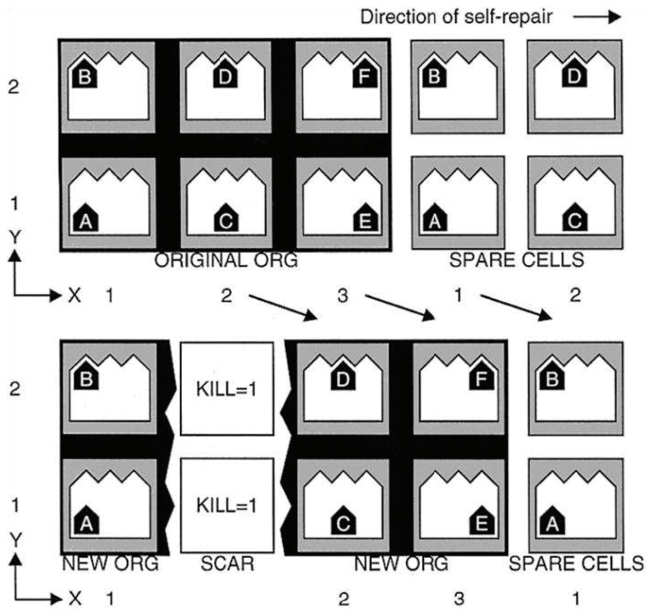


Figure 2.21: *Self-repair of a 6-cell organism with four spare cells and a faulty column.*

The Embryonics Landscape In each cell of every living being, the genome is translated sequentially by a chemical processor, the *ribosome*, to create the proteins needed for the organism's survival. The ribosome itself consists of molecules, whose description is an important part of the genome.

In the Embryonics project each cell is a small processor that sequentially executes the instructions of the artificial genome, the operative genome OG.

The need to realize organisms of varying degrees of complexity has led us to design an artificial cell characterized by a flexible architecture, that is itself configurable. The Embryonics project could therefore be implemented using previously described FPGA components. Each element of this FPGA (consisting essentially of a buffer and a multiplexer associated with a programmable connection network) is then equivalent to a molecule, and an appropriate number of these artificial molecules allows the creation of application specific processors, i.e. cells.

As a result of this new definition, the final architecture of the Embryonics project is based on the following four hierarchical levels of organization, described from the bottom up (Fig. 2.22):

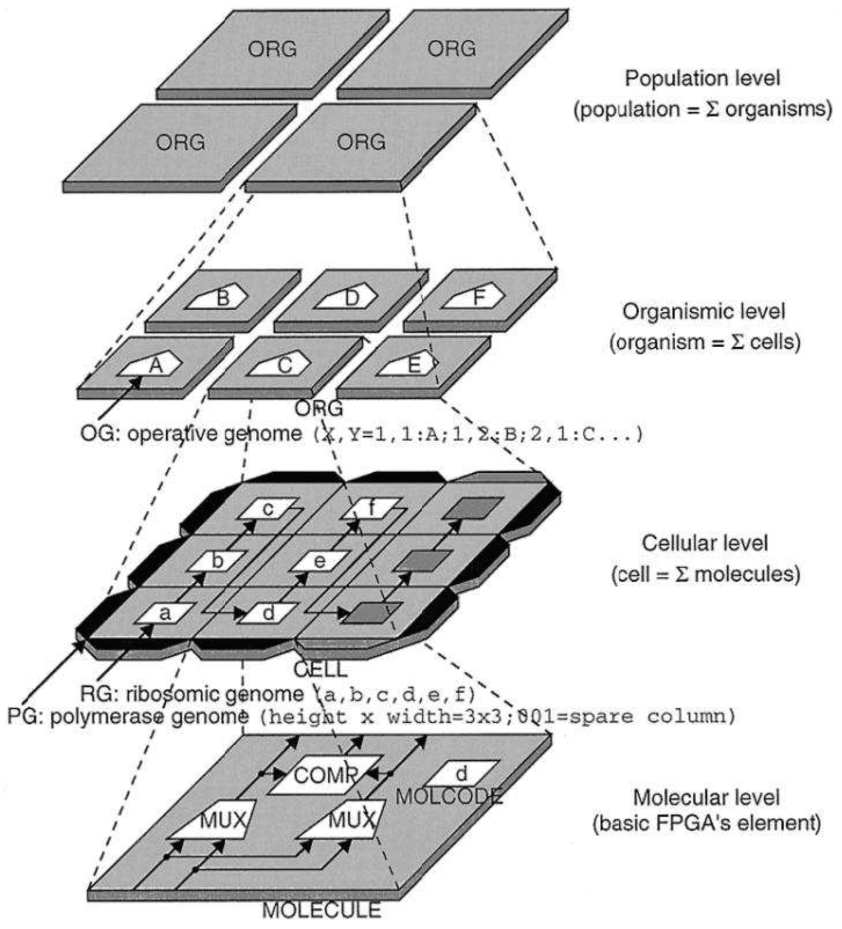


Figure 2.22: *The Embryonics landscape: a four level hierarchy.*

- The basic primitive of the system is the molecule, the FPGA element, consisting essentially of a buffer and a multiplexer associated with a programmable connection network. The multiplexer is duplicated to allow the detection of faults. The logic function of each molecule is defined by its molecular code or MOLCODE.
- A cell is made up of a finite set of molecules: essentially a processor with the associated memory. In a first programming step of the FPGA, the *polymerase genome* PG defines the topology of the cell, that is, its width, height, and the presence and positions of columns of spare molecules. In a second step, the *ribosomic genome* RG defines the logic function of each molecule by assigning its molecular code or MOLCODE.
- An *organism*, an application specific multiprocessor system is made up of a finite set of cells. In a third and last programming step, the *operative genome* OG is copied into the memory of each cell to define the particular application executed by the organism, for example the electronic watch, the random number generator,. . .

- The organism can itself self-replicate, giving rise to a *population* of identical organisms, the highest level of our hierarchy.

Since fault-tolerance is a recurrent topic in the Very-Large-Scale Integration (VLSI) industrial world [70, 29], the Embryonics project provided an illustration of the use of bio-inspiration to solve a specific problem. Even if the results of this project are not currently used in the industry as, for example, inside FPGA's architectures, the Embryonics project is a perfect example of an ontogenetic hardware [117] whose results can stimulate new kinds of research and future applications [129, 22, 77].

2.4 Towards cellular machines

In the space of a few decades, computers have produced a revolution in the industrial domain. From a small number of machines and only few researchers in the computer domain in the 1960's, computing has developed to the extent that millions of people now own a PC and several microprocessors embedded in everyday tools like watches, music players, automobiles... At the same time, the number of researchers around the world working in the computer sciences area has to be one of the most rapidly expanding communities. In a few decades, the CMOS silicon technology has been invented and has practically reached its apogee. Substitutes, like nanoelectronic components, or the quantum computer will soon be a reality.

In parallel to the research concerning these successors to silicon technology, a lot of work is being done on new kinds of architectures suited for microprocessors. One of the next promising source of inspiration is the living world and the human itself [130, 3].

We mainly focused in this chapter on bio-inspiration and the description of an ontogenetic architecture: the Embryonics project. This architecture, in addition to its intrinsic purposes which are self-replication and self-repair, showed us the use of a massively parallel and regular structure based on reconfigurable components like FPGAs, which is to say a cellular architecture.

The cellular architecture constitutes a relatively recent paradigm in parallel computing, and is becoming to be used in commercial products. Pushing the limits of multi-core architectures to their logical conclusion where the programmer has the possibility to run many threads concurrently, this architecture is based on the replication of many similar computing elements. Containing all the memory, computational power and communication capabilities, each cell provides a complete environment for running a whole thread. Thus, the problem of achieving greater performance becomes a matter of how many cells can be put in parallel and, by extension, of how well thread-level parallelism can be extracted from the application.

For our thesis research, it seemed interesting to investigate the concept of cellular architectures. Rather than focusing on networks made of a limited number of extremely powerful microprocessors, we chose to create extremely large cellular networks with rather simple reconfigurable components: FPGAs.

Our first realization will implement the Embryonics project presented in this chapter into hardware. This massively parallel machine, made of thousands of FPGAs or molecules, is the BioWall.

Chapter 3

BioWall hardware description

THE BioWall is the first realization that employs the Embryonics concepts. This bio-inspired electronic tissue was firstly developed to study and verify in hardware the three essential biological models presented in § 2.3.3:

- Phylogeny - the history of the evolution of species.
- Ontogeny - the development of an individual directed by genetic code.
- Epigenesis - the development of an individual through learning processes (nervous system, immune system), influenced both by genetic code (the innate) and environment (the acquired).

In the first section, we describe the hierarchical structure of our cellular system. Section 3.2 presents all the electronic components that make up the BioWall. Before concluding this chapter, we give a closer look at the software running this machine.

3.1 Specifications and global overview

The implementation of the Embryonics model led us to imagine a machine with the following attributes: a basic computing element with limited complexity, bio-inspired features like growth and cicatrization, homogeneity, scalability, interactivity. Moreover, to tally with the ontogenetic model, we built our system using a large amount of identical computing units (like for an organism which is composed of many cells), where each unit computes its genome in parallel.

We named our machine BioWall due to its biological inspiration, as well as its size (5.3m x 0.8m x 0.5m). Thanks to a grant from the Villa Reuge foundation, we built the BioWall and were able to demonstrate the features of the Embryonics project to the public through visual and tactile interaction (Fig. 3.1).

As the BioWall should be an easily reprogrammable flexible tool, we decided to base our design on reconfigurable components, notably Field Programmable Gate Array (FPGA) circuits. The first planned application, the *BioWatch*[124], involved 4'000 independent basic units (artificial molecules). Each artificial molecule is implemented within an FPGA. Thus we built the BioWall on an array of 4'000 FPGAs.

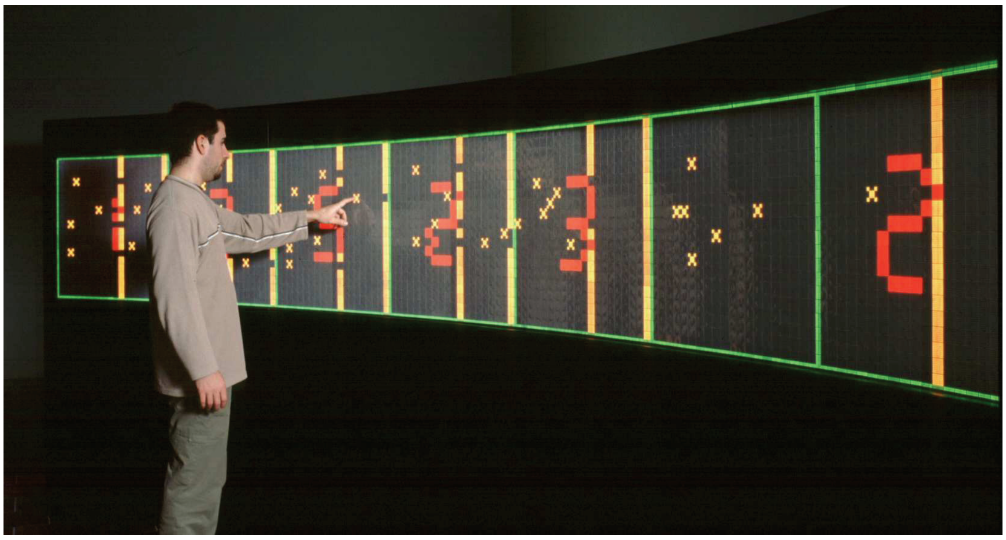


Figure 3.1: *Photograph of the BioWall running the* BioWatch *application.*

3.1.1 BioWall specifications

To fulfil with the previous mentioned specifications, we designed the BioWall as a reconfigurable (1) machine built on a mesh-topology (2), able to continuously display the state of each molecule (3) and to interact with multiple users (4):

1. Since the BioWall machine is made of a large array of FPGAs, it was important to choose a moderately priced component. We chose the XCS10XL SPARTAN® from Xilinx®, which was a new product at the start of the project and especially attractive for its low price. The main advantages of this FPGA are that there is no limit to the number of times it can be reconfigured, it has enough logic gates to match our needs and it is very simple to use. Moreover this FPGA is really robust and supports hazardous Input/Output (I/O) shortcuts due to development faults.

2. The BioWall is built on a mesh-network where each FPGA is connected with a von Neumann neighborhood to its four adjacent FPGAs using about 25 direct lines. The 4'000 FPGAs are globally synchronized using a common clock at a fixed frequency of 1.0MHz. Operation control is possible thanks to global lines under supervision of external software running on a Personal Computer (PC).

3. Each FPGA can access a dedicated screen consisting of an 8 by 8 two color Light-Emitting Diode (LED) dot matrix.

4. A transparent touchsensor directly connected to the SPARTAN® reconfigurable circuit is glued onto the top of the display module. This 3.2×3.2 cm membrane indicates if the surface is activated or disactivated.

3.1.2 BioWall overview

The BioWall is a two-dimensional tissue consisting of 4'000 identical basic elements (also named molecules), as shown in figure 3.2. Each molecule is made of:

- An input element (a touch-sensitive membrane).
- An output element (an array of 64 two-color LEDs).
- A programmable computing element (a Xilinx® SPARTAN® XCS10XL FPGA).

Our machine is constructed using a hierarchical structure, in which an external computer manages all the BioWall functions through the *BioBox*, which is the wall's main controller. 160 *BioStack* boards, each made of $5 \times 5 = 25$ molecules, are plugged into a hard iron structure. Each *BioStack* board is made up of a *BioLogic* and a *BioDisplay* boards rigidly bound together.

The electronic components are mounted on these two boards. The logic board (*BioLogic*) hosts 25 FPGAs, and the display board (*BioDisplay*) holds 25 displays and their membranes. The boards are connected together by a flat wire to allow two-way communications between the logic and the display (a dedicated circuit on the logic board automatically distributes the signals to the display).

On the logic board, the SPARTAN® devices are placed in a regular two-dimensional grid. Each FPGA is connected to its four cardinal neighbors using approximately 25 pins per link. The pins of the FPGAs placed along the edges of the board are wired to a set of connectors which allow pin-to-pin connection across boards, thus creating perfectly uniform surfaces of FPGAs spanning as many boards as required.

The remaining pins are connected to a centralized circuit that handles the distribution of the global signals (the clocks, resets, and FPGA configuration streams) arriving from the *BioBox*.

The BioWall is composed of 160 *BioStack*s for a total of 4'000 molecules. Thanks to their architecture, the boards can be seamlessly connected with each other to form a homogenous surface of any shape and size.

This tissue of 4'000 FPGAs represents an impressive amount of computational power, coupled with I/O interfaces (comprising the membranes, the LED arrays and I/O pins around the BioWall edges) that allow large-scale visual and tactile interactions. The advantage of this solution is the size of the display, which enables an immediate interaction with the applications, normally limited to software simulation on a computer screen. Furthermore, the computing power and programmability of the Xilinx® FPGAs enable the easy prototyping of new bio-inspired applications.

All biological organisms have in common the same kind of genetic information, the Deoxyribonucleic Acid (DNA), contained in each cell. The BioWall contains similar information, as each molecule gets the same configuration.

To cope with the mechanical problems caused by the big size of the BioWall, we divided the machine into several identical structures and board assemblages.

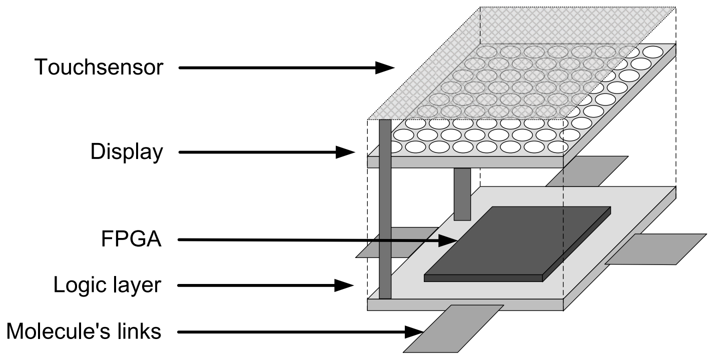
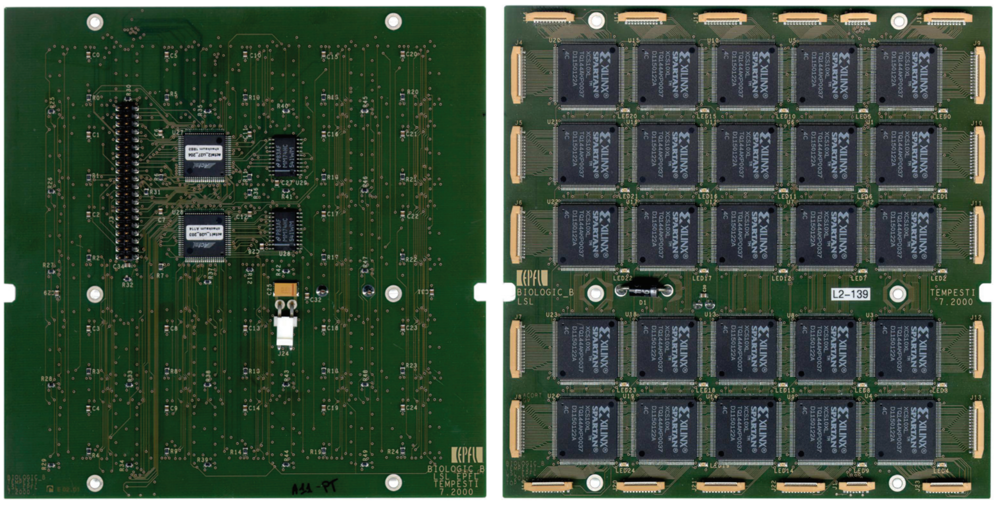


Figure 3.2: *BioWall molecule structure.*

3.2 BioWall board description

3.2.1 The *BioLogic* board



(a) Top view: connector to the *BioDisplay* board, 3.3V input connector, Actel® FPGAs controlling all the SPARTAN® FPGAs located on the bottom side.

(b) Bottom view: the 5 x 5 SPARTAN® FPGA array with connectors to next *BioLogic* boards.

Figure 3.3: *Pictures of the* BioLogic *board.*

The *BioLogic* board (Fig. 3.3) constitutes the main module of the BioWall platform and houses all the computational components within which the applications will run.

This board is articulated around a grid of 25 SPARTAN® FPGAs[1] from Xilinx® and some other components, whose tasks are dedicated to controlling the FPGA array (Fig. 3.4). All these components are soldered on a big (160 × 160 mm) 6-layer Printed Circuit Board (PCB).

[1]The exact model is the XCS10XL-4TQ144C [141].

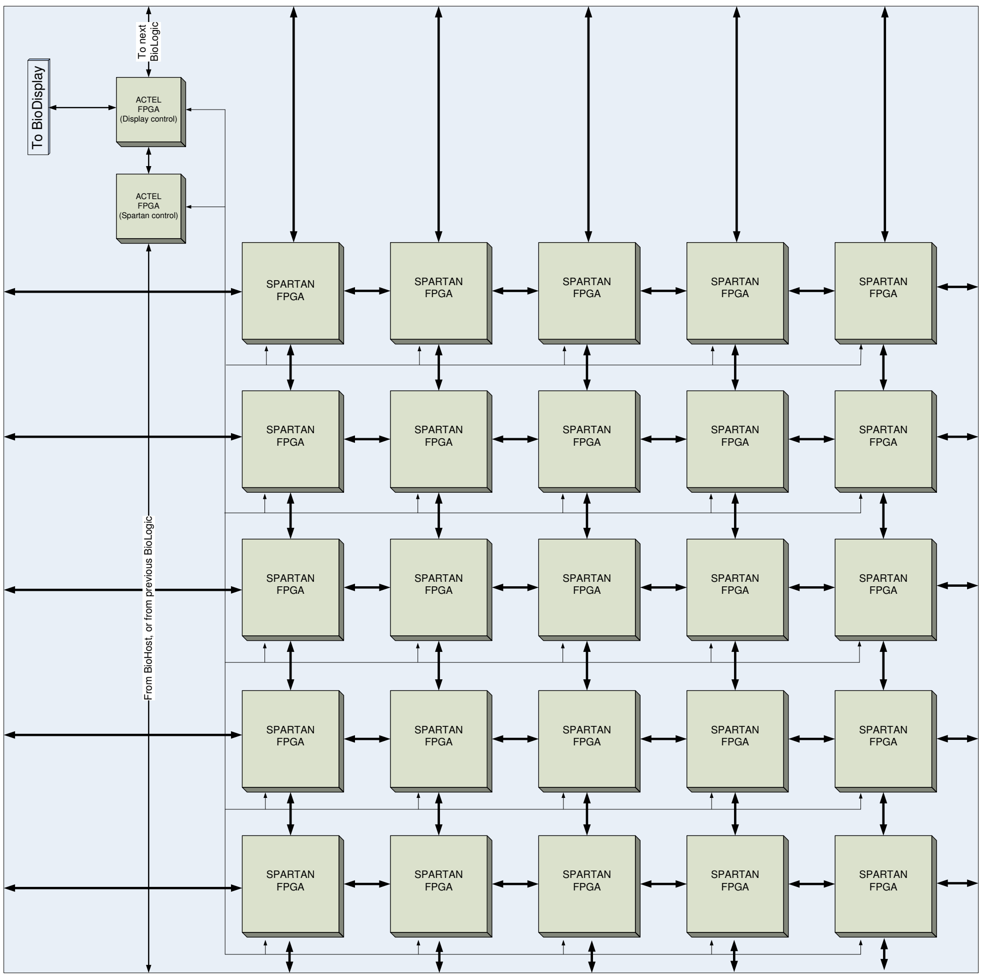

To next BioLogic
To BioDisplay
ACTEL FPGA (Display control)
ACTEL FPGA (Spartan control)
SPARTAN FPGA
From BioHost, or from previous BioLogic

Figure 3.4: *Architecture of the* BioLogic *board.*

The main features of the *BioLogic* board are as follows: (1) computational FPGAs, (2) configuration and control, and (3) *BioDisplay* interface:

1. Equivalent to $10'000$ logic gates, this FPGA articulated on a simple Configurable Logic Block (CLB) structure, each made of three Lookup Tables (LUTs) associated with two flip-flops, is really easy to use. It has direct links with its four direct neighbors and connections, through a dedicated controller FPGA, with the LED display and touchsensor located at the top. Based on Static Random Access Memory (SRAM) technology, an unlimited number of reconfigurations is possible.

2. Configuration and control of all FPGAs are performed by two dedicated Actel® FPGAs located on the top side of the PCB. The first Actel® FPGA receives configuration streams, reset, and clock signals from the *BioBox* and distributes them to the 25 SPARTAN® FPGAs.

3. The second Actel® FPGA provides an interface between all the Xilinx® FPGAs and the *BioLogic* board. It receives the serial encoded stream with the pixel values from each SPARTAN® circuit, transmits it to the *BioDisplay* board and manages the display refresh cycle. Another task assigned to this component is to scan the touchsensor surface and distribute the corresponding sensor status to the matching SPARTAN® FPGA. All communications between the *BioLogic* and the *BioDisplay* boards go through 40-pin connectors.

Each SPARTAN® FPGA possesses various connections with its neighbor FPGAs and the other components of the system. White connectors surrounding the board and visible on figure 3.3(b) allow to directly connect adjacent FPGAs of two sided *BioLogic* boards. Signals and communication possibilities available for each SPARTAN® components are described in table 3.1.

Type	Line number	Description
Horizontal links	27	CMOS signaling connecting adjacent FPGAs in the horizontal direction. These lines are bidirectional and functionality is specified inside an application.
Vertical links	21	CMOS signaling connecting adjacent FPGAs in the vertical direction. These lines are bidirectional and functionality is specified inside an application.
Clock	1	1.0 MHz clock signal coming from the *BioBox*.
Reset	1	FPGA reset signal generated by the *BioBox* (active high).
X_EN	1	Enable signal generated by the *BioBox* which frequency can vary from a step-by-step mode to a 200 khz frequency.
F_EN	1	Second enable signal generated by the *BioBox* which frequency can vary from a step-by-step mode to 200 khz.
Touchsensor	1	State of the tactile sensor; low level when pressed, high logic level otherwise.
Display	5	These lines must be connected to a specific Intellectual Property (IP) core; they provide the interface between the FPGA to the LED display.

Table 3.1: BioLogic *FPGA I/O summary.*

In the current BioWall version, the SPARTAN® FPGAs can only be programmed with the same configuration, which limits the functionality of the units to the $10'000$ logic gates of the SPARTAN® circuit, while the considerable delays inherent in propagating a global signal over distances measured in meters limit the clock speed to a low frequency. We set this clock limit to the frequency of 1.0 MHz which is more adequate if coupled with the massive parallelism of the system and considerably too fast for human interaction in many applications. In addition to the fixed frequency clock signal, two configurable enable signals make it possible to slow down the computation speed in order to have time to show the computational results on the BioWall screen. The first enable signal labeled X_EN can be controlled from the 1 Hz frequency up to 200 khz. A specific mode allows it to generate pulses uniquely on request. Frequency speed as step-by-step mode are controlled from a software running on a PC. The second signal, F_EN, has the same control possibilities as X_EN, but should be slower or equal to the frequency of X_EN. Figure 3.5 exhibits these signals and the way in which they are correlated together. The F_EN signal will always be in phase with the X_EN signal, and can never be present without X_EN being high. For most applications we only use the X_EN signal; it was for the *BioWatch* application that these two enable signals were needed.

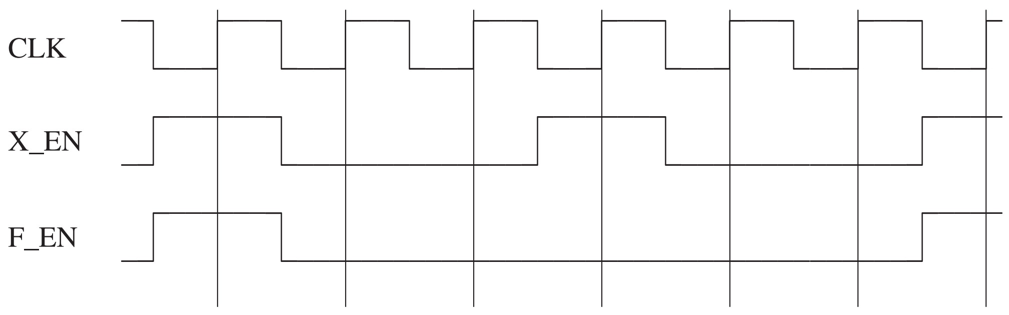


Figure 3.5: *BioWall clock and enable timings.*

3.2.2 The *BioDisplay* board

The density of components on the *BioLogic* board does not allow integration of the LED display modules on the same substrate. We solved this problem by building another board, named *BioDisplay*. This board, the size of which is 165×165 mm, is made of a grid of 5×5 small display modules each constituted by a 8×8 red and green LED matrix (Fig. 3.6). On the upper face of each of these modules is glued a touchsensor, which gives a logical 0 signal when its surface is pressed, and a logical 1 signal otherwise. In the interest of simplifying and limiting the number of transistors driving the LED display, we decided to define a pixel as a group of two by two LEDs. Thus each FPGA of the *BioLogic* board drives its corresponding display (Fig. 3.7) and can assign to the $4 \times 4 = 16$ pixels four different colors which are: black, red, green and orange. Table 3.2 gives the description of the interface of the IP core that needs to be used by each application and which will reside inside the SPARTAN® FPGA.

The assemblage of the whole BioWall has been taken in consideration in the conception of the entire system. This results in the fact that a single connector should connect the *BioDisplay* board to the *BioLogic* board. Thus, all display signals are multiplexed together with the touchsensors data; they are gathered on a connector located on the bottom side of the PCB.

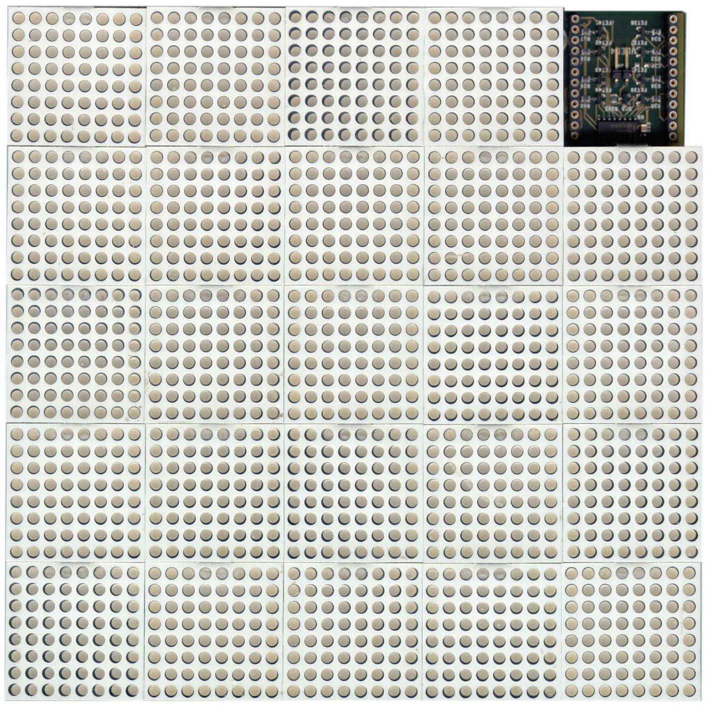


Figure 3.6: *Picture of the* BioDisplay *board with the top-right display module removed.*

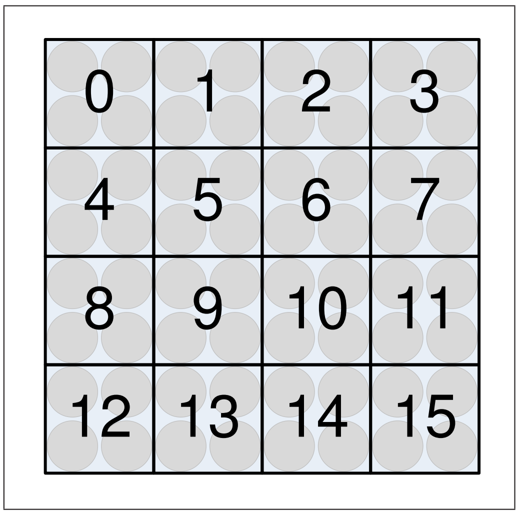


Figure 3.7: *BioWall pixel ordering.*

3.2.3 The *BioStack*

The previous paragraphs have described the *BioLogic* and the *BioDisplay* boards. These two main elements of the BioWall machine are bolted together to build a new module: the *BioStack*.

The *BioDisplay* board, which is placed on top of the stack, is easily recognizable

Name	Bus size	Direction	Description
red	16	IN	red signal; one bit for each pixel.
green	16	IN	green signal; one bit for each pixel.
DISP_R_CCK	1	IN	signals used by this core to send display data with the Actel® FPGA.
DISP_RST	1	IN	
DISP_EN	1	IN	
DISP_IN	1	IN	
DISP_OUT	1	OUT	

Table 3.2: *BioWall display interface.*

in figure 3.8, with its LED display and touchsensor surface. The flexible plastic wires between the *BioDisplay* PCB and the top of the screen constitute the connection for the touchsensors.

Under this board is fixed the *BioLogic* card, which handles all the computational logic. Both boards are connected with a 40-pin connector, which transmits all the display and sensor data.

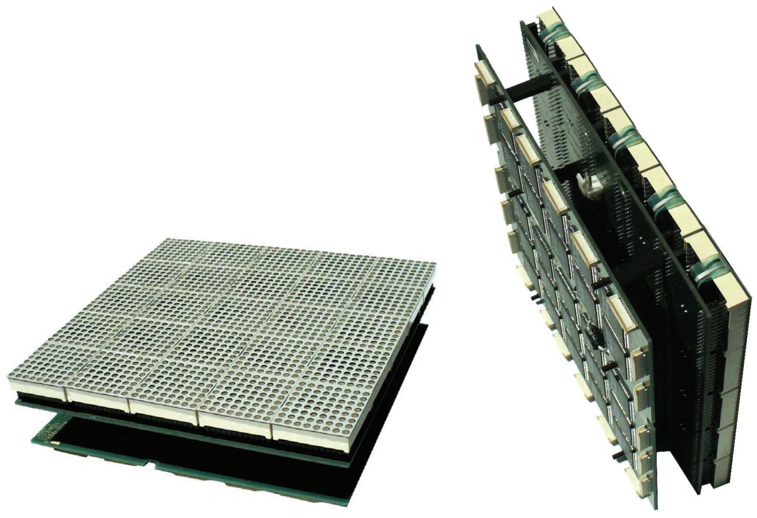


Figure 3.8: *Picture of the* BioStack.

Scalability in cellular architecture is crucial, and the *BioStack* is not the final result of our research, but an element of the system. Several of theses boards can be joined together, through border connectors in the *BioLogic* board, in a two-dimensional array. Connections are realized using flat wires linking adjacent boards.

3.2.4 The BioWall: an assemblage of *BioStack* modules

The connection of several boards together potentially allows the creation of programmable logic surfaces of any size. The current BioWall configuration that has been built and tested consists of 160 *BioStack*s, organized as a 5 by 32 rectangular array.

Due to the large BioWall sizes, we had to build a specific iron frame, or *rack*, that can hold precisely ten *BioStack*s on its structure. Figure 3.9 represents such a rack seen from the back. Once the *BioStack*s are fixed onto the rack, they are connected together using flat wires linking each adjacent FPGA together.

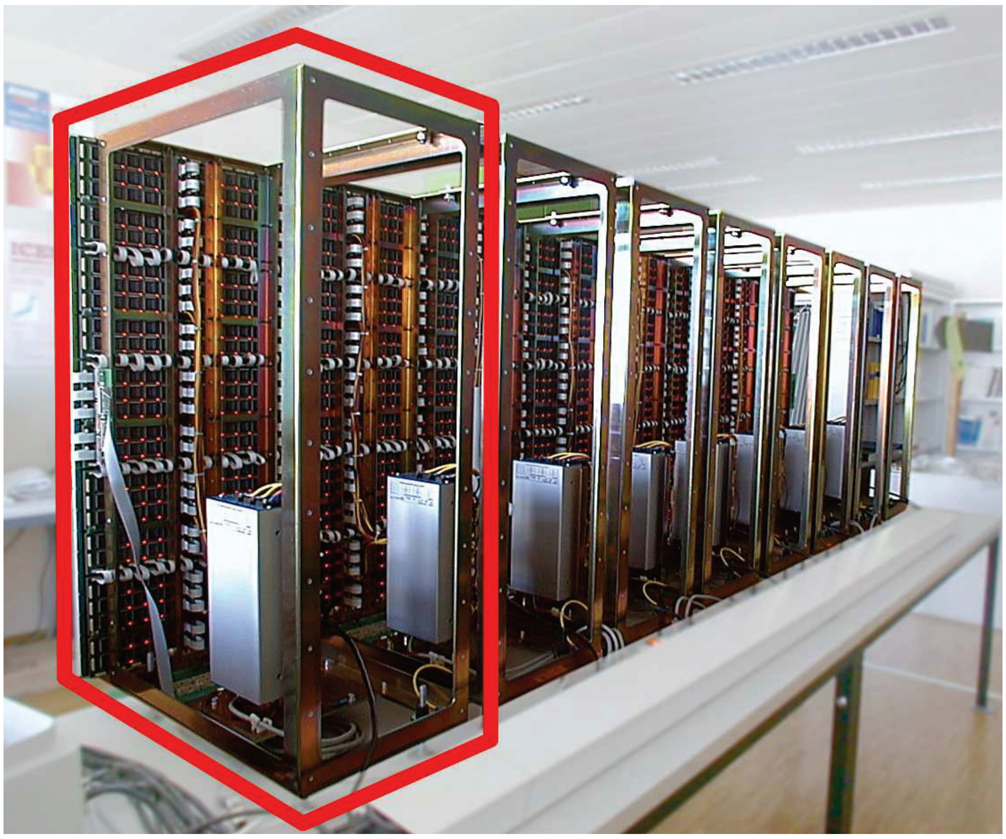


Figure 3.9: *BioWall's rear with the red frame surrounding a rack.*

The whole BioWall is built using 16 racks coupled together, edge-to-edge (horizontally). If another *BioStack* arrangement were be needed, racks could be added edge-to-edge to extend the wall surface.

Each rack holds two other essential elements: a power supply and the *BioBottom* board. This board is fixed at the bottom of the rack and gathers control signals coming from the *BioBox*. These signals are then transmitted to the two *BioStack*s located at the bottom of the rack, and thereafter relayed to the upper *BioStack* modules. Thus each rack becomes an autonomous unit with up to 250 molecules (or FPGAs), its power supply, and the connector allowing all the reconfigurable elements of the *BioLogic* boards to be driven.

Power supply

As previously noted, each rack holds a global power supply block that converts a 220 AC voltage to a regulated 3.3 V DC voltage. Since current consumption of the ten *BioStack*s constituting a rack can be up to 40 A, we wired all the power cables from each *BioStack* to an electrical distribution block fixed near the power convertor output.

Interface boards

The BioWall width is 5.3 meters, which means that a signal propagated from the *BioBox* (which is for technical purposes placed on one side of the machine) to the other side of the BioWall needs to go through several meters of wires. Since these signals are 3.3 V logic CMOS, they will be too sensitive to noise perturbations. The *BioBottom* board placed at the bottom of each rack will avoid this kind of problem, by amplifying signals coming from the *BioBox* (Fig. 3.10).

The same kind of boards, named *BioSide* are present on both sides of the BioWall and are also responsible for amplifying signals coming in or out from the BioWall sides.

Both cards (*BioBottom* and *BioSide*) are connected through 8 meter shielded wires to the *BioBox*. This extreme long length, coupled with the small bandwidth of our amplifiers, has the drawback of altering the signal quality. Thus, we limited the clock frequency and all other signals inside the BioWall to 1.0 MHz.

3.2.5 The *BioBox*

All previously described elements make up the BioWall's surface, but are unusable without the *BioBox*, which is the crucial module for the BioWall to work. This box (Fig. 3.11) is powered by a dedicated power supply and generates from a specific FPGA all the signals driving the BioWall. This box provides an interface between all the racks and the *BioSoft* software running on a PC. The multiple tasks executed by the *BioBox* are (1) the BioWall FPGA configuration, (2) the clock and enable signals generation, (3) the serial stream transmission, and (4) the BioWall status monitoring.

1. All the 4'000 SPARTAN® FPGAs can only be configured with the same configuration. The configuration data is sent by the *BioSoft* to the *BioBox* through a RS232 serial line. The *BioBox* is responsible for clearing the FPGA contents, and transmitting the bitstream to all the FPGAs at the same time. The success of this operation is signalled by a LED near each FPGA, which lights-up red. Once all the FPGAs are configured, the *BioBox* generates a reset signal which will start, at the same time, the application in all the BioWall FPGAs.

2. An accurate oscillator generates the 1.0 MHz frequency which is distributed to all the 4'000 FPGAs in parallel. The *BioSoft* software can send commands for setting the frequency of the enable signals over the RS232 bus. The *BioBox* FPGA will then generates the desired timing for both X_EN and F_EN. New RS232 commands are only necessary to change the desired frequency for the enable signals or to switch them into the step-by-step mode.

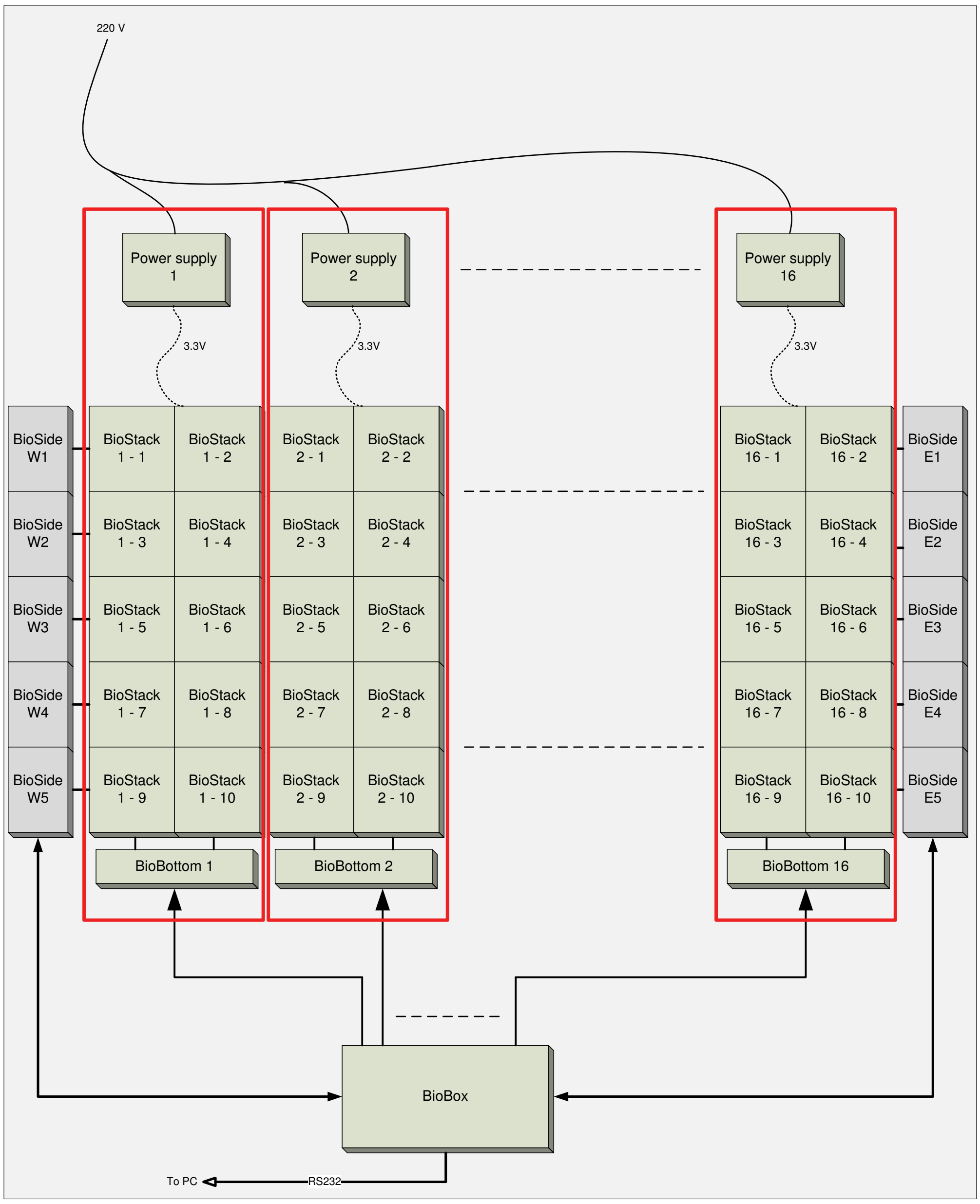


Figure 3.10: *Schematic of the BioWall architecture; each red rectangle correspond to a rack.*

3. Several applications need to receive computation data which will be processed. These data are transmitted by the *BioSoft* software and then transmitted over a serial link to one or more FPGAs on one edge (the signal goes through the *BioSide* boards).

4. BioWall status polling is possible. In the *BioWatch* application [124], we need to monitor the watches death in order to reset (or regenerate) a new watch or-

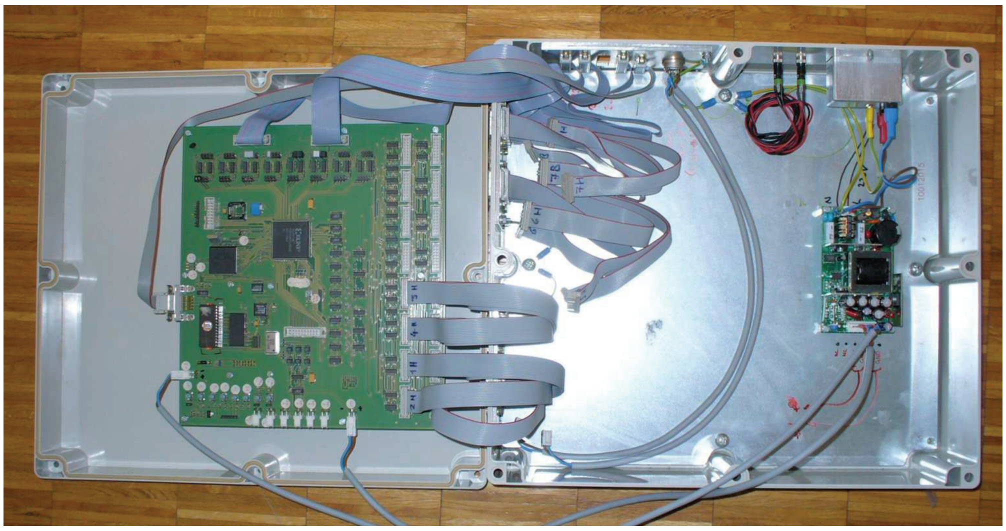


Figure 3.11: *The* BioBox *opened; the controller board is fixed in the cover on the left ; the right box helds all connectors where each cable is going to the racks.*

ganism. This kind of information can then be handled inside the *BioBox* and also be transmitted to *BioSoft*.

The last operation controlled by the *BioBox* is the electrical main status of the whole BioWall. A command sent by the *BioSoft* can switch on or off the power supply of all the racks.

3.2.6 BioSoft

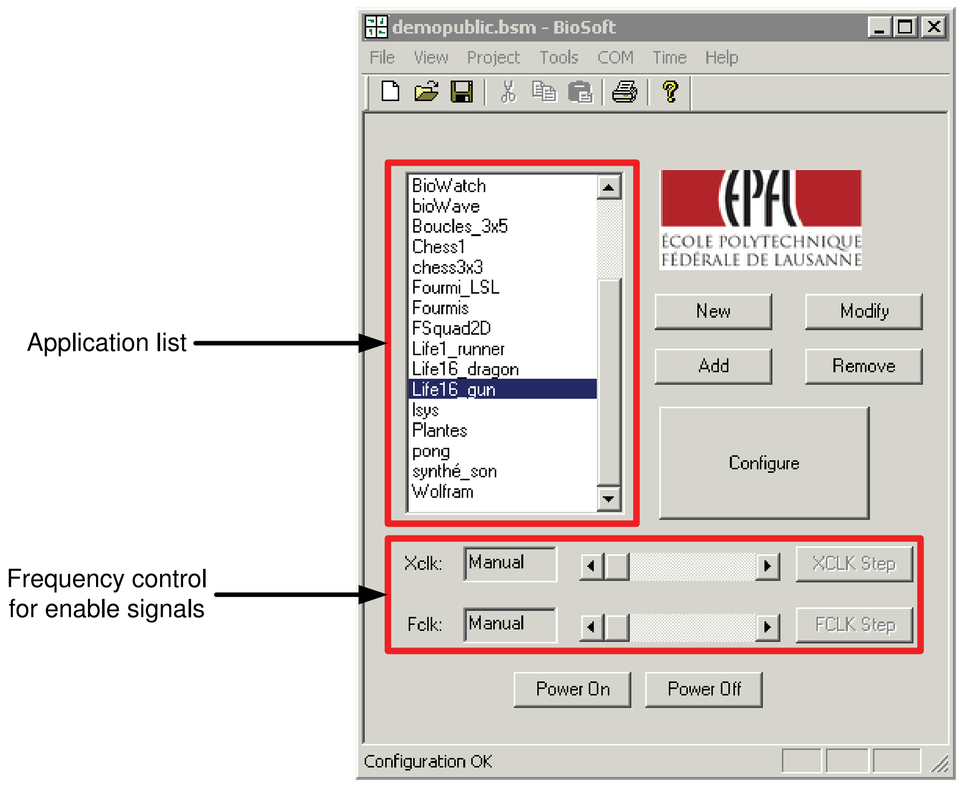
The *BioBox* performs all control tasks at the hardware level, and needs to get configuration streams and orders from a more advanced system which is a software running on a PC in this case.

A simple user interface on the PC allows the user to define a set of files that will be used to configure the tissue (Fig. 3.12). Four kinds of files are currently defined (more can be added): the configuration file for the SPARTAN® FPGAs, and three different formats used to send user-defined data via the input pins at the edges of the tissue (used, for example, to provide an initial configuration for a cellular automaton).

3.3 Conclusion

In this chapter, we described a novel hardware platform aimed at the realization of bio-inspired cellular computing. The versatility and scalability of the platform along with the potential parallel computational power it can provide make possible very interesting research applications, as we will see in the next chapter.

Aside from the computational aspect, the system is also open to several improvements related to I/O aspects. For example, it is clear that the interface between the



Xclk: Manual
Fclk: Manual
Configuration OK

Figure 3.12: BioSoft *main window.*

BioWall FPGAs and an external system can be modified and in order to bypass the *BioBox*. The perfect example of such a modification is the *BiotaMusik* application [60] where the top-right FPGA of the BioWall directly command a sound card which generates and amplifies the sound synthesized inside the wall . Other adaptations are easily possible by using Logidules [15].

Chapter 4

BioWall applications

WHEREAS the previous chapter described the hardware of the BioWall, this chapter focuses on the applications running on this large reconfigurable machine. Our first implementation of a bio-inspired machine is an organism endowed with all the features of the Embryonics project: the *BioWatch* [124, 125], which counts hours, minutes, and seconds, demonstrates the growth and self-repair capabilities of our system.

The implementation of the *BioWatch* would have been sufficient to justify the effort that went into the construction of our embryonic BioWall. However, in developing our machine, we quickly realized that the capabilities of such a platform were not limited to a single application. In fact, it is an ideal platform to prototype many different kinds of two-dimensional cellular systems, which are systems comprising arrays of small, locally connected elements. The BioWall is ideally suited to the implementation of Cellular Automatas (CAs), but is by no means limited to them.

The first application, described in section 4.1, is the famous Game of Life CA. We place the emphasis on showing how this simple automaton can be implemented on the BioWall, and that it performs very highly on this machine thanks to the massively parallel architecture of the latter. In our second application we focus on the development of a novel self-replicating loop endowed with universal construction properties, the cellular structure of which is perfectly suited to the BioWall architecture.

4.1 Game of Life

Life is complexity. The way a spider weaves its web or an ant colony builds its nest suggests that these creatures are intelligent. They are nothing of the sort. Biologists have demonstrated that by blindly following basic rules that have been gradually developed through natural selection, every animal behaves in sometimes extremely complex ways. John Conway's Game of Life [47] is a striking example of this kind of emergent behavior. The game is based on a very simple two-dimensional CA, in which each element represents an individual. Each individual has only two possible states: dead or alive. The next state of each individual depends on the current state of the individual itself and that of its eight nearest neighbors, according to the following rules:

- If the number of living neighbors is too small (zero or one), the individual dies of isolation and its future state is “dead” (loneliness).
- If an individual has exactly two living neighbors, it conserves its current state (persistence).
- If an individual has exactly three living neighbors, its future state is “alive” (reproduction).
- If an individual has too many living neighbors (four or more), it dies of overpopulation and its future state is “dead” (overcrowding).

The initial pattern constitutes the “seed” of the system. The first generation is created by applying the above rules simultaneously to every cell; births and deaths happen simultaneously. Figure 4.1 illustrates the Game of Life rules with some simple examples.

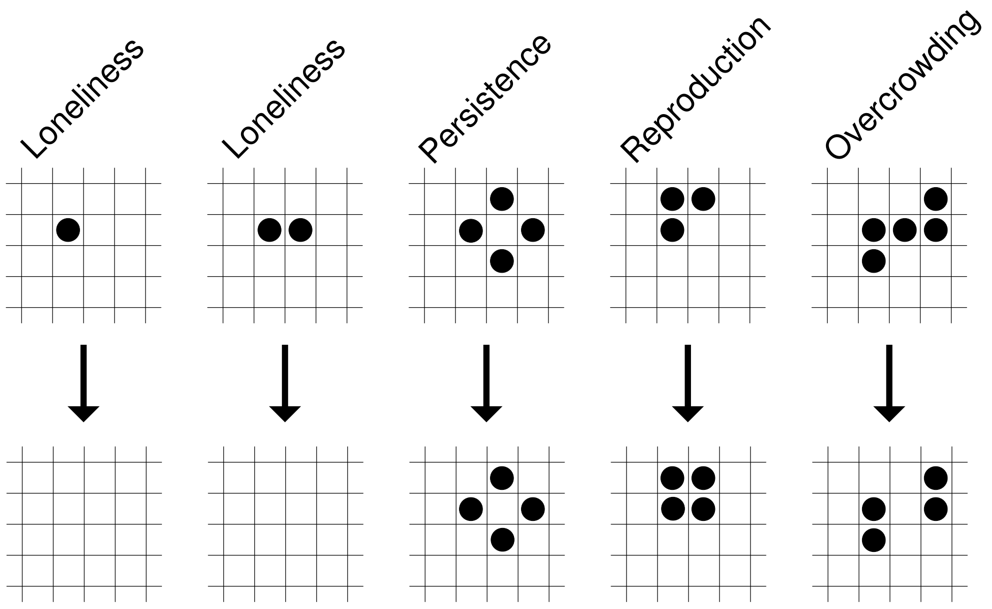


Figure 4.1: *Game of Life evolution rule examples.*

A strikingly visual application, the Game of Life is ideally suited for implementation on the surface of the BioWall, where the touch-sensitive membranes are used to override the states of the game and to give life to the individuals. We developed two different implementations:

- In Life1, each of the 4’000 molecules of the BioWall represents a single individual of the Game of Life. The surface is toroidal and the touch-sensitive membranes toggle the state of the individuals.
- In Life16, each molecule represents an array of 4x4=16 individuals (a total of 64’000 Game of Life cells on the BioWall), and the touch-sensitive membranes insert a glider (one of the stable configurations of the game of Life, which moves diagonally across the space as illustrated in figure. 4.2).

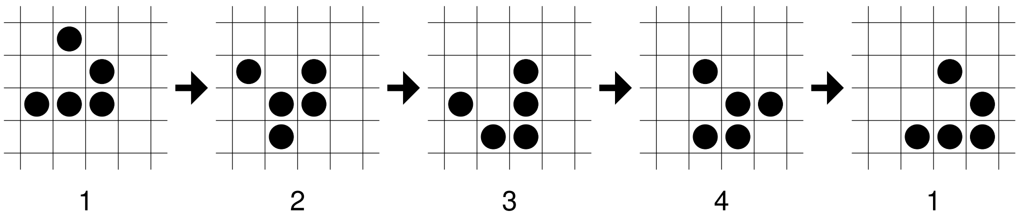


Figure 4.2: *A Game of Life stable pattern: the glider*

This application, while not of direct interest for our research (it is mostly aimed at providing direct interaction for the general public), is nevertheless a good example of an application ideally suited to illustrate the features (display the capabilities, interactivity, etc.) of the BioWall. Moreover, we chose to implement this simple CA in order to describe how a simple application can be adapted to the BioWall.

4.1.1 Version 1: Life1

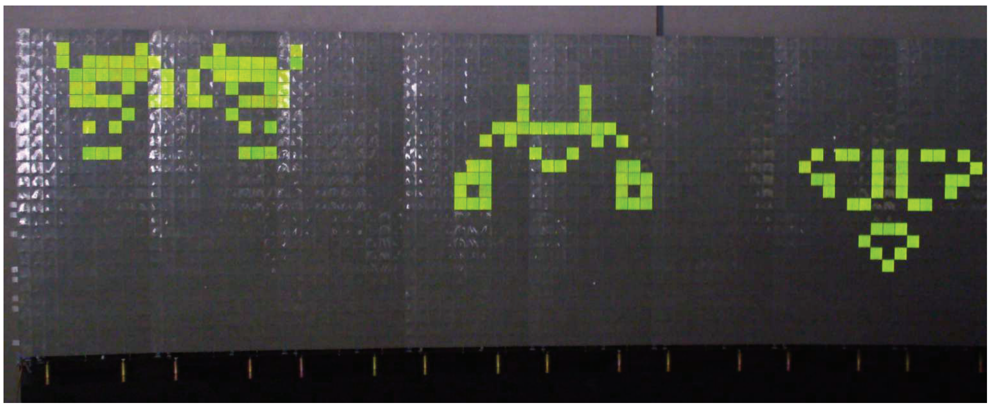


Figure 4.3: *Life1 running on the BioWall.*

The first implementation of the Game of Life automaton is Life1 (Fig. 4.3), which is perfectly suited to the BioWall architecture since each cell of the automaton can be implemented in one molecule, i.e. one FPGA. Local connections between each adjacent FPGA and a global synchronous clock allow the Game of Life computations to be parallelized. Since the automaton clock is adjustable using the *BioSoft*, the operator can switch the automaton to a frozen state and can then introduce the state of each cell. Once this task is performed, the automaton clock can be restarted to compute the previously inserted Game of Life pattern.

Another way to determine the initial state of each automaton is to set the state of the 4'000 cells of the BioWall just after having configured all the FPGAs. This process is accomplished by sending a configuration stream to all Game of Life cells, as detailed below.

For the first implementation of this CA on the BioWall, each FPGA acts as a cell of the automaton, where its new state is computed depending on the state of its eight neighbors. All cells are synchronized by a global signal, which is the X_EN enable signal in our application.

For each FPGA, we enumerated three main tasks:

- Computation of the new automaton state.
- Loading and configuring each cell with a user-defined initial state.
- Interacting with people by detecting touches on the BioWall surface and displaying the state of the Game of Life cell.

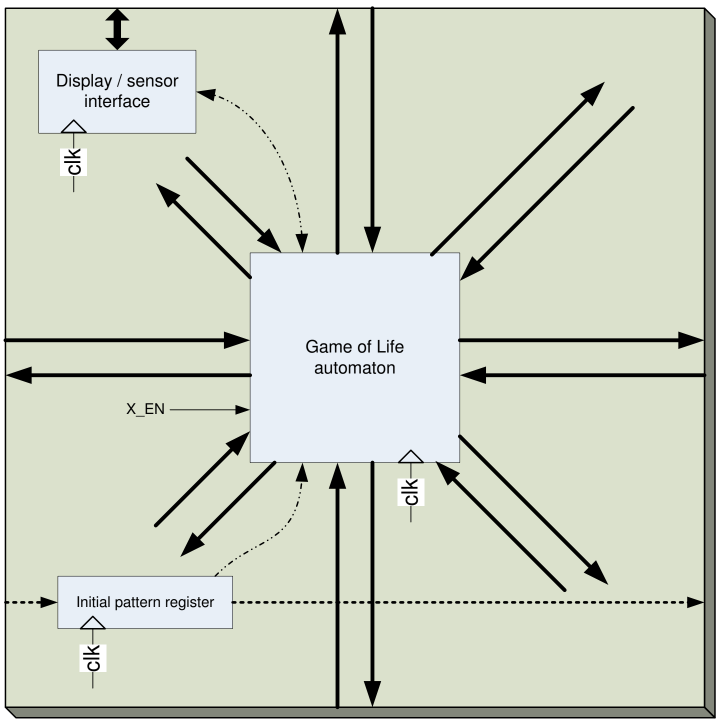


Figure 4.4: *inside the FPGA: Game of Life architecture.*

These tasks and mutual interactions can be illustrated in figure 4.4. Figure 4.5 highlights the connections of FPGAs with their neighbors. Since in the BioWall architecture each FPGA is directly connected with only four cardinal neighbors, we had to redirect some connections, as displayed in figure 4.5, to create the diagonal links needed by the Moore neighborhood.

The automaton state, which is only one-bit variable, can be “0” or “1”, dead or alive. In the alive state all of the green Light-Emitting Diodes (LEDs) of the BioWall molecule are lit, while in the dead state all of the molecule’s LEDs are turned off (black state). Each pressure exerted on the touchsensor toggles the state of the automaton from dead to alive or vice-versa. This application, like all of the other applications on the BioWall, is clocked at a fixed frequency of 1.0 MHz. However in order to display

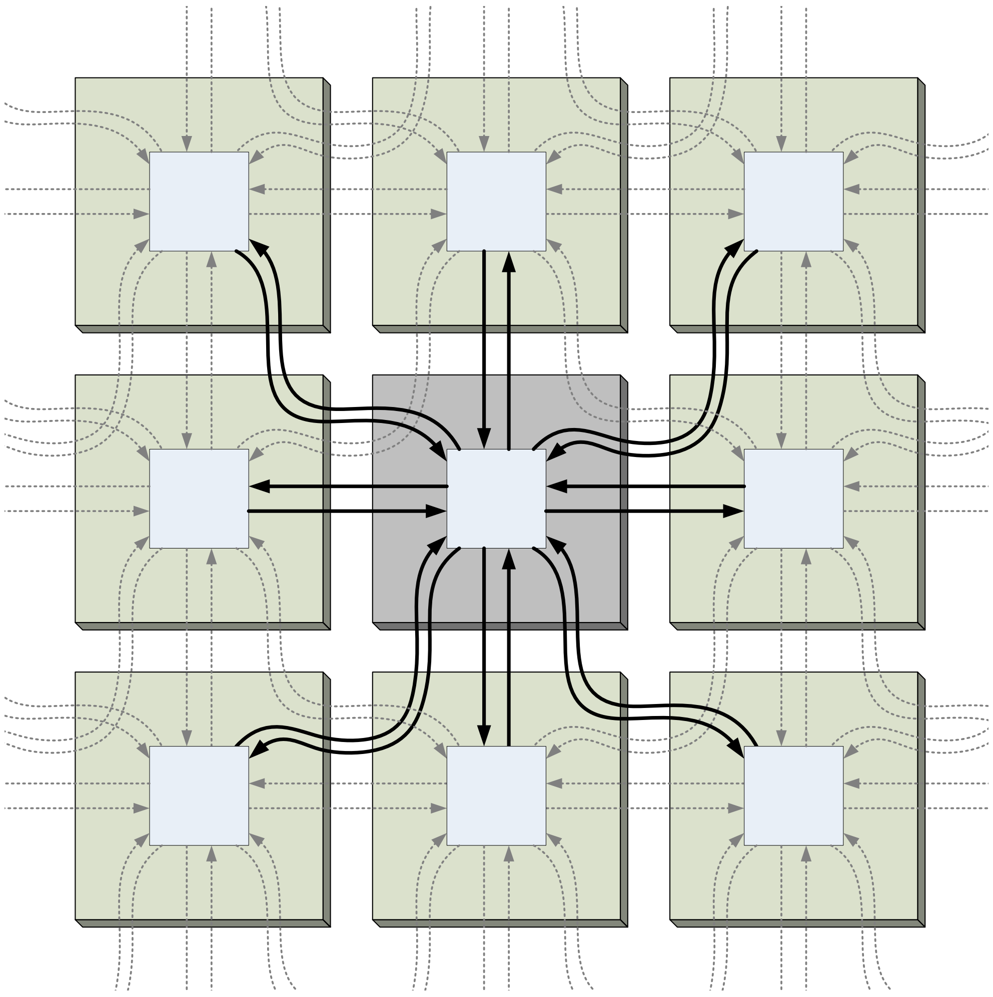


Figure 4.5: *Game of Life communication structure; each big gray box represents a FPGA and the white box inside is a Game of Life cell core.*

the automaton evolution process, we slowed down the automaton's computation speed, using the X_EN signal. This signal, which can be controlled by the user from the *BioSoft* application, allows very low computation speeds to be set. Thus, the next state of the Game of Life automaton is computed when X_EN signal is high and when a global clock rising edge occurs.

We previously mentioned that the user could load an initial pattern; the next paragraph will give some details on how this task is performed on the BioWall

Initial pattern

Figure 4.3 gives an example of a predefined pattern which, in this case, will generate three spaceships moving from top to bottom. Several others predefined patterns exist and can be found on the Internet [46].

To avoid executing the whole FPGA Xilinx® synthesis process each time we wanted to change the initial pattern, we decided to use a file with the initial states of each automaton cell along with common FPGA configuration. This is done by sending a serial stream after the FPGA configuration. In this file sent by the *BioSoft* software, each bit of the file represents the state of one of the 4'000 automaton cells. These data are transmitted to the BioWall through a specific pin located in the bottom left FPGA of the wall. The data are shifted to the next FPGA as depicted in figure 4.6. This chain acts as a 4'000-bit long Shift Register (SR), where each flip-flop of the SR is located inside one FPGA. Once the 4'000 bits are shifted and stored in their respective flip-flop, they are transferred to the Game of Life molecule where they determine the initial state (see "initial pattern register" in figure 4.4).

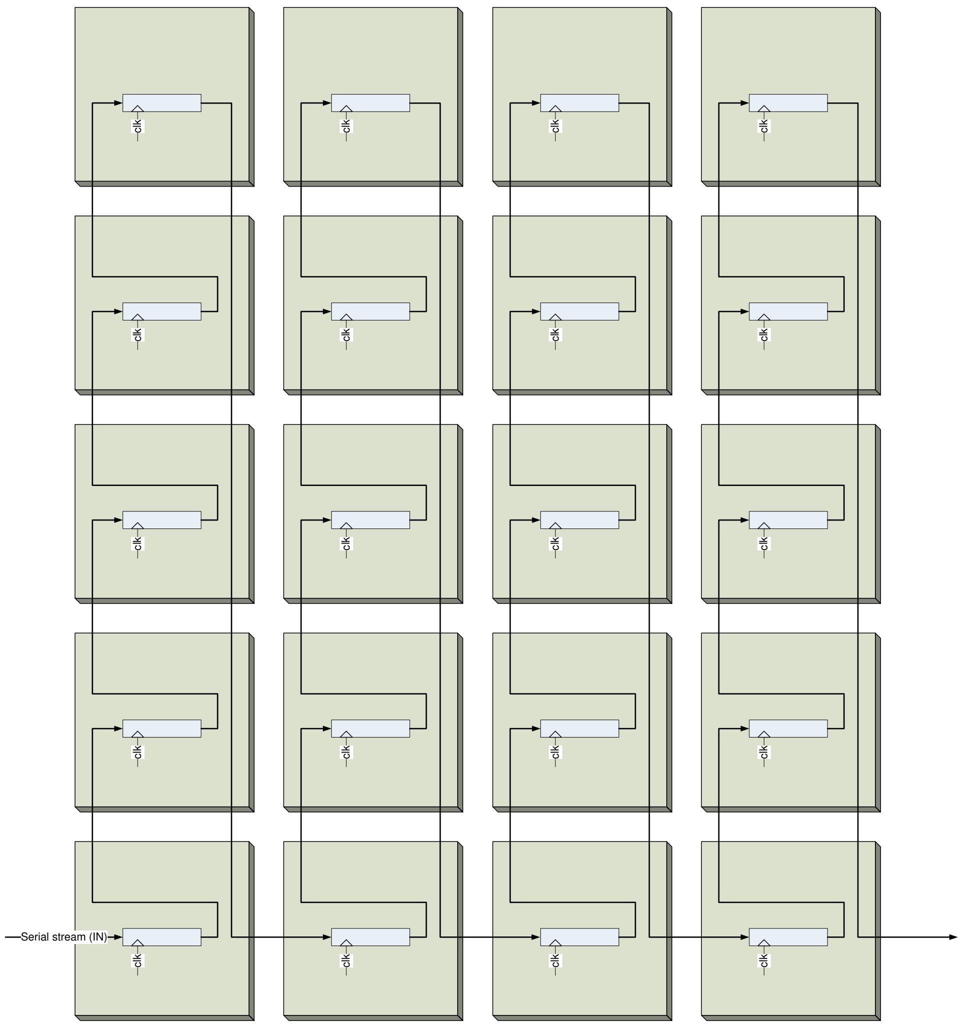


Figure 4.6: *configuration scheme for loading the Game of Life initial pattern ; each big gray box represents a FPGA and the white box inside is a serial shift register. (This process is showed over a reduced BioWall of only 5 rows.)*

4.1.2 Version 2: Life16

Life1 shows us an implementation of Conway's Game of Life on the BioWall. This application does not use all the resources of the BioWall and FPGA utilization percentage is really low. Life16 version (Fig. 4.7), reuses the previously described application, but instantiates 16 cells in each FPGA, thus giving 64'000 cells over the BioWall surface, and assigning a unique Game of Life cell to every 2×2 pixels of the BioWall. In this experiment, we changed the interaction method so that each pressure over the touchsensor would generate a glider (Fig. 4.2).

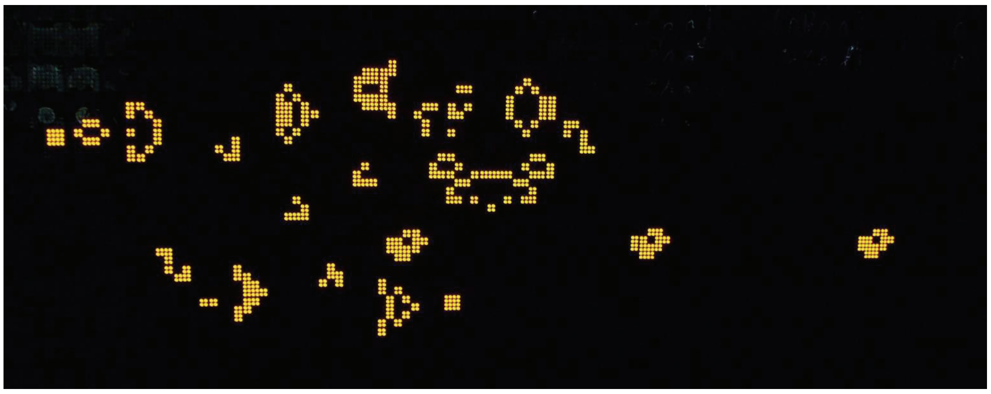


Figure 4.7: *Life16 running on the BioWall.*

4.1.3 Performance

The Game of Life automaton is probably one of the best known CAs and has been studied for a long time. In fact, a lot of computer software allows to simulate a large Game of Life array.

However, our hardware-only solution gives some really interesting results, which would be difficult to obtain on a computer. Even if the BioWall computes at very slow speeds, the parallelization of thousands of specialized cells exceeds the capability of any Personal Computer (PC). For comparison, we simulated an equivalent Game of Life surface on a PC[1] running the Xlife software [143]. This program gave us a maximum computation speed of 871 generations/second for 64'000 cells configured with an initial random pattern. In the same time the BioWall can compute as many as 200'000 generations/s.

Table 4.1 gives some comparisons between software and hardware implementations. Even if a PC can compute infinite sized organisms, it rapidly becomes slower than our Life16 application. A major advantage of the BioWall version over a PC software version is interaction, where multiple users can play together on the BioWall.

4.1.4 Game of Life conclusion

This application gives us a first overview on how CAs are easily implemented on the BioWall. Moreover, whereas the BioWall performances seem to be really limited in

[1]Pentium 4, running at 1.8 Ghz, with 1.0 GByte RAM; Linux operating system.

	Computer	BioWall
organism size	+	-
simulation speed	-	+
constant time step	-	+
interaction	-	+

Table 4.1: *Game of Life comparisons: computer/BioWall.*

speed, the BioWall's parallel architecture allows it to surpass the PC's powerful capabilities.

4.2 Tom Thumb

After a survey of the theory and some realizations of self-replicating machines, this section presents a novel self-replicating loop endowed with universal construction properties. Based on the hardware implementation of the so-called Tom Thumb algorithm, the design of this loop leads to a new kind of cellular automaton made of a processing and a control unit. The self-replication of the "LSL" acronym serves as an artificial cell division example of the loop and results in a new and straightforward methodology for the self-replication of computing machines of any dimensions.

4.2.1 Introduction and survey

Self-replicating loops

The main goal of this application is to present a new self-replicating loop endowed with universal construction properties. While the early history of the theory of self-replicating machines is basically the history of John von Neumann's thinking on the matter [136], a practical implementation requires a sharply different approach. It was finally Langton, in 1984, who opened a second stage in this field of research. In order to construct a self-replicating automaton simpler than von Neumann's, Langton [71] adopted more liberal criteria. He dropped the condition that the self-replicating unit must be capable of universal construction and computation.

Langton proposes a configuration in the form of a loop, endowed notably with a constructing arm and a replication program or genome, which turns counterclockwise. After 151 time steps, the original loop (mother loop) produces a daughter loop, thus obtaining the self-replication of Langton's loop.

To avoid conflicts with biological definitions, we do not use the term "cell" to indicate the parts of a cellular automaton, opting rather for the term "molecule". In fact, in biological terms, a *cell* can be defined as the smallest part of a living being which carries the complete blueprint of the being, that is the being's *genome*.

According to the biological definitions of a cell, we end up with the following observations.

- Langton's self-replicating loop is a unicellular organism: its genome requires 28 molecules and is a subset of the complete loop which requires 94 molecules.

- The size of Langton's loop is perfectly reasonable, since it requires 94 molecules, thus allowing complete simulation.
- There is no universal construction nor calculation: the loop does nothing but replicate itself. Langton's self-replicating loop therefore represents a special case of von Neumann's self-replication of a universal constructor. The loop is a non-universal constructor, capable of building, on the basis of its genome, a single type of machine: itself.

As did von Neumann, Langton emphasized the two different modes in which information is used, interpreted (*translation*) and uninterpreted (*transcription*). In his loop, translation is accomplished when the instruction signals are executed as they reach the end of the construction arm, and upon collision of signals with other signals. Transcription is accomplished by duplication of signals at the arm junctions.

More recently, Byl [21] proposed a simplified version of Langton's automaton. Last but not least Reggia *et al.* [104] discovered that having a sheath surrounding the data paths of the genome was not essential, and that its removal led to smaller self-replicating structures which also have simpler transitions functions. Moreover, they found that relaxing the strong symmetry requirement consistently led to transition functions that required fewer rules than the corresponding strong symmetry version.

Self-replicating loops with computing capabilities

All the previous loops lack any computing and constructing capabilities, their sole functionality being that of self-replication. Lately, new attempts have been made to redesign Langton's loop in order to embed such calculation possibilities. Tempesti's loop [128] is thus a self-replicating automaton, with an attached executable program that is duplicated and executed in each of the copies. This was demonstrated for a simple program that writes out (following the loop's replication) the "LSL", acronym of the Logic Systems Laboratory. Finally, Perrier *et al.*'s self-replicating loop [94] shows some kinds of universal computational capabilities. The system consists of three parts, loop, program, and data, all of which are replicated, followed by the program's execution on the given data.

So far, all self-replicating loops lack universal construction, i.e. the capability to construct a computing machine of any dimension, even though this goal is of great interest in the development of new cellular automata, for example three-dimensional reversible cellular automata designed by Imai *et al.* [56] for the emerging field of nanotechnologies.

Self-replicating loop with universal construction

Our main goal is to show that a new algorithm, the *Tom Thumb algorithm*, will make it possible to design a self-replicating loop with universal construction easily implemented into silicon.

A second goal is to generalize the notion of the classical "cellular automaton" by introducing the Data and Signals Cellular Automaton (DSCA) which perfectly suits the specifications of our basic molecule. Moreover, such an automaton will allow a straightforward and systematic methodology for synthesizing cellular automata, a methodology which is completely absent at the present time.

In § 4.2.2, our new algorithm will be described by means of a minimal mother cell composed of four molecules which will grow and then divide to trigger the growth of two daughter cells. This example is sufficient to derive the detailed architecture of the basic molecule. Paragraph 4.2.3 deals with the generalization of the methodology previously described and its application in a real example, the self-replication of the "LSL" acronym. Universal construction and computation are briefly demonstrated. We will conclude by opening new avenues based on the self-replicating loop with universal construction and by showing the experimental results on the BioWall.

4.2.2 The Tom Thumb algorithm for cell division

Cell division in living organisms

Before describing our new algorithm for the division of an artificial cell, let us remember the two key roles that cellular division plays in the existence of living organisms.

- The construction of two daughter cells in order to grow a new organism or to repair an existing one (genome *translation*).
- The distribution of two identical sets of chromosomes in order to create two copies of the genome from the mother cell in order to program the daughter cells (genome *transcription*).

Starting with a minimal cell made up of four artificial molecules, we will propose a new algorithm, the *Tom Thumb algorithm*, to construct both the daughter cells and the associated genomes. This algorithm will finally allow us to derive the detailed architecture of our final molecule. A tissue comprised of such molecules will be endowed with both universal construction and computation properties.

Initial conditions

The minimal cell compatible with our algorithm is made up of four molecules, organized as a square of two rows by two columns (Fig. 4.8). Each molecule is able to store in its four memory positions four hexadecimal characters of our artificial genome, and the whole cell thus contains 16 such characters.

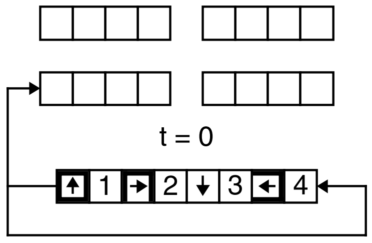


Figure 4.8: *The minimal cell* (2×2 *molecules) with its genome at the start* ($t = 0$).

The original genome for the minimal cell is organized as a string of eight hexadecimal characters, i.e. half the number of characters in the cell, moving counterclockwise by one character at each time step ($t = 0, 1, 2, ...$).

The 15 hexadecimal characters composing the alphabet of our artificial genome are detailed in figure 4.9. They are either *empty data* (0), *molcode data* (for molecule code data, from 1 to 7) or *flag data* (from 8 to E). Molcode data will be used for configuring our final artificial organism, while flag data are indispensable for constructing the skeleton of the cell. Furthermore, each character is given a status and will eventually be *mobile data*, indefinitely moving around the cell, or *fixed data*, permanently trapped in a memory position of a molecule.

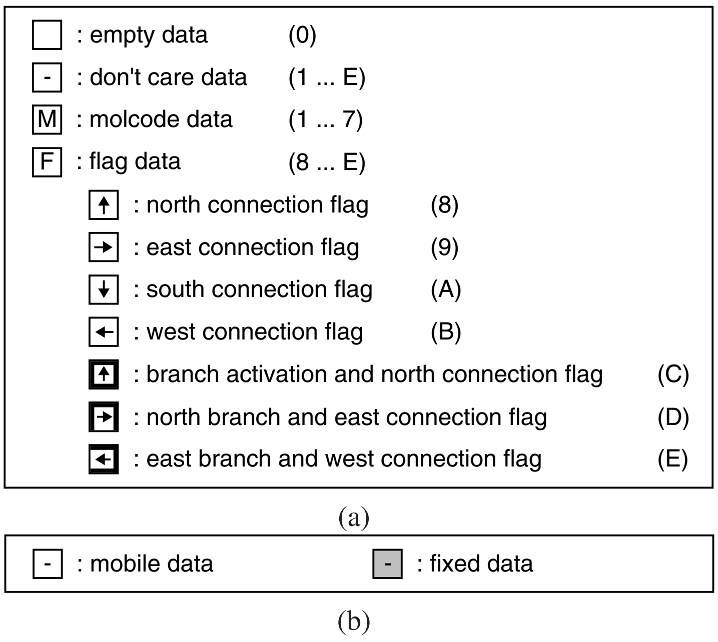

: empty data (0)
- : don't care data (1 ... E)
M : molcode data (1 ... 7)
F : flag data (8 ... E)
: north connection flag (8)
: east connection flag (9)
: south connection flag (A)
: west connection flag (B)
: branch activation and north connection flag (C)
: north branch and east connection flag (D)
: east branch and west connection flag (E)
(a)
- : mobile data
- : fixed data
(b)

Figure 4.9: *The 15 characters forming the alphabet of an artificial genome. (a) Graphical and hexadecimal representations of the 15 characters. (b) Graphical representation of the status of each character.*

Constructing the cell

At each time step, a character of the original genome is shifted from right to left and simultaneously stored in the lower leftmost molecule (Figs. 4.8 and 4.10). Note that, due to our algorithm, the first, third, etc. character of the genome (i.e. each odd character) is always a flag F, while the second, fourth, etc. character (i.e. each even character) is always a molcode M. The construction of the cell, i.e. storing the fixed data and defining the paths for mobile data, depends on two major patterns (Fig. 4.11).

- If the two, three or four rightmost memory positions of a molecule are empty (blank squares), the characters are shifted by one position to the right (shift data).

- If the rightmost memory position is empty, the characters are also shifted by one position to the right (load data). In this situation, the rightmost F' and M' characters are trapped in the molecule (fixed data), and a new connection is established from the second leftmost position toward the northern, eastern, southern or western molecule, depending on the fixed flag information (F' = 8 or C, 9 or D, A, B or E).

At time $t = 16$, 16 characters, i.e. twice the contents of the original genome, have been stored in the 16 memory positions of the cell (Fig. 4.10). Eight characters are fixed data, forming the phenotype of the final cell, and the eight remaining ones are mobile data, composing a copy of the original genome, i.e. the genotype. Both *translation* (i.e. construction of the cell) and *transcription* (i.e. copy of the genetic information) have therefore been achieved.

The fixed data trapped in the rightmost memory position of each molecule remind us of the pebbles left by Tom Thumb to remember his way back.

Dividing the mother cell into two daughter cells

In order to grow an artificial organism in both the horizontal and vertical directions, the mother cell should be able to trigger the construction of two daughter cells, northward and eastward.

At time $t = 11$ (Fig. 4.10), we observe a pattern of characters which is able to start the construction of the northward daughter cell; the upper leftmost molecule is characterized by two specific flags, i.e. a fixed flag indicating a north branch (F = D) and the branch activation flag (F = C). This pattern is also visible in figure 4.12 (northward signal, third row). The new path to the northward daughter cell will start from the second leftmost memory position.

At time $t = 23$, another particular pattern of characters will start the construction of the eastward daughter cell; the lower rightmost molecule is characterized by two specific flags, i.e. a fixed flag indicating an east branch (F = E), and the branch activation flag (F = C). This pattern also appears in figure 4.12 (eastward signal, third row). The new path to the eastward daughter cell will start from the second leftmost memory position.

The other patterns in figure 4.12 are needed for constructing the inner paths of the minimal cell (Fig. 4.10) as well as a cell more complex than the minimal cell, for example that of figure 4.18(b).

Growing a multicellular organism

In order to analyze the growth of a multicellular artificial organism, careful observation of the interactions between the different paths created inside and outside each individual cell are needed.

As for the initial conditions, we suppose that at time $t = -1$ a first path is constructed from the shift register storing the original genome (Fig. 4.8) to the lower leftmost molecule. After each period of eight time steps (i.e. t = 7, 15, 23, ...), the same order will trigger the construction of this path again (Fig. 4.13(a)).

The construction of the cell is characterized by the successive launch of four inner paths northward ($t = 3$), eastward ($t = 7$), southward ($t = 11$), and westward ($t = 15$)

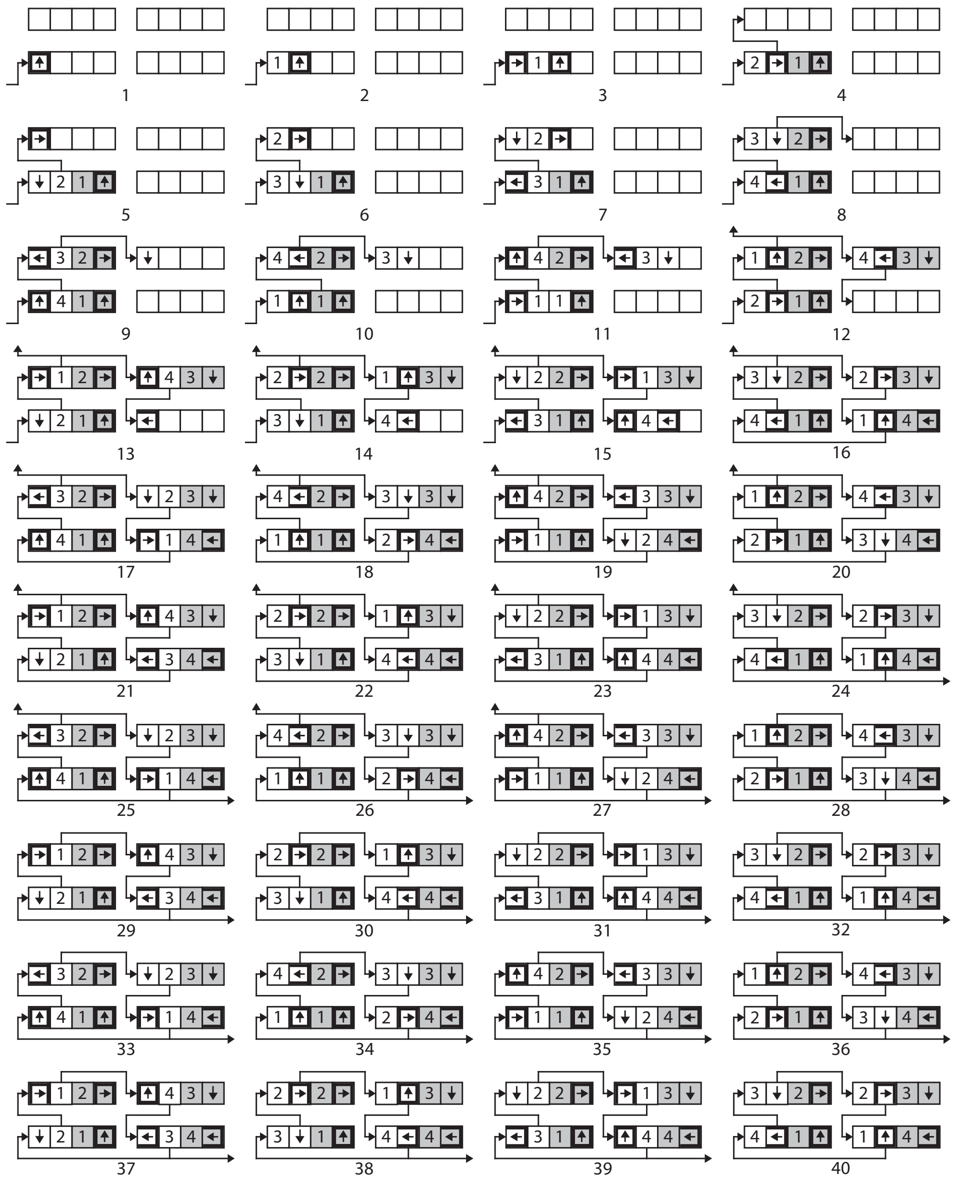


Figure 4.10: *Constructing the minimal cell ($t = 4$: north path, $t = 8$: east path, $t = 12$: south path and north branch, $t = 16$: west path and loop completion, $t = 24$: east branch, $t = 28$: north branch cancelation, $t = 40$: east branch cancelation).*

(Fig. 4.13(a)). Due to our algorithm, this construction is carried out only once, and these paths are never reactivated. Just notice the collision between the two signals at time $t = 15$ where priority is given to the westward inner path.

Finally, the division of the mother cell into two daughter cells will trigger a north-

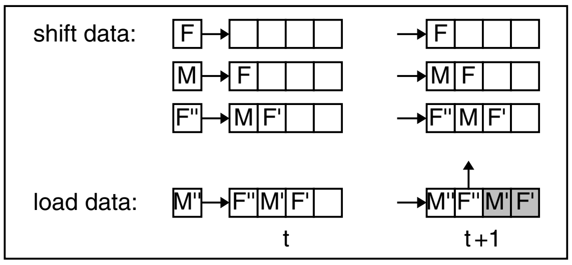


Figure 4.11: *The two memory patterns for constructing a cell.*

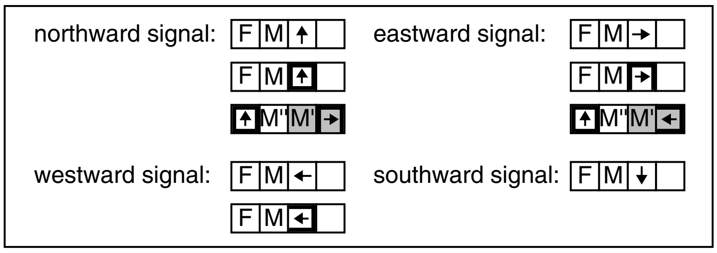


Figure 4.12: *Patterns of characters triggering the paths to the north, east, south and west molecules.*

ward outer path at time $t = 11$. Due to our algorithm, this path is reactivated periodically every eight time steps, i.e. t = 19, 27, 35, ... For the same reason, the cell division will trigger another eastward outer path at time $t = 23$; this path is also reactivated periodically every eight time steps, i.e. t = 31, 39, ...

A macroscopic representation of the mother cell is given in figure 4.13(b) where the different activation times of the initial path (ti = -1, 7, 15, 23, 31, 39, ...), the northward outer path (tn = 11, 19, 27, 35, ...), the eastward outer path (te = 23, 31, 39, ...), and the inner path closing the loop ($tc = 15$) are summarized. It is then possible to derive the number of time steps Δtn between the occurrence of the first initial path ($ti = -1$) and the first northward outer path ($tn = 11$):

$$\Delta tn = tn - ti = 12 \tag{4.1}$$

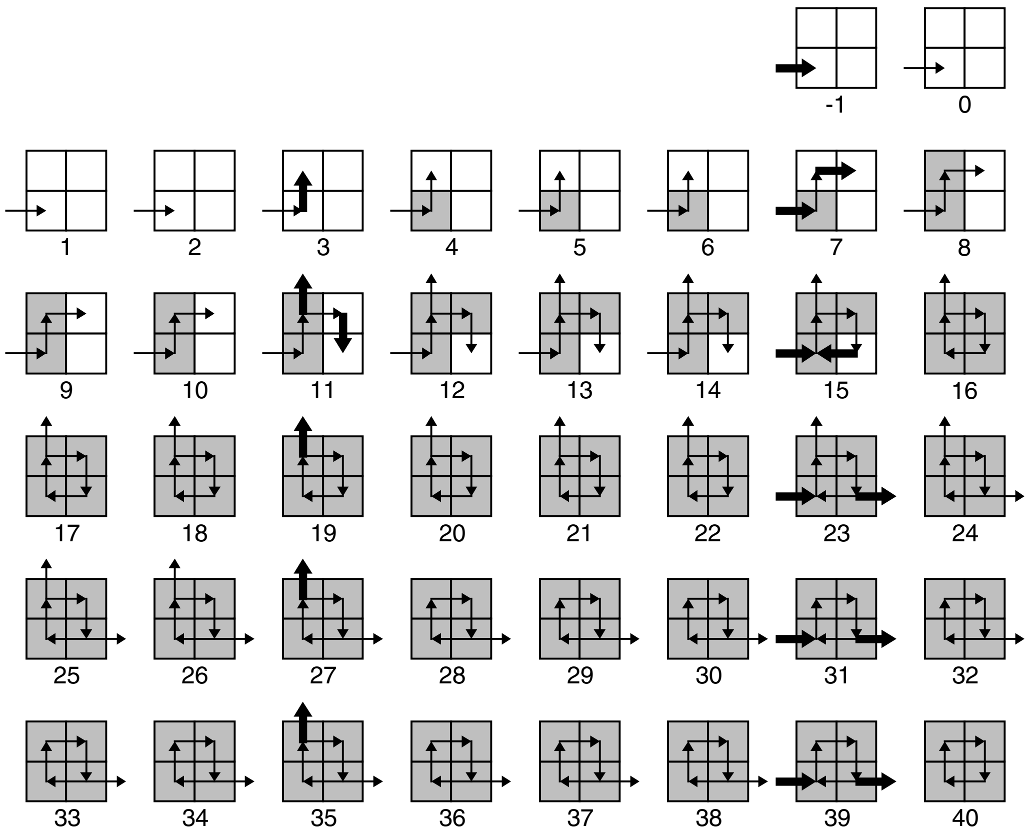
The number of time steps Δte, until the first eastward outer path, becomes:

$$\Delta te = te - ti = 24 \tag{4.2}$$

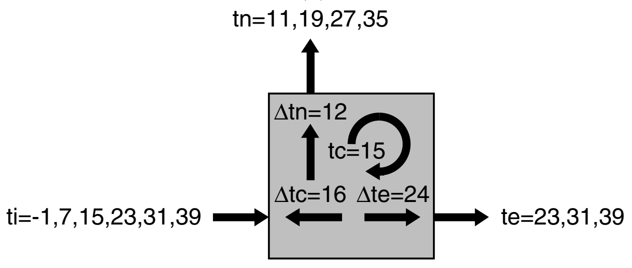
while Δtc, the number of time steps until the inner path closing the loop, corresponds to:

$$\Delta tc = tc - ti = 16 \tag{4.3}$$

Figure 4.14(a) shows the macroscopic representation of a multicellular organism made up of $2 \times 2 = 4$ cells where each path activation (northward, eastward, and closing the loop) is given its precise timing according to the temporal characteristics of the minimal cell (Fig. 4.14(b)). In this cell tn, te and tc are defined as follows:



(a)



(b)

Figure 4.13: *Macroscopic representations of the mother cell. (a) Activated path from* $t = -1$ *to* $t = 40$*. (b) Number of time steps* Δtn, Δte *and* Δtc.

$$tn = ti + \Delta tn + K.8 = ti + 12 + K.8 \tag{4.4}$$

$$te = ti + \Delta te + K.8 = ti + 24 + K.8 \tag{4.5}$$

$$tc = ti + \Delta tc = ti + 16 \tag{4.6}$$

where K is an integer (0, 1, 2, 3, ...).

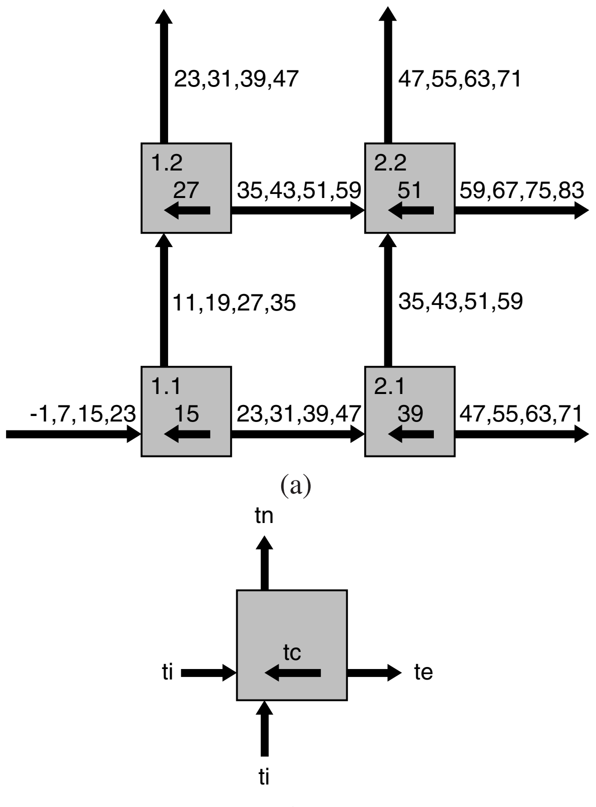


Figure 4.14: *Macroscopic representations of a multicellular organism. (a) The* 2×2 *organism. (b) Temporal characteristics of the minimal cell with the different activation times of the initial path (ti), the northward outer path (tn), the eastward outer path (te), and the inner path closing the loop (tc).*

Defining the priorities between cells

When two or more paths are simultaneously activated, a clear priority should be established. We have therefore chosen three growth patterns (Fig. 4.14(a)).

- For cells in the lower row (1.1 and 2.1) a collision occurs at time $tc = ti + \Delta tc = ti + 16$ between the closing loop and the path entering the lower leftmost molecule. As already mentioned, the inner loop (i.e. the westward path), will have the priority over the eastward path.
- For cells in the leftmost column (1.2), the inner loop (i.e. the westward path), will take priority over the northward path, at the exception of the mother cell 1.1.
- For all other cells (2.2), two types of collisions may occur:
  - between the northward and eastward paths (2-signal collision);
  - between these two previous paths and a third one: the closing loop at time tc (3-signal collision).

In this last case, the northward path will have priority over the eastward path (2-signal collision), and the westward path will have priority over the two others (3-signal collision).

The results of such a choice are as follows: a closing loop has priority over all other outer paths, which makes the completed loop entirely independent on its neighbors, and the organism will grow by developing bottom-up vertical branches. This choice is quite arbitrary and may be changed according to other specifications.

It is now possible to come back to the detailed representation of a multicellular organism made up of 2×2 minimal cells (Fig. 4.15) and exhibit the latter at different time steps in accordance with the priorities mentioned above.

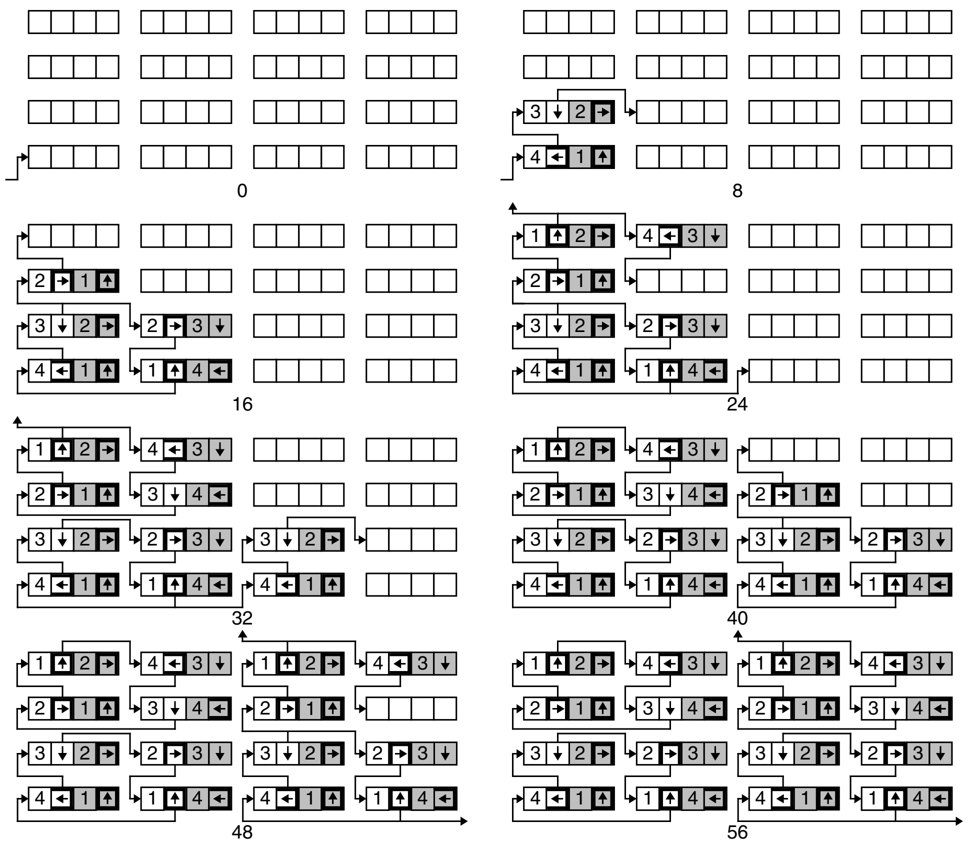


Figure 4.15: *Analyzing a multicellular organism made up of* 2×2 *minimal cells (*$t = 32$*: cell 1.2 closed on itself and independent of its mother cell 1.1,* $t = 40$*: cell 2.1 closed on itself and independent of its mother cell 1.1,* $t = 56$*: cell 2.2 closed on itself and independent of its mother cell 2.1).*

Towards a hardware implementation: the Data and Signals Cellular Automaton (DSCA)

We are now able to describe the detailed architecture of our actual molecule (Fig. 4.16(a)) which is made up of two main parts, an upper part or *processing unit*

(PU) and a lower part or *control unit* (CU) (Fig. 4.16(b)). The *processing unit* itself consists of three units.

- An input unit, the multiplexer DIMUX, selecting one out of the four input data ($NDI$3:0, $EDI$3:0, $SDI$3:0 or $WDI$3:0), plus the empty data 0000; this selection is operated by a 3-bit control signal $I$2:0.
- A 4-level stack organized as two *genotypic registers* GA3:0 and GB3:0 (for mobile data), and two *phenotypic registers* PA3:0 and PB3:0 (for fixed data) according to the definitions of figure 4.11. The two phenotypic registers are idle (i.e. performing the HOLD operation) only when the rightmost memory position of the molecule is a flag (i.e. HOLD=$PB3 = 1$).
- An output unit, the buffer DOBUF, which is either active ($PB3 = 1$, flag in the rightmost memory position) or inhibited.

The *control unit* is itself divided into two units.

- An input encoder ENC, a finite state machine calculating the 3-bit control signal $I$2:0 from the four input signals NSI, ESI, SSI, and WSI. The specification of this machine, which depends on the priorities between cells as mentioned above (Figs. 4.13(a) and 4.14(a)), is described by the state graph of figure 4.16(d). The five internal states QZ, QN, QE, QS, and QW will control the multiplexer DIMUX for choosing the input value 0000 or the input data $NDI$3:0, $EDI$3:0, $SDI$3:0 or $WDI$3:0 respectively.
- An output generator GEN, which is a combinational system producing the northward, eastward, southward, and westward signals (NSO, ESO, SSO, and WSO) according to the patterns described in figure 4.12.

The processing unit (PU) and control unit (CU), in addition to the final molecule, are represented at macroscopic levels in figures 4.16(b) and 4.16(c); these figures define a new kind of generalized cellular automaton, the Data and Signals Cellular Automaton (DSCA) [127].

What's new with the Data and Signals Cellular Automaton (DSCA) ?

A look at figure 4.16(a) allows the calculation of the number of state variables involved in the molecule (each sequential register being represented by a small triangle), i.e. 16 for the stack ($GA$3:0, $GB$3:0, $PA$3:0, $PB$3:0) and three for the control signals of the input multiplexer ($I$2:0), which amounts to a total of 19. Therefore, the number of possible states is 2^{19}. Thanks to our methodology, i.e. separating the molecule into a processing unit and a control unit, we do not need to carry out the whole state table with 2^{19} rules.

- 16 variables ($GA$3:0, $GB$3:0, $PA$3:0, $PB$3:0) are data variables, required for transferring or storing the flags and molcodes of the original genome.
- Three variables ($I$2:0) are control variables, required for coding the five states of the graph in figure 4.16(d) and for controlling the different priorities.

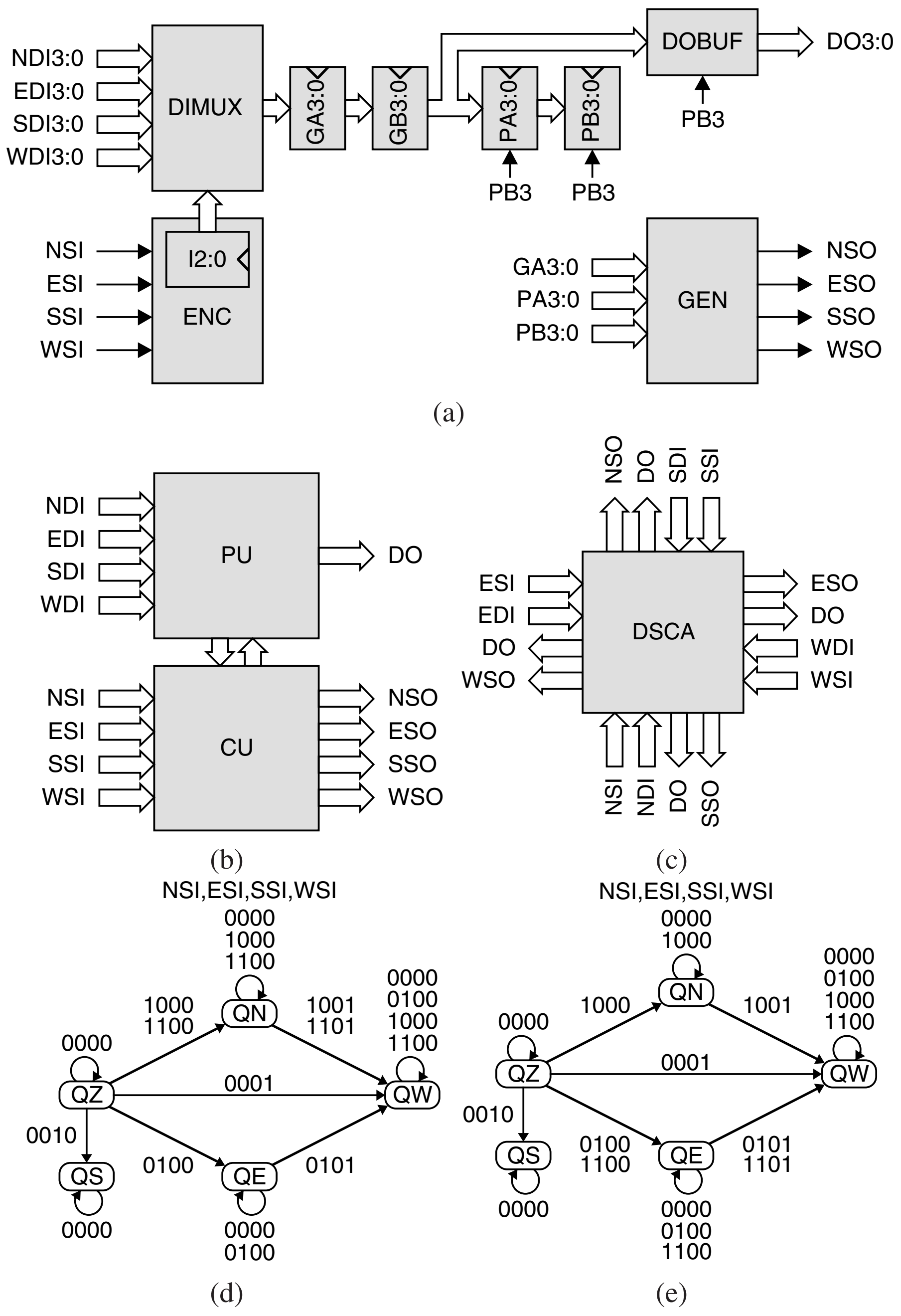


Figure 4.16: *A possible implementation of the basic molecule as a novel Data and Signals Cellular Automaton (DSCA). (a) Detailed architecture. (b) Macroscopic representation made up of a processing unit (PU) and a control unit (CU). (c) Macroscopic representation of the DSCA. (d) State graph of the finite state machine ENC. (e) Modified graph of the finite state machine ENC.*

The information containing all the characteristics of the self-replicating loop (height, width, changes of direction, useful information) is entirely included in the genome (flags and molcodes), which is easily programmable by the user: it constitutes the data part of the 19 variables.

The only information needed for controlling our DSCA is used for priorities calculation. Any change of specifications will then necessitate a transformation of the graph of figure 4.16(d). As an example, if we wish to build our multicellular organism row by row (and not column by column as in figure 4.15), we would have to start with the modified graph of figure 4.16(e).

4.2.3 Generalization and design methodology

Non minimal loops

The self-replicating loops in figure 4.17 are two examples of non minimal loops. Note that the molcode data can be directly used to display some useful information, such as in the example on page 69, or can be used indirectly as a configuration string able to control a programmable device such as a Field Programmable Gate Array (FPGA) (for such an application, see [124]).

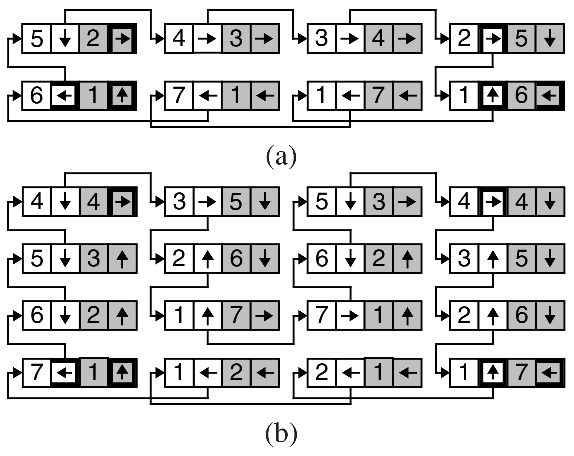


Figure 4.17: *Two examples of non minimal self-replicating loops. (a) A* $4 \times 2 = 8$ *molecule loop (*$\Delta tn = 20$, $\Delta te = 28$, $\Delta tc = 32$*). (b) A* $4 \times 4 = 16$ *molecule loop (*$\Delta tn = 40$, $\Delta te = 60$, $\Delta tc = 64$*).*

If C is the number of columns in the cell and R its number of rows, it is easy to derive the following relationships.

- The total number of the molecules M in a cell is:

$$M = C.R \tag{4.7}$$

- The total number T of hexadecimal characters in a cell is therefore:

$$T = 4.C.R \tag{4.8}$$

while the length L of the artificial genome is half the value of T, i.e.:

$$L = 2.C.R \tag{4.9}$$

The period of a cell, i.e. the time needed for a complete circulation of the genome, is equal to L time steps. A careful examination of the new self-replicating loop obtains the following relationships defining the different numbers of time steps Δtn (first northward outer path), Δte (first eastward outer path), and Δtc (inner path closing the loop):

$$\Delta tn = L + 2.R = 2.(C + 1).R \tag{4.10}$$

$$\Delta te = 3.L = 12.R \quad \text{if } C = 2 \tag{4.11}$$

$$\Delta te = 2.L - 2.(C - 2) = 4.C.R - 2.(C - 2) \quad \text{if } C > 2 \tag{4.12}$$

$$\Delta tc = T = 2.L = 4.C.R \tag{4.13}$$

The LSL acronym design example

In [128], Tempesti has already shown how to embed the acronym "LSL" (for Logic Systems Laboratory) into a self-replicating loop implemented on a classic cellular automaton. Thanks to a "cut-and-try" methodology and a powerful simulator, he was able to carry out the painful derivation of over ten thousand rules for the basic cell.

Unlike Tempesti's heuristic method, we will show that the same example can be designed in a straightforward and systematic way thanks to the use of our new DSCA associated to the Tom Thumb algorithm.

The "LSL" acronym is first represented in a rectangular array of 12 columns by 6 rows (Fig. 4.18(a)). While the number of rows is not important, the number of columns should be even in order to properly close the loop (Fig. 4.18(b)). The cell is therefore made up of $12 \times 6 = 72$ molecules connected according to the pattern in figure 4.18(b): bottom-up in the odd columns, top-down in the even columns, with the lower row reserved for closing the loop. It is then possible to define all the flags in the rightmost memory position of each molecule (grey characters in figure 4.18(b)) without forgetting the branch activation and north connection flag in the lower molecule of the first column, the north branch and east connection flag in the upper molecule of the first column, and the east branch and west connection flag in the lower molecule of the last column.

Among the 72 molecules, 25 are used to display the three letters "L", "S" and "L", and are given the character "2" as molcode (black data in figures 4.18(a) and 4.18(b)), while 47 are blank (molcode "1").

The detailed information of the final genome, i.e. $72 \times 2 = 144$ hexadecimal characters (Fig. 4.18(c)), is derived by reading clockwise the fixed characters (black and grey characters in figure 4.18(b)) of the whole loop, starting with the lower molecule of the first column.

Lastly, it was possible to embed the basic molecule of figure 4.16(a)in each of the 4'000 field-programmable gate arrays of the BioWall and to show the rather spectacular self-replication of our original cell (equivalent to a unicellular artificial organism), the "LSL" acronym, in both vertical and horizontal directions (Fig. 4.18(d))[2].

[2]In this experiment we used only half of the original BioWall, i.e. 2'000 molecules instead of 4'000.

The LSL acronym design example can be easily generalized to produce the following algorithm:

1. Divide the given problem in a rectangular array of C columns by R rows. While the number of rows R is unimportant, the number of columns C should be even in order to properly close the loop.

2. Define all the flags in the rightmost memory position of each molecule according to the following patterns: bottom-up in the odd columns and top-down in the even columns, with the lower row reserved for closing the loop.

3. Complete the path by adding the branch activation and north connection flag (C) in the rightmost memory position of the lower molecule of the first column, the north branch and east connection flag (D) in the rightmost memory position of the upper molecule of the first column, and the east branch and west connection flag (E) in the rightmost memory position of the lower molecule of the last column, in order to trigger the two daughter loops northwards and eastwards respectively.

4. According to the original specifications, complete all the molcode data in the second rightmost memory position of each molecule. These molcode data constitute the phenotypic information of the artificial cell.

5. The detailed information of the final genome, i.e. the genotypic information of the artificial cell, is derived by reading clockwise along the original path the fixed characters of the whole loop, i.e. the two rightmost characters of each molecule, starting with the lower molecule of the first column. The genotypic information, or artificial genome, is used as the configuration string of the artificial cell and will eventually take place in the two leftmost memory positions of each molecule.

Classical cellular automaton versus Data and Signals Cellular Automaton (DSCA)

Coming back to Tempesti's self replicating loop [128], we can now point out the major differences between his method and our new approach.

Tempesti used a classical cellular automaton (CA). With its self-replicating mechanism, the "LSL" acronym is entirely wired inside the CA, by means of more than a thousand rules, written thanks to a heuristic "cut-and-try" methodology. Even a slight modification of the original specifications could be very complex. It is impossible to demonstrate the property of universal construction.

With the Tom Thumb algorithm and its implementation as a data and signals cellular automaton [127], all the description of the "LSL" acronym is part of an external program, the artificial genome, simply flowing through the processing units of the DSCA. The design is straightforward, and the modifications are immediate. Changes inside the DSCA are only necessary for modifying the priorities which regulate the growth of the successive self-replicating loops.

Universal construction

In his original contribution [136], von Neumann defined construction (or constructibility) as the capability of constructing, i.e. assembling and building an automaton from appropriately defined "raw materials" using another automaton, the constructor. More precisely, the constructor, a two-dimensional automaton, is able to build in the two-dimensional array defined by von Neumann a specimen of another automaton described by a one-dimensional string of characters (the artificial genome) stored into the tape of the constructor.

According to von Neumann [136], a constructor is endowed with universal construction if it is able to construct every other automaton, i.e. an automaton of any dimension. This concept is pointed out by Freitas [116], where construction universality implies the ability to manufacture any of the finitely sized machines which can be formed from specific kinds of parts, given a finite number of different kinds of parts but an indefinitely large supply of parts of each kind.

If we assume, firstly the existence of an array, as large as desired, of molecules such as that described in figure 4.16(a) and secondly the existence of a string of characters, as large as desired (the artificial genome), then we are able to construct a computing machine of any dimensions into the array. Remember that the molcode data M, limited to the range $1 - 7$, may be directly used, as in the previous example, for displaying the given specifications or may configure any kind of field-programmable gate array aimed at defining a more complex digital architecture. There are only two restrictions involved in our actual implementation.

- The number of rows and/or columns should be even, in order to properly close the loop.
- For any artificial organism characterized by a molcode alphabet greater than $1 - 7$, we would be led to slightly modify the architecture of the actual molecule (Fig. 4.16(a)) and either use a deeper stack (with an even number of registers: 4, 6, 8, ...) or use larger registers (with more than 4 bits). For a flag alphabet greater than 8,...,F (particularly for addressing the 3-dimension case), larger registers would also be required.

If the two conditions are met, we can insert into an array of molecules any array of boolean (octal, hexadecimal) values and observe the self-replication of the original pattern.

On the other hand, it has already been shown that a universal Turing machine may be included in a regular array of identical cells [106], themselves decomposed and implemented onto a regular array of molecules. Our new loop with universal construction can therefore verify *universal computation*, thus meeting the two basic properties of the historical self-replicating cellular automaton designed by von Neumann [136], i.e. *universal construction and computation.*

4.2.4 Tom Thumb conclusion

Several years before the publication of the historic paper by Crick and Watson [139] revealing the existence and the detailed architecture of the Deoxyribonucleic Acid

(DNA) double helix, von Neumann was already able to point out that a self-replicating machine necessitated the existence of a one-dimensional description, the genome, and a universal constructor able to both interpret (translation process) and copy (transcription process) the genome in order to produce a valid daughter organism. Self-replication will allow not only division of a mother cell (artificial or living) into two daughter cells, but also the growth and repair of a complete organism. Self-replication is now considered as a central mechanism indispensable for those circuits which will be implemented in the nascent field of nanotechnologies [107, 40].

A first field of application of our new self-replicating loops with universal construction is quite naturally the study of all self-replicating automata.

A second, and possibly more important field of application is Embryonics, where artificial multicellular organisms are based on the growth of a cluster of cells, themselves produced by cell division [74, 76].

A major by-product of this research is the introduction of a new kind of cellular automaton, the Data and Signals Cellular Automaton (DSCA) [127], divided into a processing and a control unit, which allows for a systematic and straightforward design methodology which is lacking at the moment.

Other avenues to explore are the evolution of such loops and/or their capability of carrying out massive parallel computation [25].

4.3 Conclusion

This chapter details two different applications: Game of Life shows how easily a classic cellular automaton can be implemented on the BioWall, while the Tom Thumb application focuses on research based upon a new kind of CA, the Data and Signals Cellular Automaton (DSCA). This last application, while highlighting the BioWall's capabilities as a cellular computer, uses the maximum capacities of the FPGA. Improving of the Tom Thumb algorithm by adding self-repair concepts is thus impossible within the present BioWall hardware configuration. Some other limitations exist for the development of new applications. The POETIC tissue [133], for example, is not compatible with the BioWall architecture: the number of FPGA logic gates is not sufficient, nor is the number of lines between each molecule. Because of these limitations, the POETIC tissue was simulated using a computer, which has the big disadvantage of hindering the interactions and the parallel computation allowed by the BioWall, resulting in very slow tissue simulation timings (less than 10 simulated clock steps per second) even with a recent computer. These drawbacks will all be avoided with the development of the BioTissue (cf. chapter 7), which is smaller than the BioWall in terms of size, but represents a much more powerful cellular computing system.

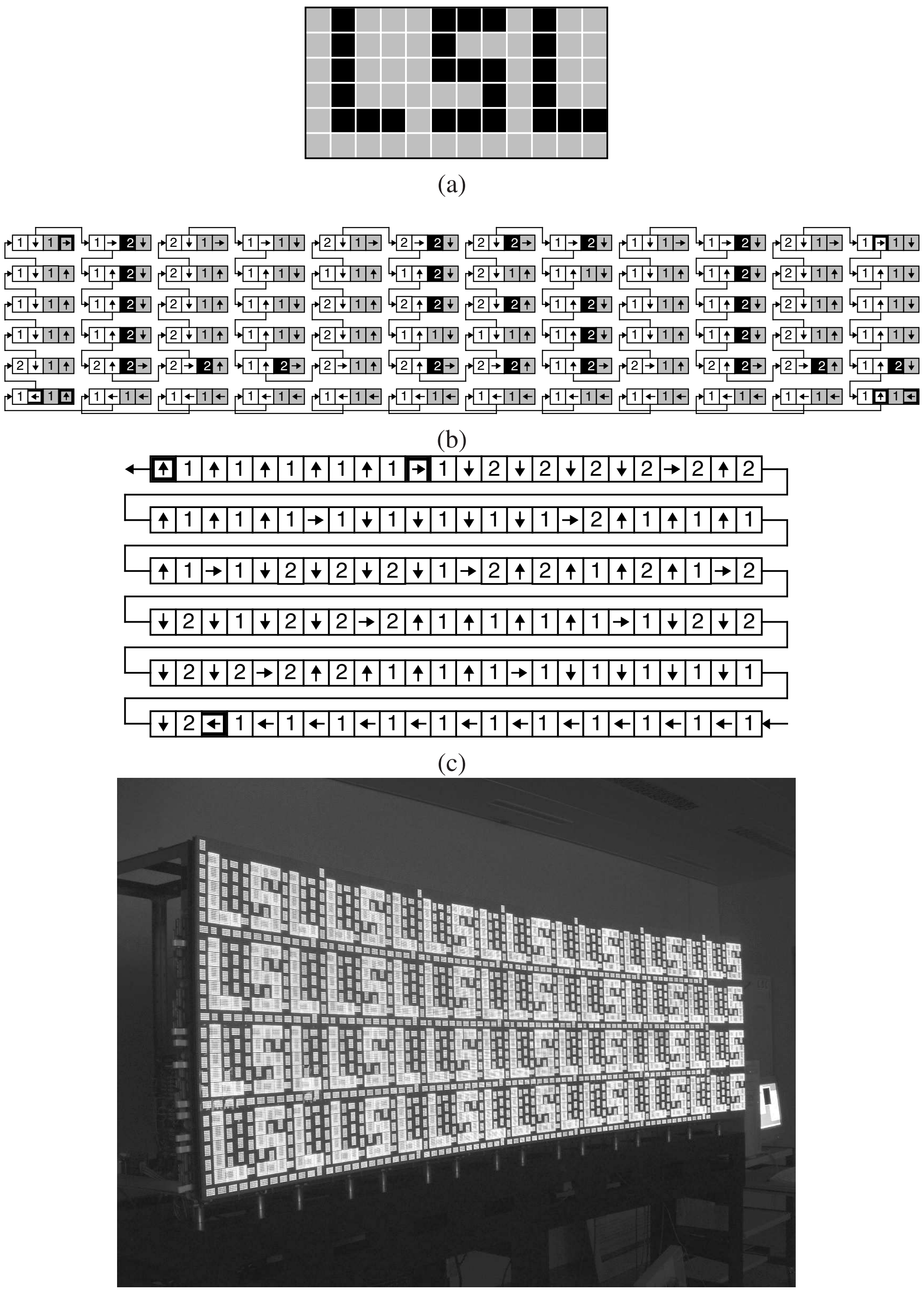


Figure 4.18: *Self-replication of the "LSL" acronym. (a) Original specifications (LSL = Logic Systems Laboratory). (b) The* $12 \times 6 = 72$ *molecules of the basic cell. (c) Genome. (d) BioWall implementation displaying both the genotypic path and the phenotypic shape (Photograph by E. Petraglio).*

Chapter 5

BioCube hardware description

DUE to the permanent evolution of technologies, the transistor size of circuits is getting smaller while at the same time the complexity of electronic circuits increases. Current industrial process technology is based on a 65 nm grid size for transistors, and is evolving towards ever smaller nanometer dimensions. In a few years, the limits of the current manufacturing process will be reached, and new solutions will need to be found to continue improving component performance. While all current micro-electronic layout methodologies still place transistors on a 2D grid, we are seeing the beginning of circuit construction stacking several conventional dies (cf. § 2.2.1 on page 15). It seems likely that 3D design methodologies will replace 2D methodologies in the near future, and that nanotechnologies will govern the construction of the 3D chips (cf. section 2.3.2).

The BioWall is a large 2D cellular machine made up of 4'000 identical molecules. However, this machine is not able to simulate or study topologies such as 3D cellular structures.

Our new machine, called BioCube[1], is closer to the biological world, where organisms are an assemblage of cells arranged in a 3D architecture. We will build the BioCube using the BioWall architecture, but expanded to a 3D structure. Each molecule of the BioCube is still composed of a Field Programmable Gate Array (FPGA), but differs from the BioWall by its topology and its display.

In the next section, we describe the structure of this new cellular machine. Section 5.2 presents all the electronic components that makes up the BioCube, and the improvements made to the BioWall architecture. In section 5.3 we explain how the tasks of configuring and differentiation of the FPGAs' contents is performed.

5.1 Specifications and global overview

One of the BioCube's specifications was to extend the 2D BioWall structure to a 3D architecture, which could act as a hardware simulator for 3D cellular machines, such as machines built on the nanometer scale.

Similarly to the BioWall, the BioCube (Fig. 5.1) is designed as a reconfigurable

[1]This name comes from the contraction of bio and cube, a cube-shaped bio-inspired machine.

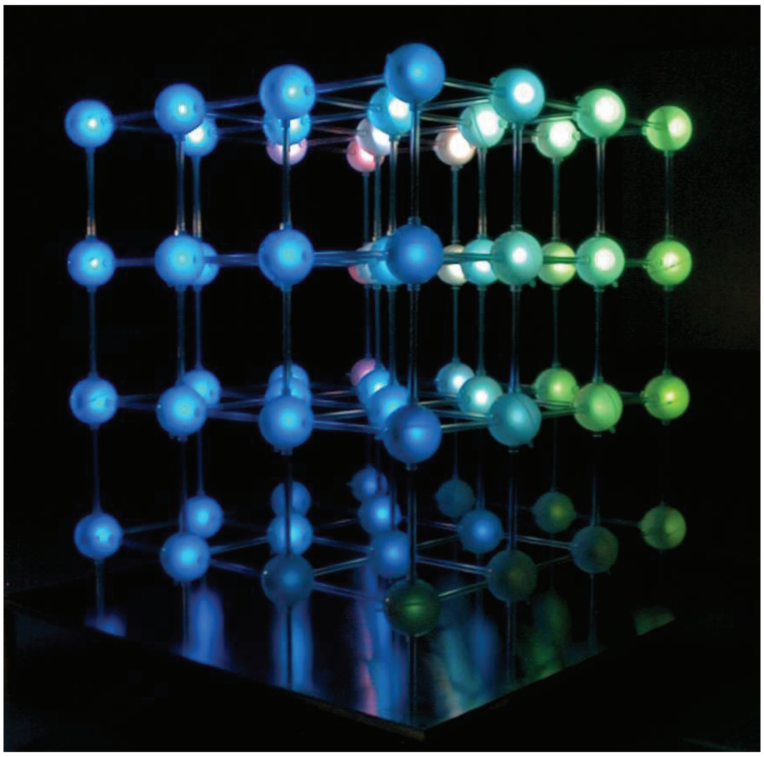


Figure 5.1: *The BioCube.*

3D computing structure able to interact with its environment by means of touch-sensitive elements coupled with LED displays. Its structure is inspired by cellular organisms, which are made up of 3D cell arrays.

The BioWall showed us that interaction is really crucial, since each BioWall FPGA can display its logical status on a LED display and receives stimuli from the sensitive membrane glued on the display. For this reason, the BioCube will be built using an original structure inspired by the atomic world, where crystalline elements are represented by an array of spheres (the atoms) connected together with cylinder links (the atomic links): both displays and sensitive spheres will make interaction possible.

Each atom, a semi-transparent plastic sphere, contains a board with all the electronic components. Two truecolor Light-Emitting Diodes (LEDs) light up the whole surface of the sphere. Thanks to a capacitive sensor, the sphere surface can detect a finger touch. Each sphere is connected to its neighbors by six links, with the exception of the spheres at the surface of the BioCube. These links are made of a small plastic tube. The whole structure stands on a metal place which provides the energy distribution through the columns. The sphere located at the $X, Y, Z = 0, 0, 0$ coordinate has an additional link with the controller board.

The BioCube contains 64 spheres, arranged as 4 layers of 4 rows by 4 columns, measuring $60 \times 60 \times 60$ cm.

Each sphere can be located inside the BioCube using a 3D Cartesian coordinate system where each sphere gets a unique X, Y, Z coordinate. We also chose the following nomenclature to represent interconnections between spheres: a positive X-axis

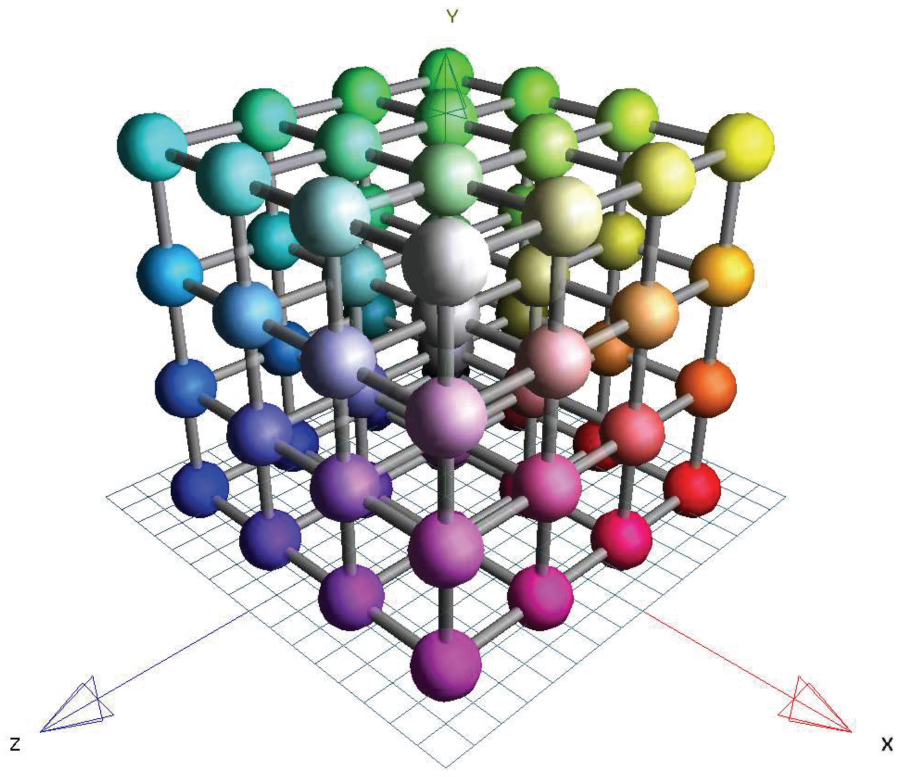


Figure 5.2: *BioCube simulator.*

points eastwards; a negative X goes westwards; a positive Z-axis southwards; a negative Z northwards; a positive Y-axis up; a negative down.

5.1.1 BioCube specifications

The BioCube is designed as a reconfigurable 3D computing structure capable of interacting with its environment by means of touch-sensitive elements coupled with LED displays. Its structure is inspired by cellular organisms, which are 3D arrays of cells.

The BioCube is built using a similar architecture to the BioWall. Thus it also contains a reconfigurable component (1) interconnected with its neighbors (2), able to display the operations of the system (3) and to interact with several users (4). The whole cube is controlled by a Personal Computer (PC) (5). We decided to add some additional features, highly desirable for machines built on the nanometer scale: asynchronous computation (6) and configuration differentiation for each FPGA (7).

1. Because the BioWall machine is made up of a large array of FPGAs, it was important to choose an inexpensive component. Since the BioCube is a machine composed of only a small number of spheres, we chose a more powerful FPGA than the SPARTAN® FPGA of the BioWall. We chose the XCS3S200 SPARTAN®3 from Xilinx®, which is attractive for its low price, and is still more powerful than the first version of the SPARTAN® used in

the BioWall. The main advantages of this FPGA are that it has a higher number of logic gates and a more powerful architecture, and includes several internal modules like Random Access Memory (RAM) blocks, Digital Clock Manager (DCM), hardware multipliers...

2. The BioCube is built on a mesh-network where each FPGA is connected by a von Neumann neighborhood to its six adjacent FPGAs. To simplify the construction of the BioCube, and for aesthetic reasons, we limited the number of wires between each sphere to a 10-line flat wire. Thus, communications between all FPGAs are multiplexed over a high-speed serial line.

3. Each FPGA can control a Red Green Blue (RGB) LED serving an output element designed to illuminate the sphere uniformly with one of 16'777'216 colors (8 bits for each basic color).

4. A proximity sensor acting as an input element is able to detect a finger touching any part of the plastic bubble. This information is then transmitted to the FPGA.

5. The whole cube is controlled by a dedicated controller. The latter supervises and synchronize the operations of the 64 units. Dedicated software running on a PC can interact with the controller and transmits and receives data from all the spheres.

6. In the BioWall, clock signals were transmitted by the main global lines. To avoid this in the BioCube, each sphere has a local oscillator, resulting in a Globally Asynchronous, Locally Synchronous (GALS) architecture for the whole BioCube.

7. We have already shown that having the same configuration for all the FPGAs was a limitation in the BioWall; we therefore added the possibility of having different configurations running at the same time in different FPGAs of the BioCube.

5.1.2 BioCube overview

Before constructing the BioCube, we developed a simulator which helped us to find the best ratio for sphere size, the cylinder diameter and the cylinder size. Figure 5.2 shows the output window of the simulator representing the RGB cube [13] with a different color (or state) for each sphere.

We named the elementary units of the BioCube, i.e. the spheres or atoms, *BBall*. Each *BBall* is made of a Printed Circuit Board (PCB) contained in a plastic sphere. The scalable structure allows several topologies, but for cost and time reasons, we chose to build our BioCube with $4 \times 4 \times 4$ *BBall*s.

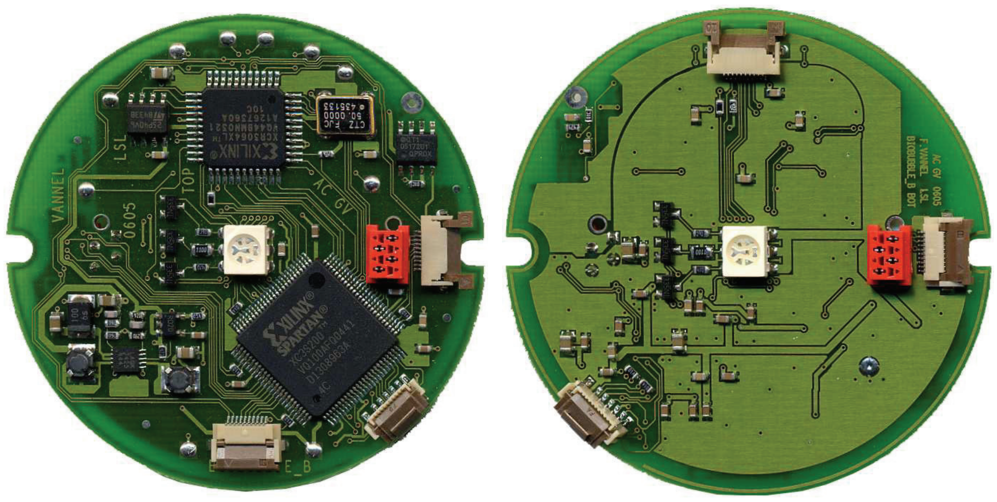
The whole system is controlled by a computer. Simple software and a specific interface (*BioCubeCtrl*) allow new FPGA configurations to be sent from the computer to the BioCube's *BBall*s. These configurations can be saved in FLASH memory slots. A very interesting possibility is to have different FPGA configurations running in each sphere. With this option we can, for example, configure all the *BBall*s in the interior of the cube with one application, and the external *BBall*s with another configuration.

This external interface also fulfils a second task, allowing synchronization of all the cube's *BBall*s with a specific synchronizing signal.

The entire BioCube consists of about 13 million reconfigurable gates. This number can be compared with the 40 million gates of the BioWall.

5.2 BioCube board description

5.2.1 The *BBall* board



(a) Topside view: the RGB LED is located in the center of the picture; all the logic components are located on this face: FPGA, CPLD, FLASH and also the power convertor on the bottom-left edge.

(b) Underside view: a second RGB LED is placed in the center; the surrounding connectors link the *BBall* to its neighbors; the red connector is the power supply input.

Figure 5.3: *Pictures of the* BBall *board.*

The *BBall* board (Fig. 5.3) constitutes the main unit of the BioCube platform and houses all the computational components within which the applications will run.

This board is articulated around an FPGA coupled to a Complex Programmable Logic Device (CPLD) which has a FLASH memory for storing the FPGA configuration. The output display element is composed of two LEDs in the same location on the two sides of the PCB. A capacitive proximity sensor [113] allows an user to interact with the system (Fig. 5.4). All these components are soldered on a round (60 mm diameter) 2-layer PCB.

It was decided at the beginning that, unlike the BioWall, the BioCube would be designed with rather limited dimensions. The low price argument was therefore not so crucial, and we decided to choose a new more powerful FPGA (as mentioned in 5.1.1).

Each *BBall* consists of:

- An input element. We used a proximity sensor which enables each unit to detect a finger touching any part of the plastic bubble. The small line circling the PCB in figure 5.3(b) is the sensitive electrode which is linked to a specific electronic

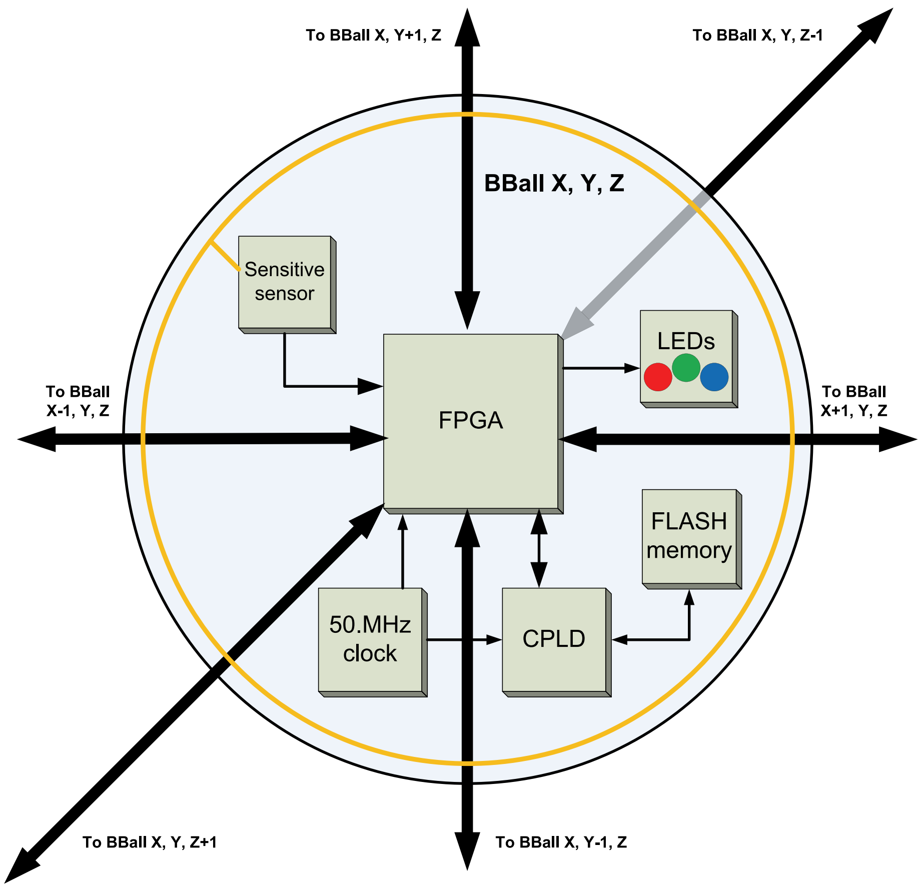


Figure 5.4: *Architecture of the* BBall *board.*

circuit. A logic signal toggles when a human presence is detected near this electrode wire (the sensitivity is of about 5-10 mm on our board).

- An output element. A tri-color LED capable of lighting uniformly the sphere with one of 16’777’216 colors (8 bits for each basic color). Whereas each cell of the BioWall drives a display of 8×8 pixels constituted of two LEDs (i.e. four colors), a single pixel of the BioCube can display one of millions of colors. Thus several states can be displayed by changing the color of a sphere.

- A programmable computing element, an FPGA (SPARTAN®3 XC3S200 Xilinx® FPGA[2]) with 200’000 equivalent logic gates. Some of the interesting features included are the 18x18 multipliers, the several 18 Kb Block RAM and the four digital clock managers (DCM) which make it possible to multiply the clock frequency.

- A FLASH memory and a CPLD. The FLASH memory can save up to four differ-

[2]The exact model is the XC3S200-VQ100C.

ent FPGA configurations. The CPLD manages the configuration protocol of the FPGA and can configure the FPGA with one of the four configurations stored in the FLASH memory.

- A local clock at 50MHz. The FPGA uses this clock as an input, and can multiply it with one internal DCM up to 300 MHz.
- I/O with neighboring units. Each link uses a 10-wire cable. This choice was made with the aim of having small cable routing from sphere to sphere. Communications use serial data transfers. Signals need to be multiplexed and demultiplexed inside each FPGA before going to these serials links.

Each FPGA possesses various connections with its neighbors in other *BBall*s. Beige connectors surrounding the board and visible in figure 5.3 allow direct connection of adjacent *BBall*s. Signals and communication possibilities available between SPARTAN®3 components are described in table 5.1.

Type	Line number	Description
Point to point links	2× (1 LVDS pair + 1 CMOS wire)	One output communication with the next FPGA and one input communication; each link is constituted of a Low Voltage Differential Signaling (LVDS) pair plus a single Complementary Metal-Oxide-Semiconductor (CMOS) line allowing a bandwidth of 400 Mb in each direction.
Global control lines	2	CMOS signaling connecting all FPGAs together with the external control module. The status of these lines is set by the *BioCubeCtrl* box.
GND	2	Ground line for setting the same reference voltage between each *BBall*s.

Table 5.1: BBall *FPGA I/O summary.*

Display management

Two RGB LEDs, each located on a face of the *BBall* PCB, light up the sphere. The FPGA drives each LED-color (red, green, blue) directly. A logic signal set to “1”, lights up the corresponding LED, whereas a logic “0” turns it off.

The RGB color model is an additive model in which red, green, and blue are combined in various ways to produce other colors. We know that the light output of a LED is dependent on the current flowing through it [66]. However, controlling brightness using a variable current source is not a recommended method because a very precise current source, would be needed, and this would be complex to build. The preferred technique for brightness control is through Pulse-width modulation (PWM) [10]: a square wave whose duty cycle is modulated should be used resulting in the variation of the average value of the waveform, which controls the amount of power sent to the LED. (Fig. 5.5)

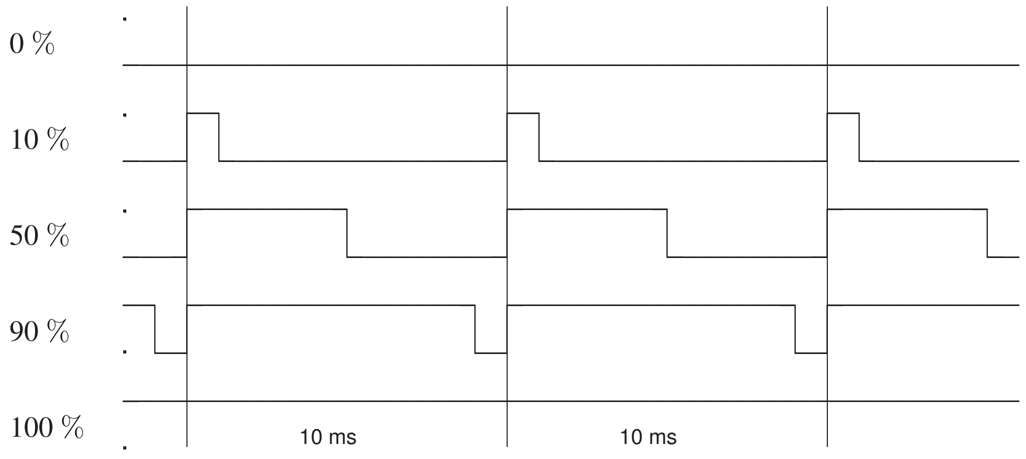


Figure 5.5: *LED intensity controlled by a PWM signal.*

Persistence of vision is the human visual phenomenon that allows video images to be viewed without flicker [28]. When the human visual system is presented with an image, that image continues to be perceived even though it is no longer in the visual field of the observer, albeit for a short time. The rate at which the frames are refreshed is termed the refresh frequency. If the frequency is above a certain threshold frequency, the observer will not notice any flickering. For LED displays, a refresh rate of above 60 Hz is recommended [4]. For our PWM control module implemented into the FPGA we chose a refresh frequency of 100 Hz to avoid any flickering effects (Fig. 5.5). The pulse width is controlled by a 10 bit value, where the value 0 means LED off, and the value 1023 means full luminosity output. The intermediate values set the pulse width proportionally to the value selected. The range between the values 0 and 1023 must give all the intermediate gradation levels in a linear way. Unfortunately, as in the majority of all luminary devices, the transfer function between the electrical and optical components of the display system is non-linear. If this non-linearity is not compensated, high brightness regions are expanded and dim regions are compressed [4]. Thus we added into the FPGA a gamma [102] correction lookup table (8-bit wide input, 10-bit wide output), which compensates the non-linearity of the LED display, and results in a linear transfer function. We adjusted the contents of this Lookup Table (LUT) based on a personal subjective measurement.

Finally to control the sphere display, a VHDL module was implemented into the FPGA with three 8-bit buses. Each module controls the luminosity intensity of the red, green and blue LEDs in a linear way.

5.2.2 The BioCube: an assemblage of *BBall* boards

We decided to build our machine as a cubic arrangement of 64 *BBall*s (4 layers of 4 by 4 spheres). Each *BBall* board is placed into a plastic sphere from which six pipe connections allow connections to other *BBall*s. These plastic pipes give mechanical solidity to the BioCube structure, and can carry electric wires inside then. Figure 5.1 shows the BioCube running: since the links are themselves not lighted, they are not easily visible in this illustration, but can be spotted in the synthesis image of figure 5.2. This image is a snapshot from the BioCube simulator, software specially developed for simulating the BioCube and also helping us to define the construction ratios of the machine. Some parameters can be adapted to change the number of spheres on each side of the cube, the diameter of each sphere, the length and the diameter of the pipes. Since we were looking for ratios allowing the internals layer of the BioCube to be seen without having too big a gap between spheres, we chose a ratio of $1/2$ between the diameter of the sphere and the length of the tube with the help of this simulator. For the BioCube we thus have the following dimensions:

- Sphere diameter = 6 cm.
- Tube length = 12 cm.
- Tube diameter = 1 cm.

The tube between each *BBall* has an inside diameter allowing the insertion of a flat cable made of 10 thin wires and another cable made of four wires to carry the 5 V power supply. Each *BBall* is connected to the others in all directions using the signal flat wire. The power supply is only carried by the vertical links, i.e. connections of columns composed of four *BBall*s.

The whole BioCube structure is placed on a dedicated table, the bottom layer of the cube held above the table-top by shorter 6 cm pipes. An electric distribution network is fixed beneath this support, allowing each column of four *BBall*s to be powered with a regulated 5 V supply. An external AC/DC power supply provides energy to the whole BioCube, which can consume up to 125 W when a complex configuration is running inside the FPGA and when all the *BBall*s are simultaneously lit with the color white.

Since the BioCube is not an autonomous machine, it needs to be controlled by an external device. The *BBall* located at coordinates $X, Y, Z = 0, 0, 0$ is connected through a flat wire to a controller module called *BioCubeCtrl*. This electronic board allows a PC running dedicated software to interface with the BioCube. Details of this control system will be given in paragraph 5.2.3.

Paragraph 5.2.1 detailed the structure and the performances of the *BBall* board. We will now explain the method used to communicate between *BBall* modules.

Cellular communications

Since the BioCube's main purpose is the study of conventional cellular systems and unconventional systems like regular nanometer structures, we needed simple communication protocols between *BBall*s. The most appropriate and simplest intercommunication system is a simple bus made up of several wires linking each *BBall* together.

The BioWall is already constructed in this manner, but as we saw in chapter 4, the predefined number of wires was not large enough for more complex applications. We wanted to avoid this limitation, but the idea of a bus made up of dozens of wires was not a practical solution, requiring large flat cables, incompatible with the tiny links between spheres.

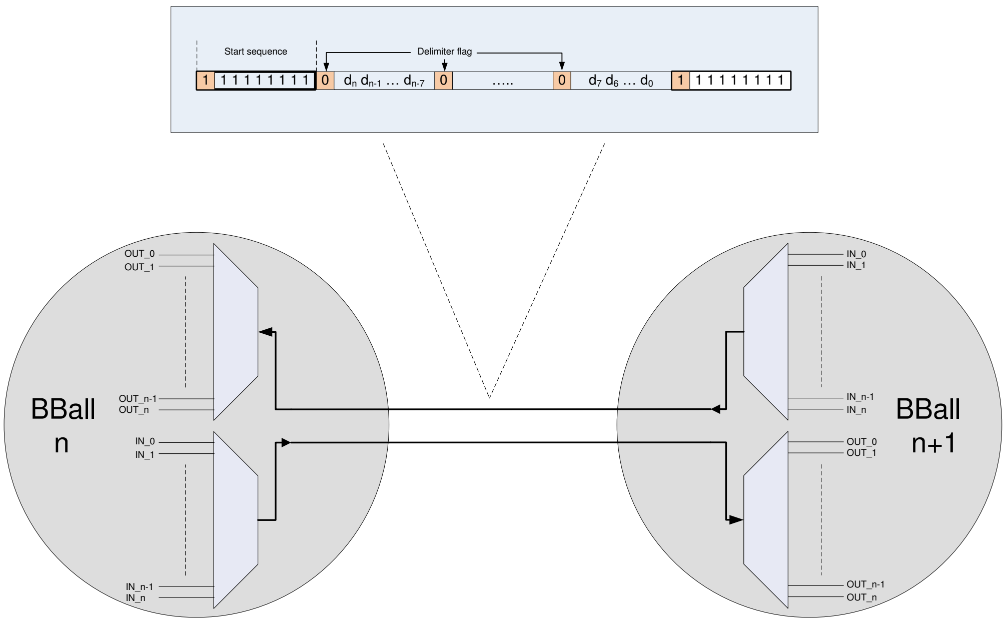


Figure 5.6: *Detail of the* BBall *inter-links.*

The chosen solution (Fig. 5.6), based on Time Division Multiplexing (TDM) [7], is a type of digital multiplexing in which two or more signals are transferred apparently simultaneously as sub-channels in one communication channel, but in reality taking turns on the channel. TDM avoids having a large number of cables, while still offering the advantage of having big busses interconnecting FPGAs. We have only three wires in each direction between two adjacent FPGAs. The first wire is the clock of the transmitting FPGA, and the two other nets are a differential line transmitting the data, which can sustain a bandwidth of 400 MHz. On the input side of our VHDL module, we have as many channels as we want. The module has a bus on the input side whose size is defined by the application (size should be a multiple of eight and not exceed 256), and which encodes the inputs and serializes them on a single line. The receiving FPGA decrypts the stream and reconstructs the bus states on the output module. This process is done in the following way, as illustrated in figure 5.6:

1. A start of sequence header composed of nine consecutive bits set to 1 is sent ont the output serial bus.

2. The next transmitted bit is the delimiter flag always set to 0.

3. The eight Most Significant Bit (MSB) input bus are scanned and serialized.

4. Go to step 2 until eight Least Significant Bit (LSB) input buses have not been transmitted.
5. Repeat from step 1.

The VHDL module inside the receiving FPGA analyzes the incoming stream and reconstructs the state of the bus in a similar way to the module used for the transmission.

The multiplexing and demultiplexing protocol allows the state of a large bus to be transmitted over a single line from one *BBall* to another. The weakness of this method is the need for several clock steps for the state of the bus to be replicated to the other FPGA. This system is perfectly suited to our architecture, since it allows all kinds of cellular applications to run easily inside the BioCube while physical connections between each *BBall* use only a small 10-wires cable. The operating frequency for these communications can even be high, and frequencies of some megahertz are possible.

The time in number of clock steps to parse and serialize all the bits of the input bus is defined as follow:

$$9 + \frac{9}{8}n \tag{5.1}$$

where n is the bus size which should be a multiple of 8.

By adding delays (12 in our current implementation) for processing the data in the multiplexing and demultiplexing modules, we obtain a maximum frequency in MHz of:

$$\frac{400}{9 + \frac{9}{8}n + 12} \tag{5.2}$$

This value is based on the maximum bandwidth sustainable by the differential lines: 400 MHz.

For the state of the bus to transmit correctly from an FPGA to its neighbor, the signals on this bus must not change at a frequency higher than the one computed by the previous formula.

For example, with a 32-bit wide bus (which is bigger than that of the BioWall), we can have a system working at 7 MHz, which is 35 times faster than on the BioWall. This frequency is maximum for interfaces, but not for core operations where speeds can be up to 100 MHz.

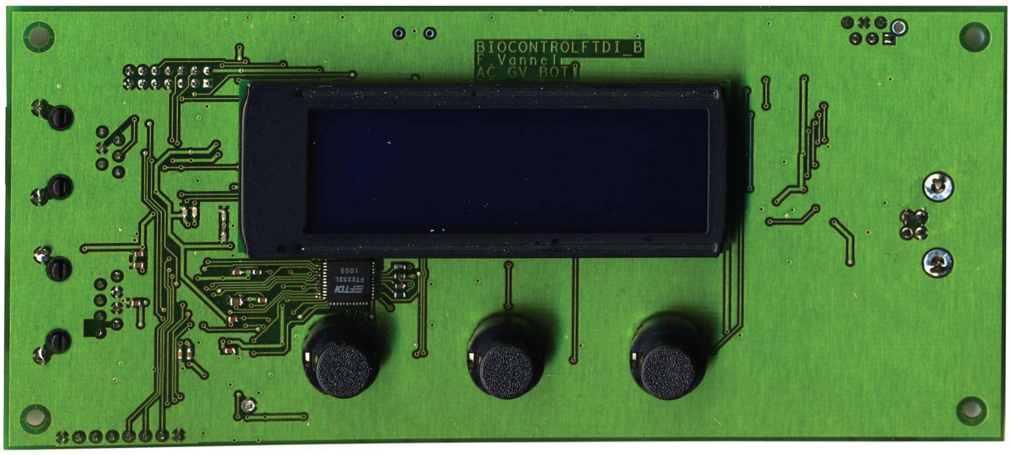
The module described here only illustrates connections between two *BBall*s, but there are five similar links with the other cardinally located FPGAs.

5.2.3 The *BioCubeCtrl*

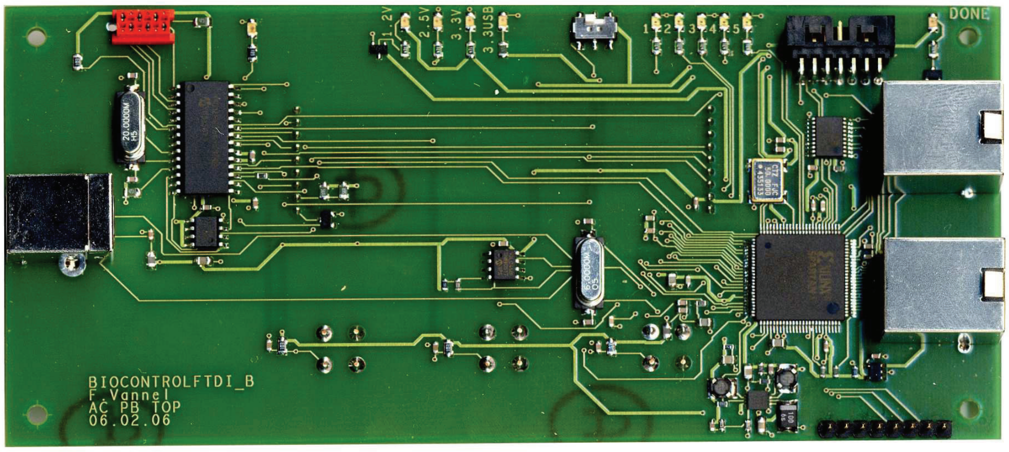
The *BioCubeCtrl* board (Fig. 5.7) is responsible for interfacing with and controlling the BioCube from a PC. Simple software running on the PC helps to exploit the possibilities of the BioCube.

The tasks executed by the *BioCubeCtrl* module and its software are:

- Transmission of the FPGA bitstream from the computer to the FLASH memory of all *BBall*s.



(a) Top view: LCD display with the three control buttons.



(b) Bottom view: USB connector on the left; connections with the BioCube are accomplished with the two connectors on the right; the FPGA controlling the whole system is on the right.

Figure 5.7: *Pictures of the* BioCubeCtrl *board.*

- Launch of the application.
- Global synchronization of the whole system, as for Cellular Automata (CA) applications.
- Reception of the states from the BioCube. This allows results to be gathered from the BioCube and transmitted to the PC where they can be stored or processed.

Global synchronization

In the current BioCube version, the SPARTAN®3 FPGAs run with their local 50.0 MHz clock. To synchronize all of the *BBall*s, as for cellular automaton applications, we added a synchronization system controlled by the *BioCubeCtrl* interface. A signal, labeled C_EN, is generated by the *BioCubeCtrl* board and synchronizes the state of each *BBall*. This signal acts as an enable signal where its high state means for the *BBall* FPGA: *compute*. The low state of C_EN means: *update the output ports with the computed data*. This signal can also slow down the computation speed to give the user time to view the computational results. This C_EN signal can thus be controlled from the 1 Hz frequency up to 1 MHz. A specific mode allows it to generate pulses only on request. Frequency speed and step-by-step mode are controlled through a software running on a PC or by the *BioCubeCtrl* module using its local interface. Figure 5.8 shows this signal and how it is correlated with the clock of an FPGA.

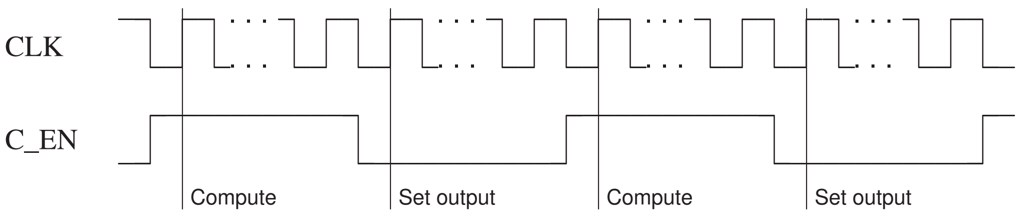


Figure 5.8: *BioCube synchronization: enable timing.*

In addition to the point to point connection between FPGAs, two global signals (G bus) are controlled by the *BioCubeCtrl*. Their purpose will be detailed in the next section.

5.3 Boot loader

Whereas the BioWall FPGAs were configured by the *BioBox* using dedicated lines, the BioCube does not have such configuration possibilities. This method would have had the big disadvantage of adding five more wires between each *BBall*, resulting in bigger and unaesthetic pipes linking the spheres. To solve this problem, the FPGA configuration can only be stored inside the FLASH memory. Thus, the CPLD will be in charge of configuring the FPGA using the configuration stored inside the FLASH memory.

This G control bus allows the state of all the *BBall*s to be changed at the same time, which is really important in order to have the same bus interface between each *BBall* at the same time.

G	boot loader	application
00	test mode	not used
01	not used	run mode
10	command mode	reset
11	enumeration mode	kill application and reload boot loader

Table 5.2: *G control signal meaning for the boot loader and the applications.*

The *BioCubeCtrl* can send commands to each *BBall* individually, or to all of them at the same time.

5.3.1 Enumeration

Since each *BBall* is identical in terms of components, we can not differentiate them inside the BioCube tissue. However such a feature is required by the boot loader and by some applications, which need to locate each *BBall* inside the tissue with its X,Y,Z coordinates. To perform this task, we added a dynamic enumerating process executed at the start of the boot loader configuration. Once the BioCube is started, each *BBall* configures the FPGA with the boot loader configuration. At the same time, the G[1:0] control bus should be set to 11, meaning that the enumeration task must be performed.

All the *BBall*s set their X,Y,Z identifier to {0,0,0}. They transmit their identifier, i.e. {0,0,0}, over North-, East- and Up-links in this situation. The *BBall*s connected on the other side of these links receive the identifier and add 1 to this value on the corresponding axis. For example, the next *BBall* connected on top of another *BBall*, will add the value 1 only to the Y coordinate (Bottom-Top axis). The *BBall*s on the West, South and Bottom sides don't have any connection with any *BBall*s in these directions, hence they will not receive any coordinate value, meaning that they keep the initial 0 value. As a result of these operations, all *BBall*s will increase their coordinates by 1 in the three X,Y,Z axes, respectively in the East-, North- and Up-directions. After a short time, all BioCube *BBall*s are identified with their X,Y,Z coordinates

A unique identifier (0 to 63) is also computed based on the X,Y,Z coordinates:

$$ID = 16 * Z + 4 * Y + X \tag{5.3}$$

$$with\ X, Y, Z = \{0, 1, 2, 3\}$$

This identifier is then stored inside a dedicated register of the CPLD where the value will remain until the power is switched off. Each application subsequently loaded can communicate with the CPLD to get the coordinates of the *BBall*s inside the BioCube.

The *BioCubeCtrl* can transmit dedicated commands to specific *BBall*s by addressing them with the previously computed identifier.

The identifier is needed primarily by the boot loader. In the next paragraph, we will look at how this coordinate value is used to send FPGA configurations from the *BioCubeCtrl* and store them in the dedicated FLASH memory.

5.3.2 Application launch

As we chose to avoid direct configuration of the FPGA, as for the BioWall, the configuration must be stored into the FLASH memory of the *BBall*. The FLASH memory has four storage slots that can each contain an FPGA configuration. The number three slot is by default reserved for the boot loader. Slots 0 to 2 are available for storing applications. At the power-on of the BioCube and once the CPLD of the *BBall* has completed its initial starting phase, the CPLD reads the contents of the number three slot of the FLASH and configures the FPGA with the bitstream stored into this slot. Once the FPGA has been configured, it owns the boot loader application and thus adds the control feature to the *BBall*.

The *BioCubeCtrl* can send a command requesting the boot loader to reconfigure the FPGA of its *BBall* with the application number 0,1 or 2. Once this command is sent, the boot loader transfers it to its CPLD which will erase the FPGA's bitstream and configure the FPGA with the requested application.

Since the boot loader commands can only be sent while the G[1:0] control bus is set to 10, this bus value means "reset" once the new application is configured inside the FPGA. The value of the G bus needs to be changed to 01 in order to start the application.

There are two methods to change the running FPGA configuration. The first one is performed by the application itself, which will request the CPLD to reconfigure the FPGA with another application stored into the FLASH. This method only applies to the *BBall* which initiated the process and can not be propagated to all the other *BBall*s. The second way, which is the most common, is based on the G control bus. Changing the state of the bus from 01 (running application mode) to 11 kills the application and reloads the boot loader to all the FPGAs of the BioCube at the same time. A request for loading a new application can then be performed by changing the G bus state to 10 (boot loader command mode). This is equivalent to the methodology explained at the beginning of this paragraph.

Since having only three applications in execution is too limiting, a configuration bitstream needs to be replaced by another when an application which is not yet available in one of the three application slots is called.

5.3.3 Application storage

During the fabrication of the *BBall* board, the boot loader application is stored inside the FLASH memory using a dedicated programming tool. The boot loader will be responsible for erasing application bitstreams and installing new ones into the three available memory slots.

The *BioCubeCtrl* module will firstly send a command to the boot loader of some or all of the *BBall*s in the BioCube, giving the order to erase a slot (number 0,1 or 2) of the FLASH memory. Once this task is complete, which can take a few seconds, another command initiates the writing of a new FPGA bitstream inside the FLASH memory. This is followed by a stream broadcast from the *BioCubeCtrl* to all the *BBall*s and which will be saved inside the FLASH memory. The whole process of decoding the serial stream, and state machine's handling of data storage inside the memory is executed by the boot loader.

Once the new configuration bitstream is stored in the FLASH, the boot loader reverts to command listening mode, and can, for example, start the process of storing another bitstream into another memory slot.

5.4 Conclusion

In this chapter, we described a new hardware platform designed to enable advances in 3D bio-inspired cellular computing. The BioCube is also a hardware simulator for cellular machines implemented at the nanometer scale. The versatility and scalability of the platform along with its potential parallel computational power offer very interesting prospects for application research, as we will see in the next chapter.

It is also interesting to notice, that the BioCube gave us some really nice artistic effects when switched on in the dark. We are not interested by these aesthetic features, but two similar machines have been built by other laboratories at the same time and are only used for their graphical (or artistic) performances. The Cubatron [32], an industrial project, is a $9 \times 9 \times 9$ cube where the color of each sphere can be controlled by a PC. The NOVA [88] project, which is an academic creation, is a huge construction made of $25 \times 25 \times 10$ spheres. It is exposed in the main hall of the Zürich central station and displays many different artistic performances. Both machines are passive displays whose spheres do not contain any processing capabilities.

Several other 3D displays exist as prototypes [85, 39] or even commercial products [80], but none of these have, in contrast to the BioCube, processing power contained in their structures.

Chapter 6

BioCube applications

WHEREAS the BioWall was a 2D cellular machine, the BioCube can be seen has the product of an evolution towards the third dimension. The main differences when compared with the BioWall reside in the physical aspect of the system and in the display capabilities, which are limited to one unique truecolor pixel (16,777,216 unique colors) for each FPGA. The isometric crystalline structure of the BioCube is perfectly suited for the study of Cellular Automata (CA) up to the third dimension.

This chapter will cover the evolution from 2D to 3D of the two BioWall applications mentioned in chapter 4. The first application implemented on the BioCube is the 3D Game of Life, with each *BBall* acting as a cell of the automaton. The second application extends the Tom Thumb algorithm to a 3D structure, thereby showing how a bio-inspired computing machine built at the nanometer scale could be programmed.

6.1 3D Game of Life

The 3D generalization of John Conway's Game of Life was investigated by Carter Bays [11]. In an M-dimensional Life automaton with a Moore neighborhood, a cell has $3^M - 1$ neighbors, i.e. 8 for 2D $(M = 2)$ and 26 for 3D $(M = 3)$. Therefore the 3D version has richer rules and structures than the 2D one. The difference with Conway's 2D version is in the rules defining the dead or alive states. In the 3D Game of Life, more variations are possible and although several of them have been studied by Carter Bays and Alexander Keewatin Dewdney [11, 37], no specific rules are commonly used for the 3D automaton. With the help of a java applet [1], we simulated a $4 \times 4 \times 4$ cube with different rules. Since the dimensions of our BioCube are quite small, we looked for rules that allow us to have a 3D Game of Life automaton with interesting visual animations, and initial patterns which don't converge too rapidly towards a static situation or death. Von Neumann's neighborhood, where each cell is only connected to its six 3D direct neighbors, gave us better results than Moore's neighborhood (with 26 neighbors) and was thus chosen for our application. The following rules fulfilled our expectations and we also found some initial patterns from which oscillator patterns emerged:

- If the number of living neighbors is too small (zero), the individual dies of isolation and its future state is “dead” (loneliness).
- If an individual has exactly one living neighbor, it conserves its current state (persistence).
- If an individual has two or three living neighbors, its future state is “alive” (reproduction).
- If an individual has too many living neighbors (four or more), it dies of overpopulation and its future state is “dead” (overcrowding).

6.1.1 Application operations

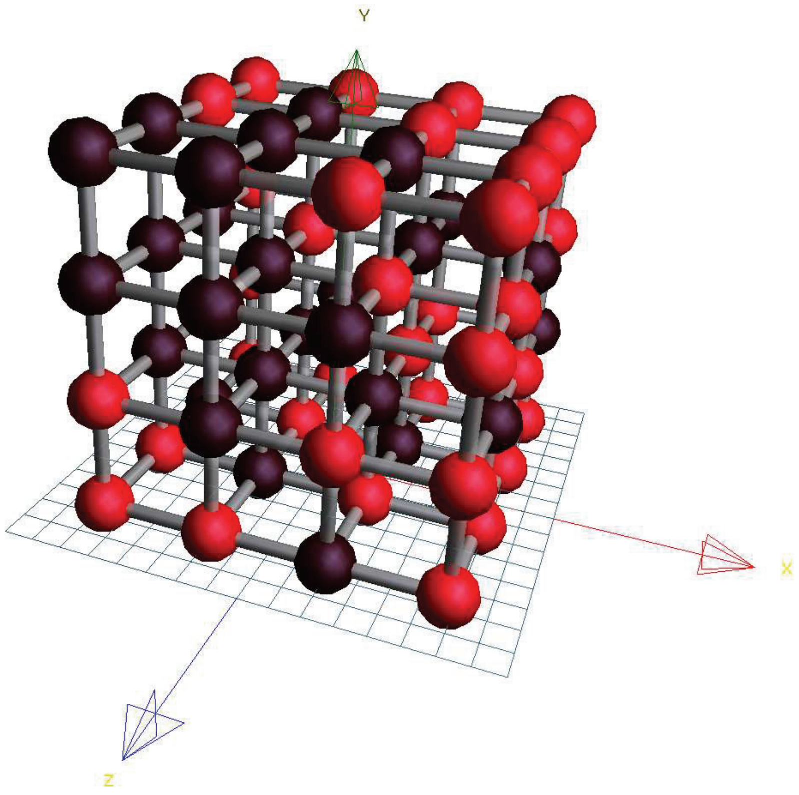


Figure 6.1: *Simulation of the 3D Game of Life automaton running on the BioCube. Red: the cell is alive; black: the cell is dead.*

In a similar way to the Life1 application (cf. § 4.1.1), the 3D version implements a unique cell of the Game of Life automaton in each *BBall*. The state of the cell is displayed by the *BBall* lighting up red when the cell is alive, and going out (becoming black) when the cell is dead. Figure 6.1 shows a view of the BioCube running the 3D Game of Life application, which is only composed of 64 Game of Life cells (in comparison to the 64’000 cells of the Life16 application). Interaction with the cube is very simple: merely touching a *BBall* toggles the status of the automaton cell from death to life, and vice versa. As for the BioWall, the speed of the automaton is controlled by the external device *BioCubeCtrl*.

6.1.2 Application architecture

The BioCube has the same structure as the BioWall where each FPGA is connected to its adjacent neighbors, and where a global enable signal, generated by the *BioCubeCtrl* module, allows synchronization of the computation of all *BBall*s. This *BioCubeCtrl* board controls the BioCube operations, and carries out the following tasks for our application:

- Configuration of all BioCube FPGAs with the Game of Life automaton application.
- Sending the initial states of each automaton cell to all the *BBall*s of the BioCube.
- Generating the global synchronization signal for the automaton (the C_EN enable signal).

This procedure allows us to start the 3D Game of Life automaton with a predefined pattern. Since the number of cells in the BioCube is smaller than that of the BioWall, it was difficult to find many interesting initial patterns capable of producing a periodic animation. Through computer simulations, we succeeded in finding a small periodic oscillator running over our small $4 \times 4 \times 4$ BioCube. Without any human interaction, our initial pattern converges after more than 30 steps to a state where this oscillation emerges.

Figure 6.2 summarizes all the tasks executed inside each FPGA, and shows the connections with the adjacent FPGAs. The main tasks are:

- Computation of the new automaton state.
- Loading and configuring each automaton cell with the initial state sent by the *BioCubeCtrl*.
- Interacting with users by sensing the *BBall* surface and displaying the state of the Game of Life cell.

As for the BioWall, the automaton clock needs to be slow enough to interact with human users. For enhanced performance, this global signal can be speeded-up to several MHz. While this possibility is not interesting for our Game of Life automaton, it could be needed in applications requiring a lot of computing resources and speed.

Since the communication capability between *BBall*s is limited to a single Low Voltage Differential Signaling (LVDS) link in each direction, we used the cellular communication module described in § 5.2.2 to transmit signals between *BBall*s. Figure 6.3 shows the assignment of signals to the inputs and outputs of this module. The two first signals are used for controlling the application and the IN_2 signals transmit the state of the cell to the neighboring cells. None of the other signals are used in the 3D Game of Life application.

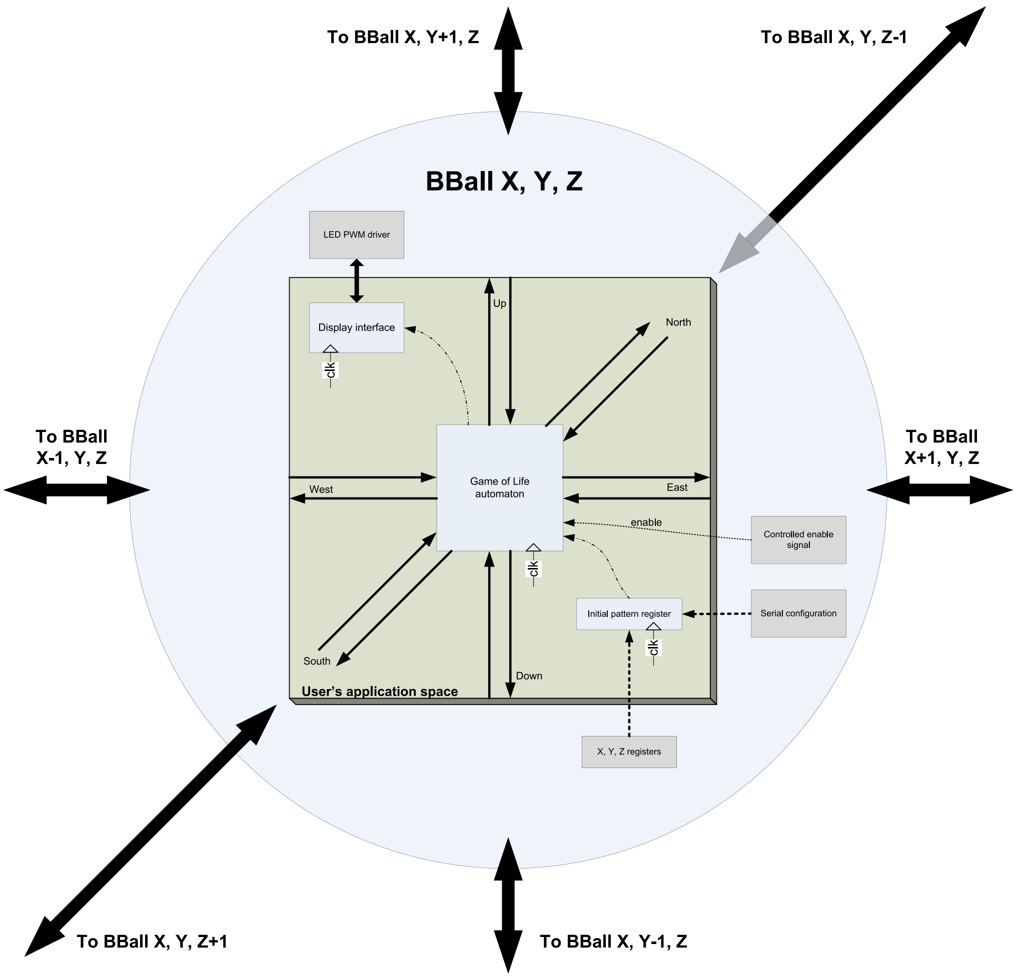


Figure 6.2: *Inside the FPGA: 3D Game of Life architecture. The big grey box implements the Game of Life including the automaton, display and configuration modules; the other external modules allow the application to interface with the I/Os of the* BBall *easily.*

Initial pattern

Once all the FPGAs of the BioCube are configured with the 3D Game of Life application, the *BioCubeCtrl* box transmits a serial stream with the initial pattern that needs to be loaded into the automaton cells in order to store its initial state. Once all the *BBall*s have received these data, the computation of the automaton can be started.

For the BioWall's Game of Life application we used a Shift Register (SR) to chain all the FPGAs together. This method had the advantage of simplicity, as serializing the data with the initial patterns was easy to implement and didn't use a lot of resources inside each FPGA. In the BioCube, we chose another method: although the SR method was still possible, we decided to add more flexibility to our configuration process and used the following solution.

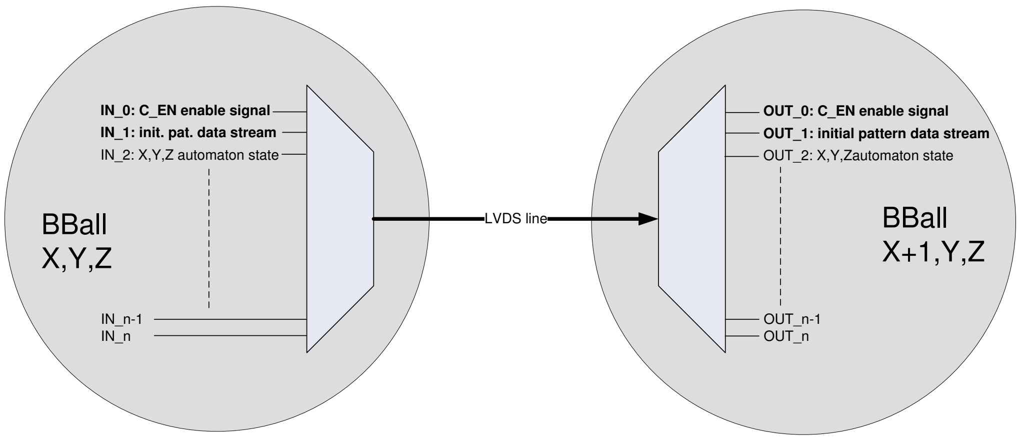

IN_0: C_EN enable signal
IN_1: init. pat. data stream
IN_2: X,Y,Z automaton state
BBall
X,Y,Z
IN_n-1
IN_n
LVDS line
OUT_0: C_EN enable signal
OUT_1: initial pattern data stream
OUT_2: X,Y,Zautomaton state
BBall
X+1,Y,Z
OUT_n-1
OUT_n

Figure 6.3: *Signals used with the cellular communication module.*

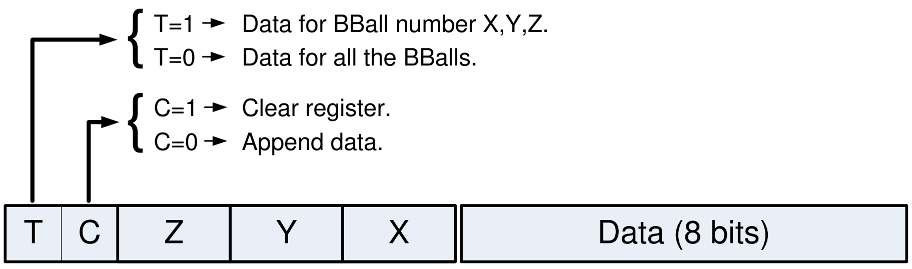

T=1 → Data for BBall number X,Y,Z.
T=0 → Data for all the BBalls.
C=1 → Clear register.
C=0 → Append data.
T
C
Z
Y
X
Data (8 bits)

Figure 6.4: *Protocol details of the serial stream transmitting the initial pattern to the BioCube.*

The FPGAs are connected in the form of a tree to the main node, which is the *BioCubeCtrl*. The signal, labeled IN_2 (Fig. 6.3) is broadcast from this controller to all the *BBall*s. The data transmitted over this line reaches all the BioCube FPGAs. Thus to assign the correct initial value to the correct cell of the automaton, a preamble is added in front of each item of data to indicate the intended receiver of the messages. These messages are always composed of two bytes, as detailed in figure 6.4. The first byte gives the identifier of the target *BBall*, and the second byte includes the payload. The following list gives the different modes of transmission available:

- $T = 0$; the data byte will be addressed to all *BBall*s in the BioCube.
- $T = 1$; the data byte will be addressed only to the *BBall* with the identifier equal to the Z,Y,X values (each of the three coordinates is encoded on two bits).
- $C = 0$; the receiving register inside each FPGA can have a dimension of several bytes. In this case, the data is appended to the data already written in this buffer.
- $C = 1$; any pre-existing data in the receiving register is firstly cleared, and the data is then stored in the Least Significant Bit (LSB) position.

In our 3D Game of Life application, only one bit needs to be transferred from the computer to each FPGA. Thus the initial state of the automaton cell is encoded in the LSB position of the payload byte. This application doesn't make the best use of all the possibilities given by such a method. However, since this module is provided to the application developer as an Intellectual Property (IP) core, it is only used in a minimal way for the 3D Game of Life, but it could be used to transmit several items of data for more complex application, as we will see with the next application in the following section.

6.1.3 3D Game of Life conclusion

This application shows us that an application from a 2D world, like the BioWall is easily transferable, to the 3D BioCube and that the applicative structure is quite similar, even if the machine's architecture is different. The BioWall has the main advantage of having a large surface able to display several pieces of information, or computation states. The BioCube suffers from the display limitations of the *BBall*, since it is more difficult for a human to visualize different states over a three dimensional structure. Thus, even though the computational power of the BioCube's FPGAs is far higher than that of the SPARTAN® used in the BioWall, the *BBall* display does not allow for more than a single Game of Life cell inside each FPGA. Consequently we will not try to exceed the computational and display surface of the BioWall, but limit our research to demonstrative applications of new 3D cellular algorithms (implemented on a hardware machine).

6.2 3D Tom Thumb

The main goal of this application is to present the hardware implementation of three-dimensional (3D) self-replicating structures endowed with universal construction and universal computation properties. Basically designed for the self-replication of two-dimensional (2D) loops with universal construction and universal computation, the Tom Thumb algorithm (cf. section 4.2) is revisited here in order to deal with the third dimension. According to this algorithm, a piece of configuration information comprised of flag data and code data is used twice during the self-replication process. It is used first in translation: the information ends up trapped in the new replicated loop, defining both its structure and its functionality. Second, it is used in transcription: the information remains mobile and moves along the loop in order to allow further replications. Through the addition of just a few supplementary flags, the Tom Thumb algorithm allows the self-replication of 3D structures equipped with universal construction and computation capabilities.

We will first describe the 3D Tom Thumb algorithm by means of a minimal structure composed of eight cells which will grow and then self-replicate to trigger the growth of three identical structures (Fig. 6.5). This example is sufficient to derive the detailed architecture of a three-dimensional Data and Signals Cellular Automaton (DSCA).

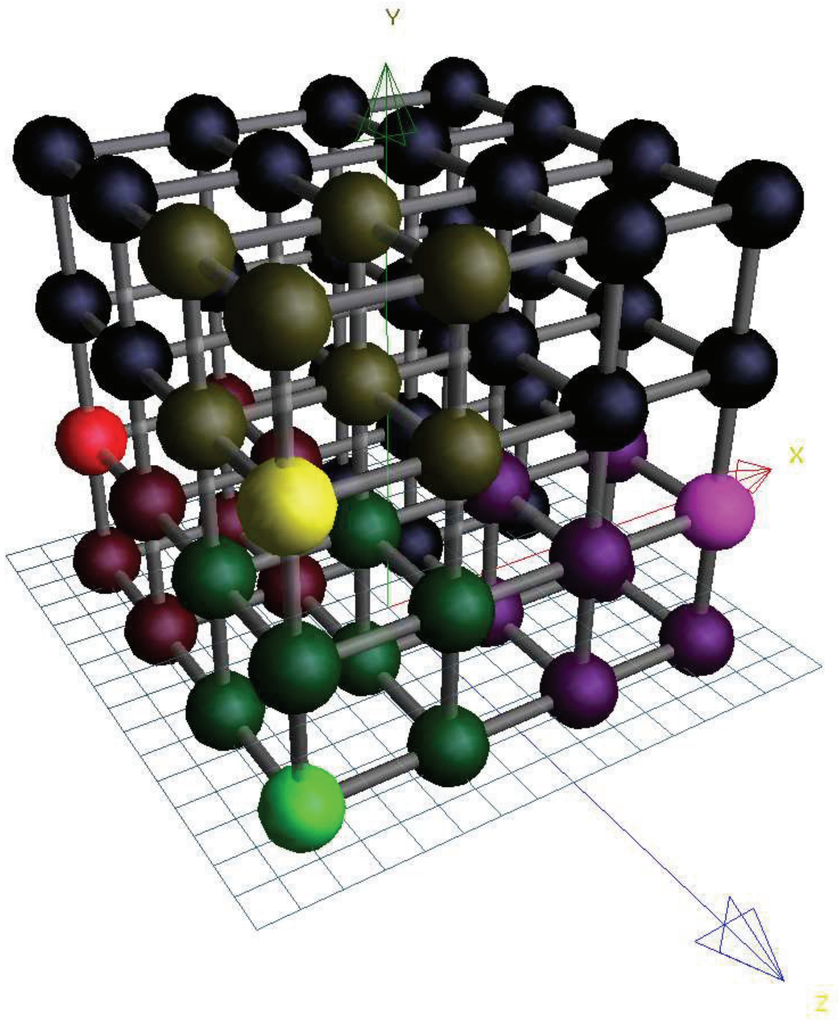


Figure 6.5: *Simulation of the 3D Tom Thumb running on the BioCube; the green cube is replicated three times with the child structures colored in yellow, pink and red.*

6.2.1 The 3D Tom Thumb algorithm

This minimal 3D structure is made up of eight cells organized as a $2 \times 2 \times 2$ array. In order to show the growth and the self-replication of this minimal structure, we have included 2D graphical representations. In figure 6.6, the eight cells of the minimal structure are organized as two levels $L = 1$ and $L = 2$ of two rows by two columns. Each cell is able to store in its four memory positions four items of configuration data. The original configuration information is a string of 16 items of data moving counterclockwise by one item of data at each time step ($t = 0, 1, 2, ...$).

The graphical representation as well as the hexadecimal representation of the data contained in the configuration string are detailed in figure 6.7. They are either *empty data* (0), *code data* (from 1 to E) or *flag data* (from 1 to 9 in addition to F). The code data is used to define the functionality of the structure. The flag data is used to

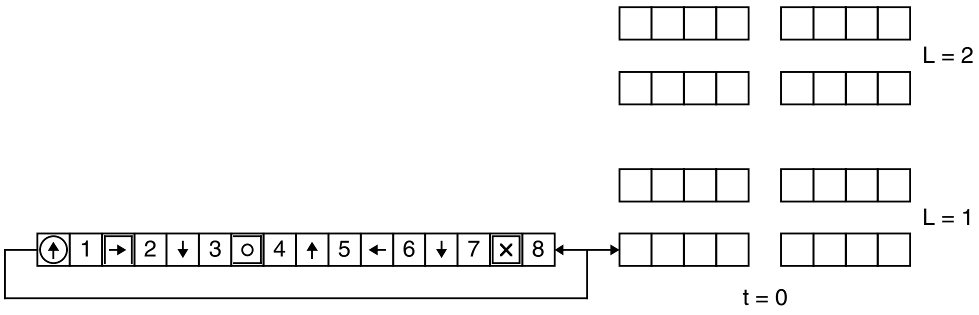


Figure 6.6: *2D representation of the minimal 3D structure ($2 \times 2 \times 2$ cells) with its configuration string at the start ($t = 0$).*

build the connections between the cells of the structure and to create branches for self-replication. The main difference with the 2D Tom Thumb is that the addition of new flags allows connections with superior and inferior levels to be created. In addition, each item of data is given a status and can be *mobile data*, indefinitely moving around the structure, or *fixed data*, definitely trapped in a memory position of a cell.

☐	: empty data (0)	C	: code data (1 ... E)	-	: mobile data
-	: don't care data (1 ... F)	F	: flag data (1 ... 9, F)	▨	: fixed data
				→	: north branch and east connection flag (7)
↑	: north connection flag (1)	←	: west connection flag (4)	○	: east branch and up connection flag (8)
→	: east connection flag (2)	○	: up connection flag (5)	×	: up branch and down connection flag (9)
↓	: south connection flag (3)	×	: down connection flag (6)	⊕	: branch activation and north connection flag (F)

Figure 6.7: *Graphical and hexadecimal representations of the data.*

At each time step, an item of data from the original configuration string is shifted from right to left and simultaneously stored in the lower leftmost cell (Fig. 6.6). Note that the first, third, ... data of the string (i.e. each odd data) is always a flag F, while the second, fourth, ... data (i.e. each even data) is always a code C. According to the Tom Thumb algorithm (cf. section 4.2), the construction of the structure, i.e. storing the fixed data and defining the paths for mobile data, depends on two major patterns (Fig. 6.8).

- If the two, three or four rightmost memory positions of a cell are empty (blank squares), the data are shifted by one position to the right (shift data).
- If the rightmost memory position is empty, the data are shifted by one position to the right (load data). In this situation, the rightmost F' and C' data are trapped in the cell (fixed data), and a new connection is established from the second leftmost position toward the northward, eastward, southward, westward, upward or downward cell, depending on the fixed flag information (F' = 1 or F, 2 or 7, 3, 4, 5 or 8, 6 or 9).

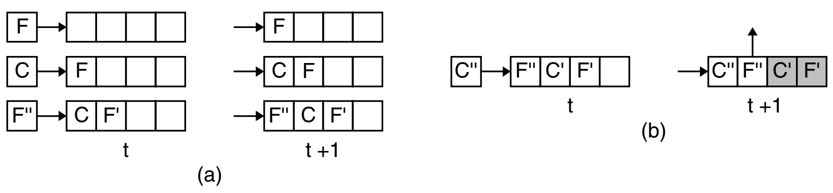


Figure 6.8: *Memory patterns for constructing a structure. (a) Shift data. (b) Load data.*

By applying the memory patterns in figure 6.8 to our original configuration string, we get two data trapped in a cell and a new connection toward another cell in the structure every four time steps (Fig. 6.9). At time $t = 32$, 32 pieces of data, i.e. twice the contents of the original configuration, have been stored in the 32 memory positions of the final structure. 16 data items are fixed data, defining both the structure and the functionality of the structure, and the 16 remaining ones are mobile data, which form a copy of the original configuration information. *Translation* (i.e. construction of the structure) and *transcription* (i.e. copy of the configuration) have thus been achieved.

For self-replication, the original structure is able to trigger the construction of three copies: one northward, one eastward and one upward. At time $t = 19$, the data pattern initiates the construction of the northward structure. In this pattern, the lower level upper leftmost cell is characterized by two specific flags, i.e. a fixed flag indicating a north branch (F = 7) and the branch activation flag (F = F). This pattern is visible in figure 6.10(a) (third row). The new path to the northward structure starts from the second leftmost memory position (Fig. 6.9). At time $t = 23$ and $t = 47$, the patterns corresponding to the third row of the eastward and upward signals in figures 6.10(b) and 6.10(e) initiate self-replication of the structure both to the east and upward. The other patterns are needed for constructing the inner paths of the structure.

The self-replicating structure in figure 6.11 is an example of a non-minimal four-column, three-row and three-level structure. All non-minimal structures can be realized according to this implementation which keeps the number of columns even in order to properly close the data path. These non-minimal structures involve a new flag (Fig. 6.12) and two more construction patterns (Fig. 6.13).

6.2.2 The Data and Signals Cellular Automaton (DSCA)

Data and Signals Cellular Automaton (DSCA) were originally conceived to provide a formal framework for designing growing structures [127, 126]. Such an automaton is made up of an array of cells, each of which is implemented as a digital system processing both data and signals in discrete time steps. The cellular array (grid) is n-dimensional, where $n = 1, 2, 3$ is used in practice.

In growing structures, the data and the signals represent two different types of information. The *data* constitutes the information that travels through the grown structure. The *signals* constitute the information that controls the growth of the structure.

The basic cell of our three-dimensional seven-neighbor DSCA works with the northward (N), eastward (E), southward (S), westward (W), upward (U) and downward (D) directed data (D) and signals (S) (Fig. 6.14). The cell computes its digital outputs O from its digital inputs I.

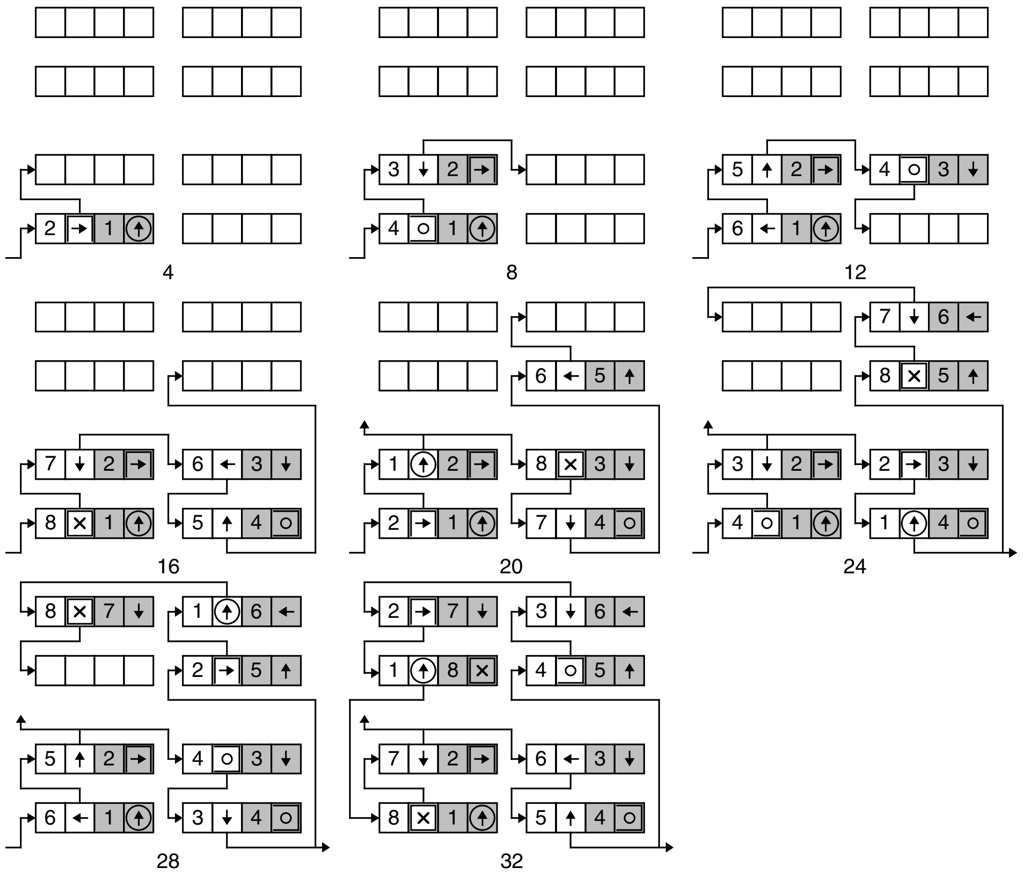


Figure 6.9: *Constructing the minimal structure ($t = 4$: north path, $t = 8$: east path, $t = 12$: south path, $t = 16$: up path, $t = 20$: north path ($L = 2$) and north branch ($L = 1$), $t = 24$: west path ($L = 2$) and east branch ($L = 1$), $t = 28$: south path, $t = 32$: down path and structure completion).*

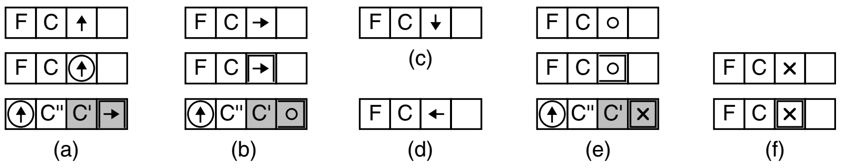


Figure 6.10: *Patterns of data triggering the path signals. (a) Northward. (b) Eastward. (c) Southward. (d) Westward. (e) Upward. (f) Downward.*

This cell is designed as a *digital system*, resulting from the interconnection of a data processing unit and a control unit. The *processing unit* handles the data. It is made up of the following resources:

- A 6-input multiplexer DIMUX for the selection of one of the six data input lines, $NDI3:0$, $EDI3:0$, $SDI3:0$, $WDI3:0$, $UDI3:0$, or $DDI3:0$.

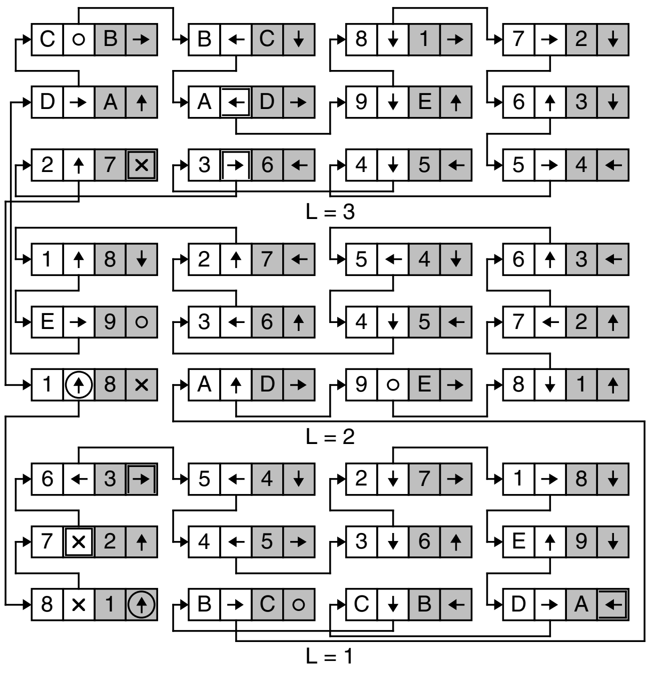


Figure 6.11: *Example of a non minimal structure (*$4 \times 3 \times 3$ *cells).*

: east branch and west connection flag (A)

Figure 6.12: *Graphical and hexadecimal representations of the additional data.*

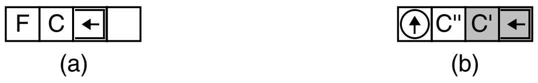
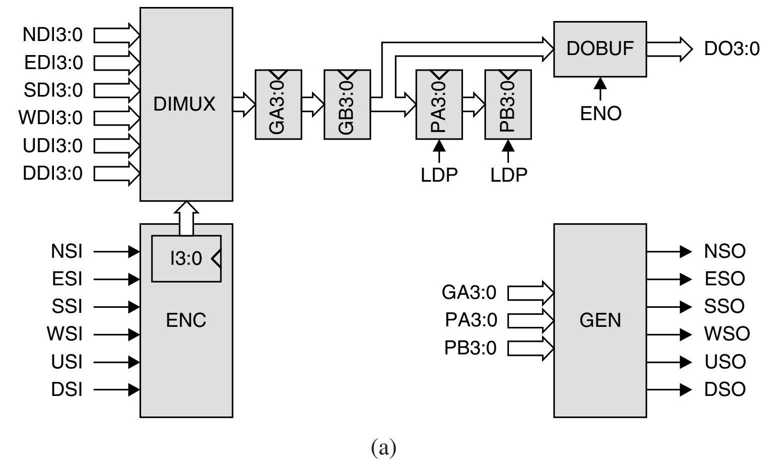


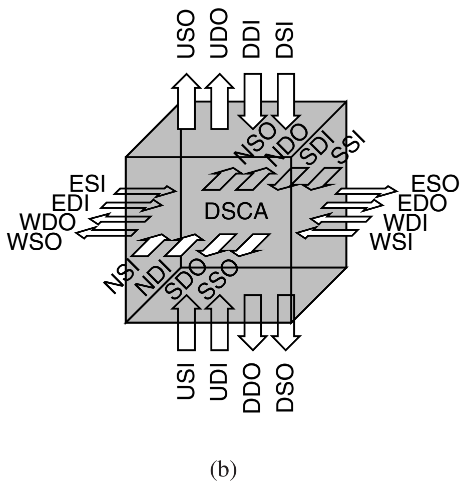
Figure 6.13: *Additional patterns of data triggering the path signals. (a) Westward. (b) Eastward.*

- A 4-level stack interconnecting two 4-bit registers GA3:0 and GB3:0 for the propagation of the configuration data with two 4-bit registers PA3:0 and PB3:0 for the memorization of the configuration data.
- A buffer DOBUF to enable the data output $DO3:0$.

The *control unit* computes the signals. It combines three resources:

- SI signals are the inputs of the encoder ENC.
- A 4-bit data register I3:0 used for the memorization of the selection operated by the multiplexer DIMUX.
- SO signals are the outputs of the generator GEN.





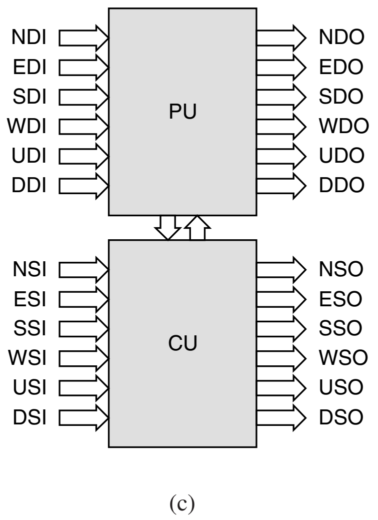


Figure 6.14: *Basic cell of the three-dimensional seven-neighbor DSCA. (a) Detailed architecture. (b) Macroscopic representation made up of the processing unit (PU) and the control unit (CU). (c) Macroscopic representation of the DSCA in 3D.*

The genotypic registers GA3:0 and GB3:0 always propagate the data DI selected by the multiplexer DIMUX, while the phenotypic registers PA3:0 and PB3:0 perform a hold operation or a load operation according to the control variable LDP. When equal to “1” this control variable indicates that the phenotypic register PB3:0 contains an empty data.

The selection realized by the data input multiplexer DIMUX is specified by the data input register I3:0.

The operations of the data output buffer DOBUF imply the control variable ENO. When equal to “1” this control variable indicates that the phenotypic register PB3:0 contains a flag.

The data input register I3:0 performs a load operation every time that there is at least one input signal equal to “1”.

The encoder ENC operates a priority coding of the input signals.

The generator GEN implements the output signals implied in the construction of the connections according to the patterns in figures 6.10 and 6.13.

6.2.3 BioCube implementation

The implementation inside the BioCube FPGAs consists of the VHDL description of the 3D Tom Thumb DSCA. Several additional modules are added to this module to allow the 3D Tom Thumb cell to interface with the resources of the *BBall*. The following modules provided as IP cores are included inside the FPGA and accomplish the described below tasks:

- Cellular communication: the control unit needs only one communication line between each cells; the processing unit transfers its data over a 4 bits bus. These communications are multiplexed over a serial line using the module described in § 5.2.2.

- Global synchronization: the C_EN enable signal generated by the *BioCubeCtrl* allows synchronization of the computation of the 3D Tom Thumb algorithm.

- Initial configuration: the configuration module described in the previous application is used for transferring the initial genome to the FPGA located at position $0, 0, 0$. These data originating from the *BioCubeCtrl* box are buffered inside the FPGA and the module output is connected to an input of the DIMUX (Fig. 6.14(a)). This application show the need fot the possibility of transmitting many data to the same *BBall* and justifies the C bit in figure 6.4, as 64 bits (or 8 bytes) corresponds to the minimal structure in figure 6.6.

- Graphic representation: we firstly chose to assign the data value stored in the phenotypic register a color displayed on the *BBall*’s Light-Emitting Diodes (LEDs). This solution shows us the configuration and replication process of the structure. Another variation of the display information we implemented consists of displaying the mobile data, or the genotype information, thus showing the data looping inside the structure. To distinguish each $2 \times 2 \times 2$ structure, we changed the color of the child’s structures.

- Replication trigger: the *BBall*’s sensitive sensor triggers the replication of the loop. This process is executed on a cell currently configured and which has a branch flag in its fixed data register.

6.2.4 3D Tom Thumb conclusion

Self-replication allows not only the growth, but also the repair of complete 3D structures. Self-replication is now considered as a central mechanism indispensable for those circuits which will be implemented through the nascent field of nanotechnologies [107, 40].

The applications of the self-replication of 3D structures are quite naturally classic self-replicating automata, such as three-dimensional reversible automata [56].

Some research could be done to improve the algorithm and its DSCA implementation to have an asynchronous implementation, which would be more suited to the BioCube architecture. In the same way, research on asynchronous cellular automata [86] is a field in which research could be carried out with the help of our system.

6.3 Conclusion

In this chapter and in chapter 4 we study two applications in a 2D and then a 3D environment. We used the Game of Life automaton to illustrate the implementation of a simple CA on the BioWall and the BioCube. This application is perfectly suited for these hardware architectures, where each FPGA can interact with its surrounding neighbors in the same way a cellular automaton does. Moreover, the interaction and display capabilities of the two machines are perfectly adapted for public interaction and demonstrating CA principles.

The 3D Tom Thumb application, by adding some specific code to the 2D algorithm (cf. section 4.2), allows the creation of complex structures, able to replicate inside an electronic crystalline structure. In chapter 9 we will discuss how the 3D Tom Thumb algorithm could be used to program and replicate a cellular structure inside an electronic paper tissue.

Aside from the major structural differences between the BioCube and the BioWall, an additional important difference resides in the FPGAs used in the BioCube: they are more powerful and can support larger designs and higher computing speeds, this could allow new kinds of research, especially in parallel computing, or interconnected networks.

In chapter 9 we will present some ideas for an electronic tissue whose internal structure is similar to that of the BioCube, but on the nanometer scale.

Chapter 7

BioTissue hardware description

As mentioned in the conclusion of chapter 3, the BioWall is a very powerful machine specially suited for huge cellular applications. However due to some weakness in its structure, it is unable to support more complex applications. Based on the limitations and missing features of the BioWall, we have developed a new cellular system, named BioTissue. As we will see in the next section, the BioTissue is still based on a cellular architecture, but we have added some main features such as dynamical reconfiguration and autonomy. The following sections will present all of the electronic boards that make up the BioTissue. Towards the end of this chapter, we will have a closer look at the firmware implemented inside this machine and how an application should interact with it in order to benefit from all of the features of the BioTissue system.

7.1 Characteristics and global overview

7.1.1 Introduction and motivation for creating a new bio-inspired machine

As seen in the previous chapters, the BioWall is a huge cellular machine, allowing experimentation with multiple bio-inspired applications. The development of this system enabled us to discover the general characteristics of such hardware architecture, but also showed us some limitations which hinder new research. A structure more closely resembling to biological cellular world was needed: autonomous and able to handle cell division, differentiation and reconfiguration. This is the reason for development of the BioTissue[1]. Its system's architecture is once more based on a cellular structure, as for the BioWall and the BioCube. The new structure is made of several layers and a partition has been created in the intelligent layer, giving us a computational layer and a control layer. This gives us the advantage of a reconfigurable element totally dedicated to the application for each cell, as we will see in this chapter.

There is another significant distinction between the BioWall, and the BioTissue. The BioWall is a large machine for which interactive dimensions are more important

[1]The word BioTissue is a combination of the words *bio* (for bio-inspired) and *tissue* (to emphasize the link with a biological cellular tissue).

than the performance. On the BioTissue, we placed the emphasis on the architecture and bio-inspired specificity rather than on size. The results is a small machine with a surface area of only 60 cm by 20 cm.

7.1.2 From BioWall to BioTissue

The BioWall has been successfully used for testing applications as described in chapter 4, but also for prototyping bio-inspired computing machines [131], and has served as a basis for the development of a second bio-inspired architecture, the POEtic tissue [133, 132]. In both cases, the same concept of highly parallel interconnected simple cells has been used as the background idea for the realization of the BioTissue architecture.

Despite the fact the BioWall fulfilled its role and has been successfully used for several others research projects, it suffers from several limitations which hinder the development of new complex applications.

BioWall limitations

These limitations are related to (1) the type of Field Programmable Gate Arrays (FPGAs), (2) the clock performance, (3) the communication performances, (4) the autonomy, and (5) the display:

1. Nowadays FPGA developments are happening very fast and new products are announced as often as new microprocessors; and performance increase as prices decrease ([6, 38]). At the time of the BioWall's components selection process, the Xilinx® SPARTAN® FPGA was able to offer quite satisfactory characteristics for a relatively low price. Due to the large size of the BioWall, we chose the XCS10XL SPARTAN® from Xilinx®. Today, the SPARTAN® third generation is on the market, which means that the BioWall's FPGA performance is quite limited in comparison to the current market offer. Moreover, the BioWall architecture forces us to use the same FPGA configuration for each cell in the whole system. It means that all of the $4'000$ FPGAs of the BioWall must be set with the same configuration, limiting the functionality of every unit to the $10'000$ equivalent logic gates of the Spartan XCS10XL. Having different FPGA configurations for the same application would have been useful for optimizing and improving some developments (like *BioWord* [20] which could be a perfect example of such a feature).

2. Due to the full synchronous architecture of the whole system with only one clock source, the considerable delays inherent in propagating a global signal over distances measured in meters limit the clock speed to about one megahertz. This latter fact confines the system to applications where the required computational speed is very low, like the ones for which human interaction is required (this is the intended target of the platform).

3. Each FPGA of the BioWall can only communicate with its immediate neighbor cells. About 20 lines connect each FPGA horizontally and vertically. As previously mentioned, the maximal frequency on these lines is less than one

megahertz. There is no direct link between FPGAs located diagonally from each other, nor any global lines linking several FPGAs together; such diagonal communications can be established, but with the disadvantage of using FPGA and lines resources. Signal multiplexing can be built inside each FPGA in order to increase the number of channels between each circuit, but has the disadvantage of being a big resource consumer.

4. The entire system is controlled by the *BioBox*, an electronic board connected to a PC, designed to configure all the FPGAs, and to set and distribute the clock signal to the $4'000$ FPGAs. Thus, the BioWall acts as a slave electronic system, although the application does not require any interaction with the host computer once configured. This limitation prevents the BioWall from being fully autonomous and introduces a functional bottleneck at the interface between the PC and the reconfigurable logic. Moreover, such architecture hinders dynamic reconfiguration, as the configuration process is controlled by the user from the PC.

5. The huge display of the BioWall is only able to display three different colors because it is composed of red and green Light-Emitting Diodes (LEDs). Furthermore, as the LEDs' intensity can not be changed, there is no brightness control, nor are intermediate shades available.

These drawbacks, along with the evolution of programmable logic devices, have led us to define a novel platform for the implementation of our BioTissue. In section 7.2, we will present this structure and its salient features.

The BioTissue specifications

We thus chose to develop the new BioTissue architecture based on a bio-inspired cellular structure like that of the BioWall but improving several aspects. Thus, we still find an array of FPGA aimed at computational purposes, coupled with input sensors, and output displays. But, the BioTissue needs to be more powerful (1), faster (2), suited for high speed communications (3), autonomous (4) and equipped with a Red Green Blue (RGB) display (5):

1. Since the BioTissue machine is made of a large array of FPGAs, it is important that the FPGA selected is powerful, and affordable. We chose the XCS3S200 SPARTAN®3 from Xilinx®, which at the beginning of this project was a new product only available as engineering samples. The main advantages are a bigger size in terms of the number of logic gates, with a more powerful architecture, coupled to several internal modules like Random Access Memory (RAM) blocks, Digital Clock Managers (DCMs), hardware multipliers...Moreover each FPGA should be rapidly configured with diverse bitstreams.

2. The SPARTAN®3 FPGA is manufactured with one of the newest lithographic process at a $90nm$ size, allowing fast functionality speeds. Consequently, a global synchronous system was no longer recommendable: the BioTissue is

built on a Globally Asynchronous, Locally Synchronous (GALS) architecture, enabling each cell in the tissue to have a dedicated clock oscillator at a fixed frequency of $50.0Mhz$.

3. Links between each FPGA must be universally configurable in terms of channel number and also fast enough to match with the computational power of the FPGA. Using a high speed serial communication protocol with a dedicated circuit will allow $1Gbit$ data bandwidth between each FPGA and will permit several communication types, like direct links between neighbored FPGAs as well as between distant ones.

4. A new kind of architecture allows the BioTissue to be totally autonomous. Each FPGA in the system is able to dynamically reconfigure a specific FPGA located elsewhere in the system. The BioTissue is no longer controlled by an external board or Personal Computer (PC).

5. The display is a 24-bit RGB LED screen.

7.1.3 Overview of the BioTissue

Autonomy and dynamic reconfiguration cannot easily be managed by a FPGA, so we decided to built our computational part on two layers. The first layer (*computational layer*), will be composed of an FPGA dedicated to the application purpose. It will not be linked to its neighboring FPGAs, but with another FPGA on a second layer(*routing layer*). The routing layer will handle the task of configuring the FPGAs on the *computational layer* and also solve the communication paradigm. As the SPARTAN®3 is built on volatile Static Random Access Memory (SRAM) memory, each cell will have a FLASH memory module to store configuration bitstreams and also user data.

This gives us the structure of our minimal unit, also called a BioTissue cell, which contains two FPGAs (only one can be configured by the user), two types of memories (FLASH and also SRAM), one touchsensor input, and an 8x8 RGB LED display. Figure 7.1 shows the layered structure of the BioTissue cell (to be compared with the BioWall molecule structure on page 38).

As we will see in the next sections, our BioTissue is composed of a set of 108 of these units. However, as the BioTissue is built using commercial components, it was not possible to design a machine made of an array of elements like in figure 7.1. We had to make some changes to fit with the dimensions of real components. Thus the system is built hierarchically from a very simple computing unit, called the *ECell* (the *computational layer*). Several of these units can then be connected, to a more complex structure called the *EStack*. Consisting of four different kinds of interconnected boards (computational, routing, power supply, and display), these stacks can then be joined to form a parallel network of any size of programmable circuits called the BioTissue (Fig. 7.3).

The *EStack* is composed of a set of stacks of Printed Circuit Boards (PCBs) (Fig. 7.2), that can be connected together side by side to create two-dimensional arrays of any size. The *EStack* size was given by the display element, which has a manufactured external dimension of about 20 cm by 10 cm.

Each *EStack* is composed of four kinds of boards:

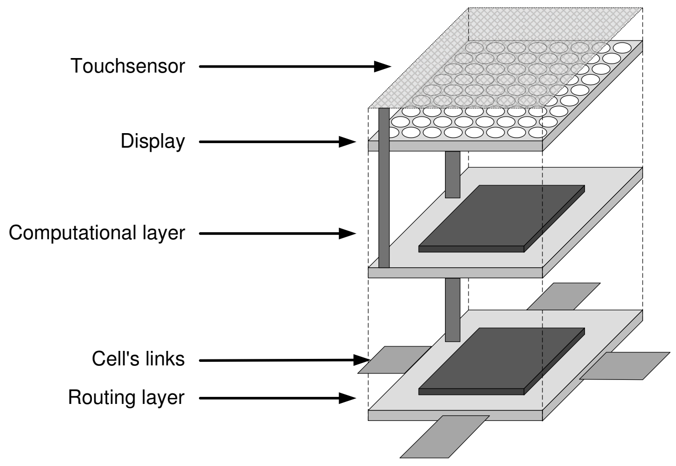


Figure 7.1: *The BioTissue cell structure.*

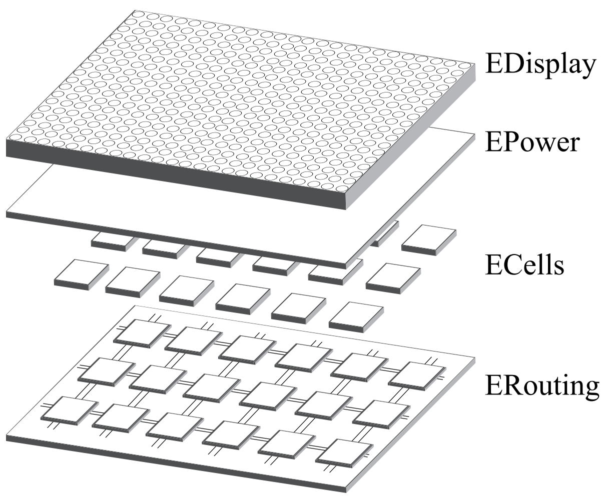


Figure 7.2: *The* EStack - *schematic*

- The *ECell* boards (18 per *EStack*) represent the computational part of the system and are composed of an FPGA and static memory. Each *ECell* is directly connected to a corresponding routing FPGA in the subjacent *ERouting* board.
- The *ERouting* board (1 per *EStack*) implements the communication layer of the system, and also carries out the configuration task for the *ECell* FPGA. Articulated around 18 FPGAs, the board implements a routing network based on a mesh topology which provides inter-FPGA communication but also communication with other routing boards.
- The topmost layer of the *EStack*, the *EDisplay* board, consists of an RGB LED display to which a touch sensitive matrix has been added.

- Above the routing layer lies a board called *EPower* that generates all the power supplies required by the system and handles functions such as startup and monitoring.

As we will later see in further detail in this chapter, some others electronics boards are needed, in addition to the *EStack*, to power, col, and monitor the BioTissue.

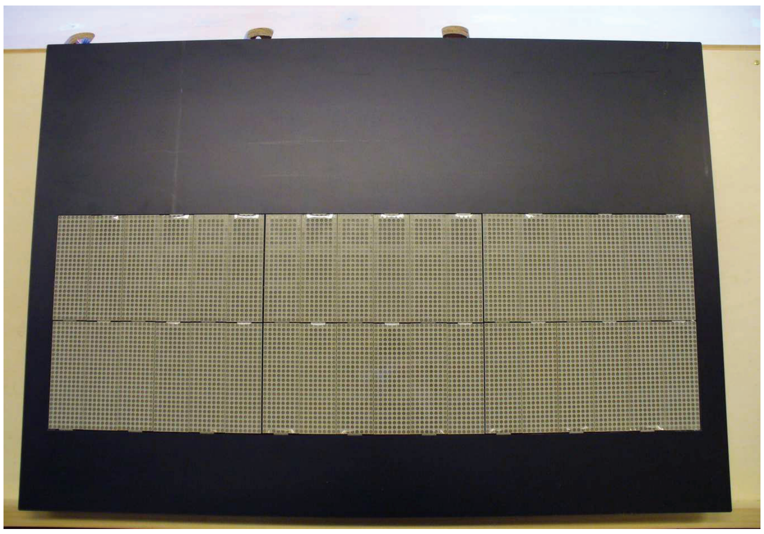


Figure 7.3: *Photograph of the BioTissue version composed of six* EStacks*.*

The BioTissue platform tries to avoid the different drawbacks described above by proposing an increased amount of versatility and interchangeability in the different constituting elements of the hardware system. Moreover, the system is built hierarchically by connecting elements of increasing complexity, which permits to handle more easily the complexity of the whole system.

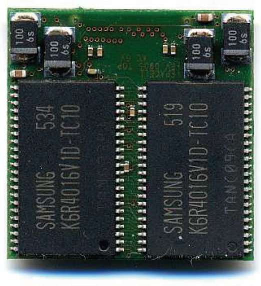
7.2 BioTissue boards description

7.2.1 The *ECell* board

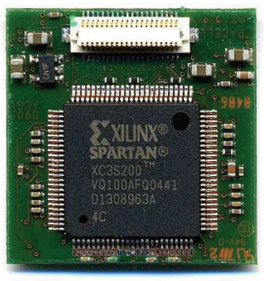
The *ECell* (Fig. 7.4), also labeled *computational layer* in figure 7.1, constitutes the basic building module of our hardware platform. All of the developed applications will run inside this board, and it thus represents the essential part of the BioTissue. As we will see in this section, all of the surrounding boards add features to the application and simplify user developments.

This board is articulated around a SPARTAN®3 FPGA[2] from Xilinx® coupled with 8 Mbits of SRAM memory and a temperature measurement chip. All of these components are soldered on a very small and thin (26 $\times$ 26 mm) 8-layer PCB.

[2]The exact model is the XC3S200-4VQ100C.



(a) Top view: SRAM memories.



(b) Bottom view: FPGA and connector (in white) to the *ERouting* board.

Figure 7.4: *Pictures of the* ECell *board.*

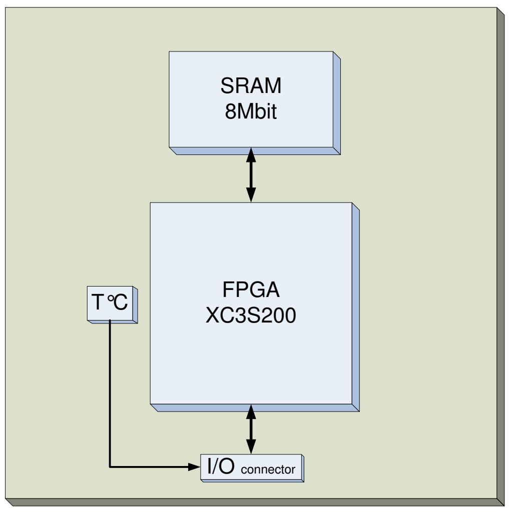


Figure 7.5: *Architecture of the* ECell *board.*

The main features are as follows: (1) FPGA, (2) memory and (3) temperature sensor:

1. Equivalent to 200′000 logic gates, this FPGA possesses some interesting features such as hardware multipliers, 216 Kb of internal dual-port memory, LVDS high-speed differential signaling and four DCM that make it possible to obtain, working frequencies up to 300 MHz, from the 50 MHz local clock.

2. In addition to the 216 Kb of FPGA internal memory, two external memory chips are present on the board, which gives additional memory of 8 Mb fast SRAM[3] directly linked to the FPGA.

[3] 10 ns access time.

3. Depending on the configuration running inside the FPGA, overheating can occur. A measurement chip located close to the FPGA can give information on local temperature, which is transmitted to the *EPower* board (the main supervisor of all *EStack* temperature sensors).

The *ECell* possesses various connections with the other components of the system that all pass through the connector visible on top of Fig. 7.4(b); these connections are described in table 7.1.

Type	Line numbers	Description
Application links	6 LVDS	Differential high-speed connections lines with the subjacent FPGA on the *ERouting* board (3 pairs in each direction, 500 Mbits per pair).
Config	5	Configuration lines needed to configure the *ECell* FPGA. Control and bitstream come from the *ERouting* board.
Clock	1	50.0 MHz clock signal coming from an oscillator located on the *ERouting* board.
Reset	1	*ECell* reset signal, generated by the *ERouting* layer.
Com.	6	Communication bus to the *ERouting* and *EPower* boards to carry the display and control signals.
Temperature measure	3	Temperature sensor lines.
Power	12	Power supply lines[a].

[a]each line can sustain a 300 mA current. The maximum allowed current for all the board is 1.5 A

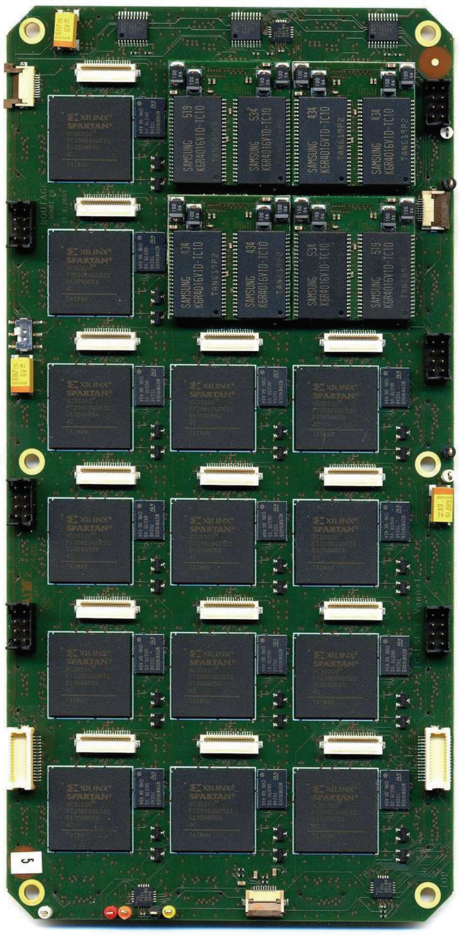
Table 7.1: ECell *I/O summary.*

7.2.2 The *ERouting* board

One of the main challenges in today's hardware architectures resides in implementing versatile communication capabilities that are able to provide a sufficient bandwidth, whilst remaining cost- and size-efficient, as evidenced in research for Network-On-Chip (NoC) [16] and other [83] systems. For our platform, we opted for a solution based on high-speed serial connections able to sustain different kinds of communication routing algorithms.

To reduce as much as possible the load on the *ECell* board, where the computational power resides, communications in our system are implemented within the *ERouting* board, which also handles tasks such as configuration and display management for the cell.

Measuring 192 $\times$ 96 mm, this highly complex board (twelve layers, more than 7'000 vias, 1'742 components) has components soldered on both sides and can host eighteen *ECell* boards (Fig. 7.6). Based on a six-by-three regular grid


(a) Top view: the *ERouting* board with four *ECell* plugged-in on the top right corner. The regular structure is easily recognizable, with close to each FPGA, the FLASH memory and the connector for hosting the *ECell* board.

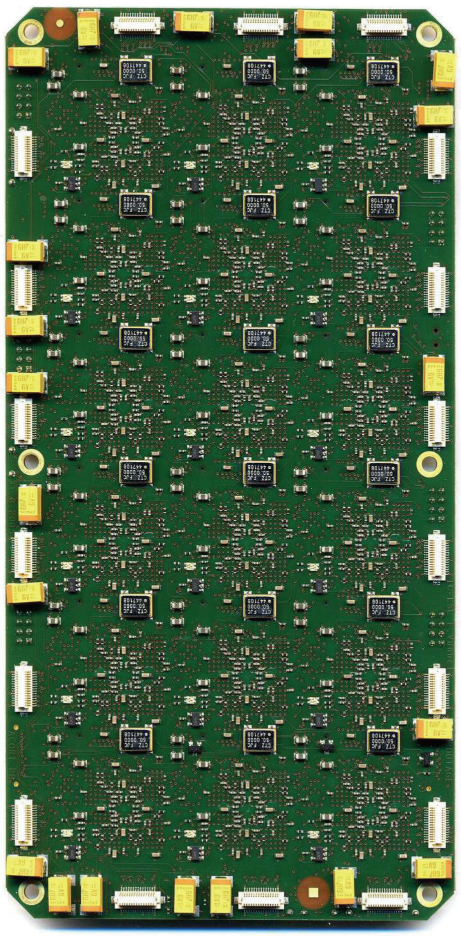

(b) Bottom view: the rectangular grey components are the oscillator; one for each cell of the BioTissue. White connectors surrounding the PCB are the links to others *EStack*.

Figure 7.6: *Pictures of the* ERouting *board.*

topology (Fig. 7.7), this board is composed of a set of eighteen reconfigurable circuits (the *ERouting* FPGAs) and eighteen FLASH memories. The functions of this board is outlined as follows: (1) FPGA, (2) memory, (3) communications, (4) temperature sensor:

1. The FPGA is the same type as the one used in the *ECell* board; this model is characterized by a greater number of pins and contrary to the *ECell* FPGA, it is not reconfigurable[4]. Its main tasks are to configure the *ECell* and manage communication protocols.

[4]Excepted for firmware upgrades that can exceptionally be done using the Xilinx® *Parallel Cable* adapter.

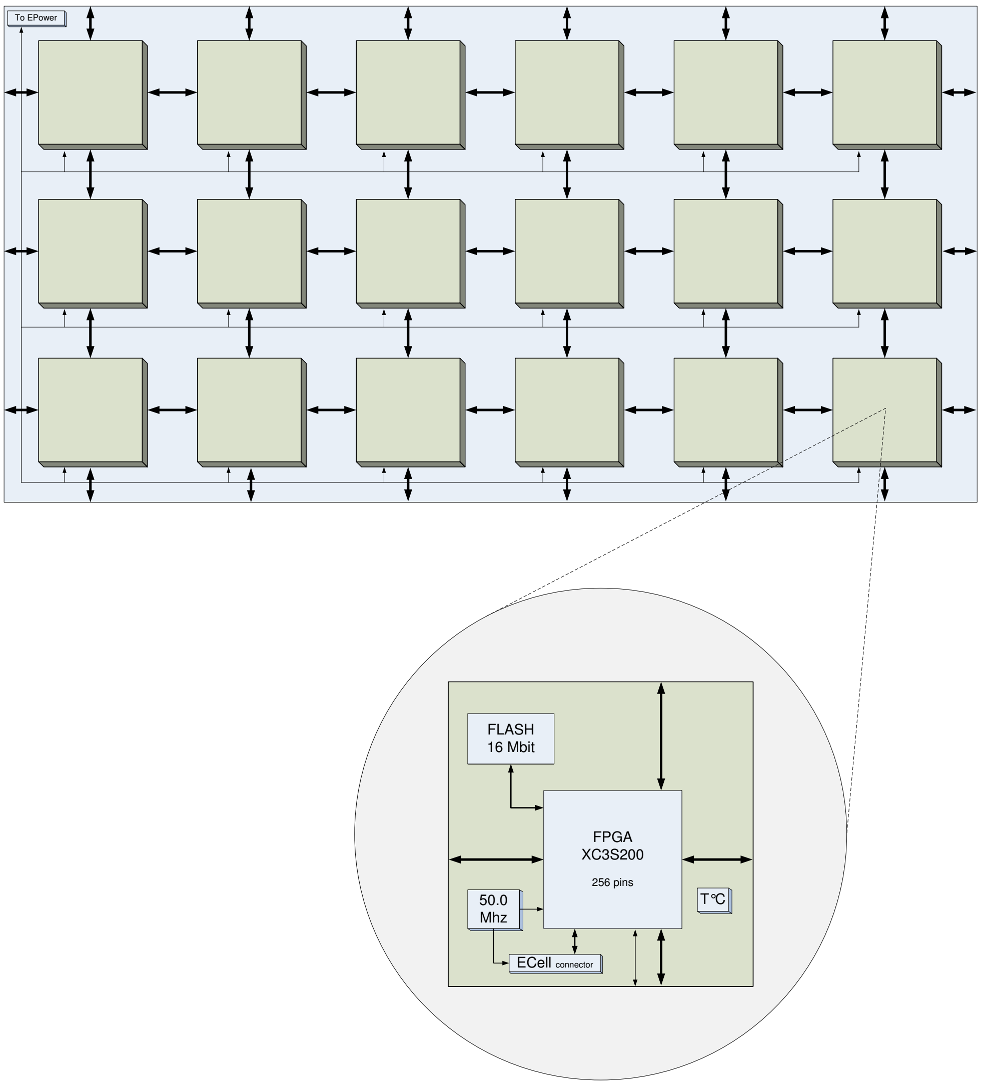


Figure 7.7: *Architecture of the* ERouting *board.*

2. A 16 Mb FLASH memory directly connected to the *ERouting* FPGA can store up to 16 different configurations for the *ECell* FPGA, and also stores data computed by an application and transmitted from the *ECell* to this memory.

3. Three different types of bus are present on this board. The first one, which will be presented in detail on next page, is a high-speed differential bus. In parallel to this one, a standard CMOS bus allows control of signaling. A third bus connects together each FPGA of the *ERouting* layer; it is also wired to the *EPower* board and has the task of getting values to be displayed onto the screen, as well as exchanging status information with each FPGA and with the *ERouting* board.

4. On this board we also have measurement circuits located close to the *ERouting*

FPGA. Three dedicated microcontrollers collect all of the temperature values from this board and also from the 18 *ECell* temperature sensors, and transmit these values to the *EPower* board, using a dedicated serial bus.

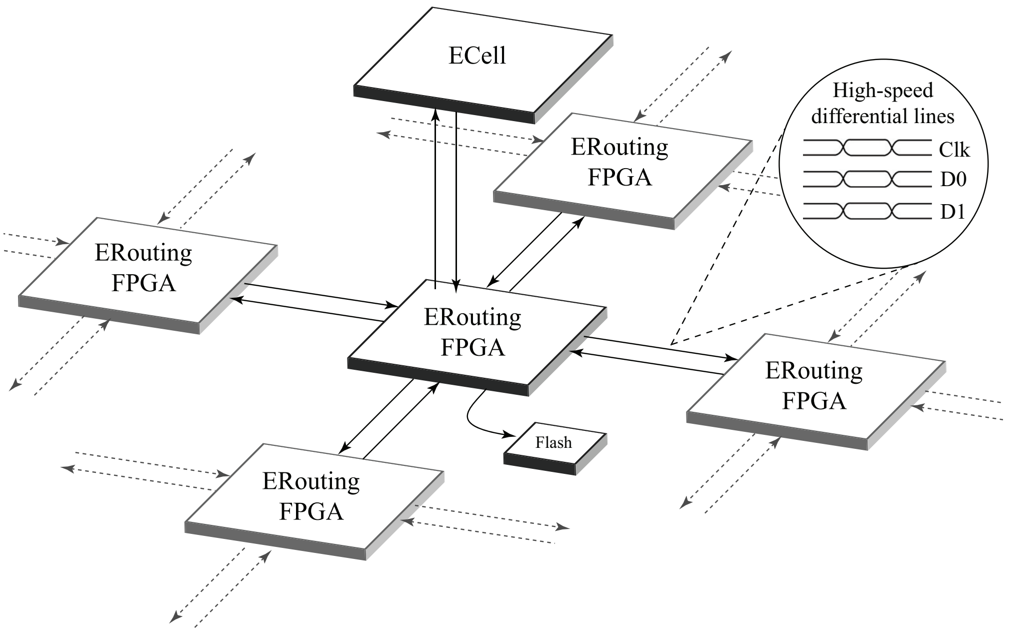


Figure 7.8: *Detail of* ERouting *FPGA links with its* ERouting *neighbors and it* ECell *module.*

High speed communication

As the main purpose of the board is to implement the routing network that connects the computational units of the system (the *ECell* boards), one of the most crucial aspects of the *ERouting* board is the type of connections that link the board's FPGAs together and with the *ECell* boards.

Physically, every *ERouting* FPGA is linked to its four cardinal neighbors and to the *ECell* board above it (Fig. 7.8). This setup was selected for its modularity and scalability (it avoids long and global communication lines that could cause bandwidth degradation in a big BioTissue configuration) and is the kind of layout typically used in cellular computing applications.

The links between each FPGA are implemented using the built-in SPARTAN® Low Voltage Differential Signaling (LVDS) I/O drivers that allow, in our case, data rates up to 500 Mbits/s. As depicted in figure 7.8, two communication buses (one for each direction, 3 bits per bus) are present for each neighboring pair. Because the SPARTAN® family does not provide serial transceivers directly integrated on-chip, the transmitter and receiver blocks were written in VHDL. Also, since there is no global clock in the system and no clock recovery possibility, a clock signal is transmitted on one differential pair to synchronize the data transmitted on the two other pairs. Thus, at the *ERouting* level, a bandwidth of 1 Gbit/s is available on each *ERouting* FPGA for every direction. Moreover, the same type of bus exists between each *ERouting* FPGA and its corresponding *ECell*.

As the *ERouting* boards constitute the communicating backplane of the whole BioTissue, connections between the different *EStack* boards are also implemented here, as seen in figure 7.6(b). External connectors are present on the four sides of the board and provide the same connectivity as the links between the FPGAs: two adjacent *ERouting* boards then represent effectively a single uniform surface of FPGAs. This setup allows the creation of systems consisting of several *EStack*s that behave as a single, larger *EStack*.

7.2.3 The *EDisplay* board

Unlike the above-mentioned BioWall, which was primarily a demonstrator, the main purpose of BioTissue is the high-speed prototyping of complex multi-cell systems. Nevertheless, the success of the earlier machine led us to integrate a relatively simple display, into the new one. On the very top of the *EStack* lies a 30-bit RGB LED display capable of displaying 48×24 pixels that can be refreshed at a rate of 100 times per second (a dedicated SPARTAN® on the *EPower* board manages the display's framebuffer).

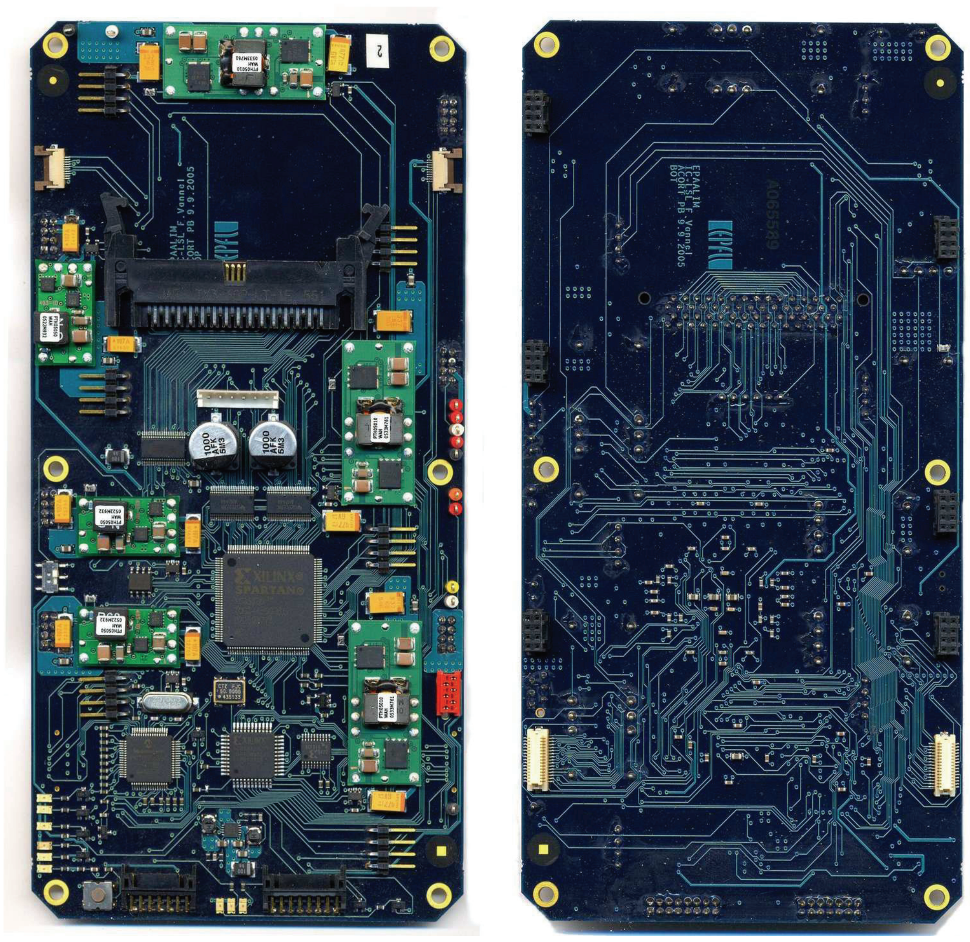
The purpose of this display is to provide a distributed overview of the way the system operates, for example, to illustrate its operation at reduced speed or to display long-term patterns such as network congestion or thermal buildup). Each *ECell* has access to only part of the screen, namely a square of eight by eight pixels directly above it. To provide a direct human interface to the system, a touch-sensitive surface is glued to each square.

Even if the resolution available for each *ECell* is very limited, the main advantage of this kind of screen resides in the fact that it is possible to put several screens border to border without any gap, a necessary feature in view of building large systems consisting of several *EStack*s side by side.

This display is manufactured by the Rohm company. It is constituted of two layers: an electronic control PCB and on top of it a circuit with all the LEDs. As the choice of displays of this type was really limited, we chose the LPM-1153BMU702 whose size is 192 mm by 96 mm. This dictated the dimension of our *EStack* system. On top of this display, we added a human sensitive touchscreen. The resolution of this screen is minimalist, with only 18 different zones, giving a one bit input for each cell of the BioTissue. These inputs are connected to the *EPower* board as we will now see.

7.2.4 The *EPower* board

The SPARTAN® FPGAs that are used throughout the BioTissue are very recent products, built on a 90 nm Complementary Metal-Oxide-Semiconductor (CMOS) process. This kind of technology makes them very fast FPGAs with multiple powerful features, but has the disadvantage of being powered with several low-power voltages. Thus, a 1.2 voltage is needed by the FPGA core, and a 2.5 V for the LVDS interface and for configuration purposes. Finally, a 3.3 V voltage is needed to interface the FLASH


(a) Top view: the big black component on top of the board is the connector for the *EDisplay* board. The two at the bottom are the 5V and GND inputs powering the whole *EStack*. Green parts of this PCB are the DC / DC convertors.

(b) Bottom view: black connectors surrounding the board are the power link for the *ERouting* and *ECell*s boards. The white one are for control and data communication between the *ERouting* and the *EPower*.

Figure 7.9: *Pictures of the* EPower *board.*

and SRAM memories. Moreover, all these voltages must be very well stabilized, as Xilinx® FPGAs allow only ±5 % tolerance.

To cope with all these requirements and the fact that the eighteen *ECell*s and the *ERouting* board are not only very complex but also power-hungry, an *EPower* board was added on top of the *ERouting* and *ECell* layers. This six-layer board, mainly responsible for supplying the correct voltage to all the components on the boards underneath, is the same size as the *ERouting* board. Articulated around six DC / DC converters, this board generates from a global 5 V the three mentioned voltages of 1.2 V, 2.5 V and 3.3 V that are then brought to the *ERouting* board using six 8-pin connectors.

Due to the high complexity of the boards in the *EStack*, a microcontroller on the

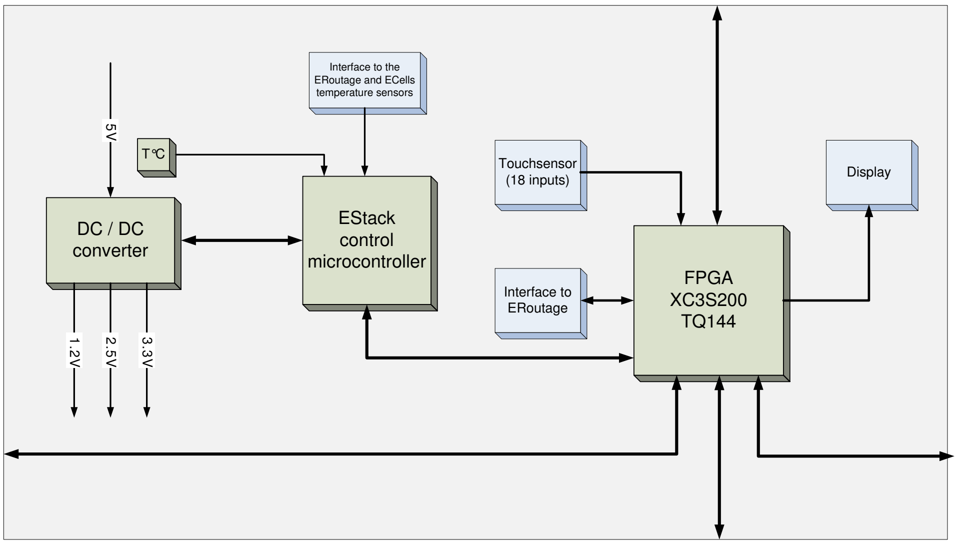


Figure 7.10: *Architecture of the* EPower *board.*

EPower board acts as a supervisor and checks several factors, like power supply stability, and monitoring the temperatures of all *EPower*, *ERouting* and *ECell* boards, in the aim of preventing failures. This microcontroller is also responsible of supervising the start-up of the whole *EStack*, a rather complex sequence that involves switching on the DC / DC converters, controlling the stability of all voltages, configuring the *ERouting* FPGAs, and setting initial states. If any of these tests should fail, warning signals are set, and depending on the severity of the problem, the entire system can be switched off to prevent damage.

As previously mentioned, another FPGA on this board is in charge of managing the framebuffer for the *EDisplay* board. It periodically gets from each *ERouting* FPGA its local video memory content (8 x 8 x 24 bits) and merges them together to reconstruct a local image of 48 x 24 pixels.

The last task of this board is to scan the touchsensor surface, debounce these inputs, and transmit the status of each sensor input to its matching *ECell* board.

Figure 7.9 shows this board and all its components while figure 7.10 focuses on the *EPower* architecture.

7.2.5 The *EStack*

Up to now, we have seen the description of four different boards. Their assemblage constitutes our main module, named the *EStack*.

In figure 7.11 can be clearly recognized the *EDisplay* board on top of the stack. The flexible plastic wires provide the connection for the touchscreen. These are directly connected to the *EPower* board. Below this, at the lowest level, is the *ERouting* board. *ECell* boards, which have a smaller size, are not visible on this picture, but they are located between the *ERouting* and the *EPower* boards, as visible in figure 7.2 on page 109.

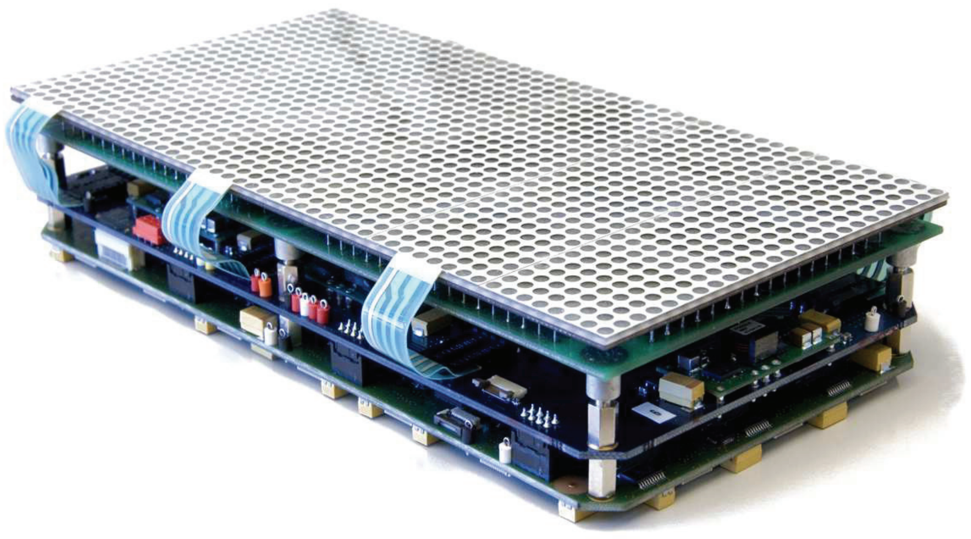


Figure 7.11: *Picture of the* EStack.

Scalability in cellular architecture is crucial. The *EStack* is not the end result of our research, but an element of the system. Several of these boards can be joined together, through border connectors in the *ERouting* board, in a two-dimensional array (details of the connections between *EStack* boards are listed in table 7.2).

Type	Line numbers	Description
Application links	6 LVDS	Differential high-speed connections lines with the FPGA on the neighbouring *ERouting* board (3 pairs in each direction, 500 Mbits per pair).
Control	6	Control lines needed for communication between FPGAs located on the *ERouting* layer.
EPower control	4	Theses lines are used for communication between *EPower* boards (the main messages transferred on these lines are *EStack* status, BioTissue power sequence status, display screenshots).
Power	8	Ground lines for potential reference between *EStack*s.

Table 7.2: *Inter-*EStack *I/O summary.*

7.2.6 The BioTissue: an assemblage of *EStack*s modules

Connecting several boards potentially allows the creation of arbitrarily large programmable logic surfaces. The current test configuration, which has been built and tested, consists of six *EStack*s in a 3 by 2 array (Fig. 7.12). Modularity given by such an architecture could let us imagine other topology like the one in figure 7.13.

The process of connecting *EStack* boards is facilitated by a mechanical structure as shown in figure 7.12. Links are not made of flat wires[5] like for the BioWall and the BioCube, but with an interconnection board consisting of a simple PCB with only female connectors on its top (the corresponding males are on the bottom of the *ERouting* boards). We built a 3 x 3 support allowing several kinds of *EStack* dispositions, but used for the framework of this thesis only the 3 by 2 configuration.

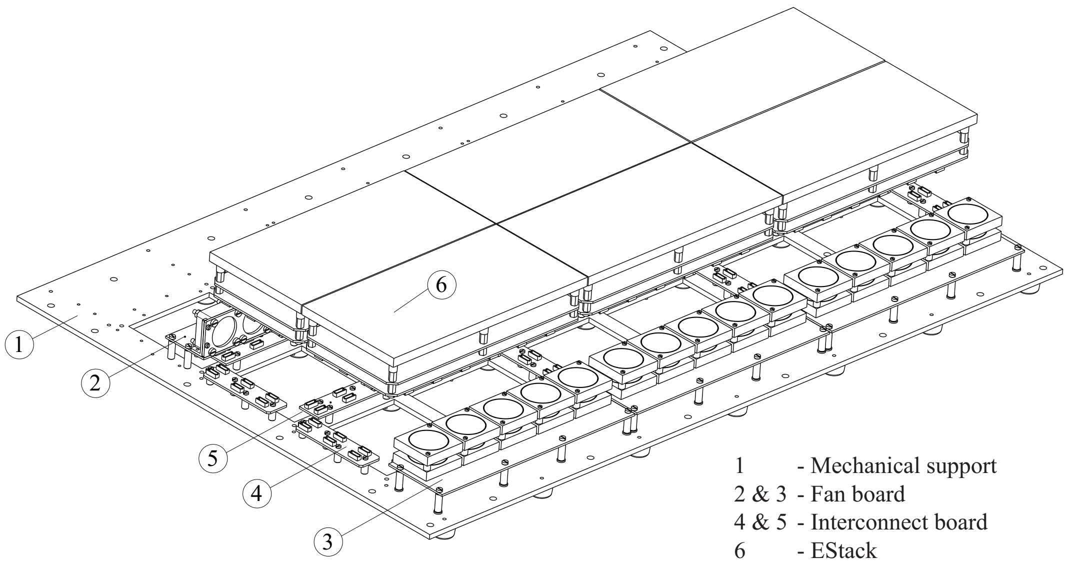


Figure 7.12: *The BioTissue system.*

Due to the fact that our system is made of a set of really complex and powerful boards, we need to take in consideration several other factors like power supply for the BioTissue, and a common problem in high-energy consumers: heat dissipation.

Power supply

Since each application uses a different FPGA configuration, power consumption is very difficult to calculate in advance within a reconfigurable system. However, Xilinx® XPower®[51] tools give a worst case current consumption of 1 A for the core. This estimation is based on the SPARTAN® 3 FPGA used, with a running clock frequency of 100 MHz and using most of the FPGA resources.

As we have a total of thirty-six of these FPGAs for each stack, the maximum current that might be drained could be as high as 36 A on the 1.2 V power supply. In

[5]Defective links could occur too easily and moreover the assemblage process is difficult.

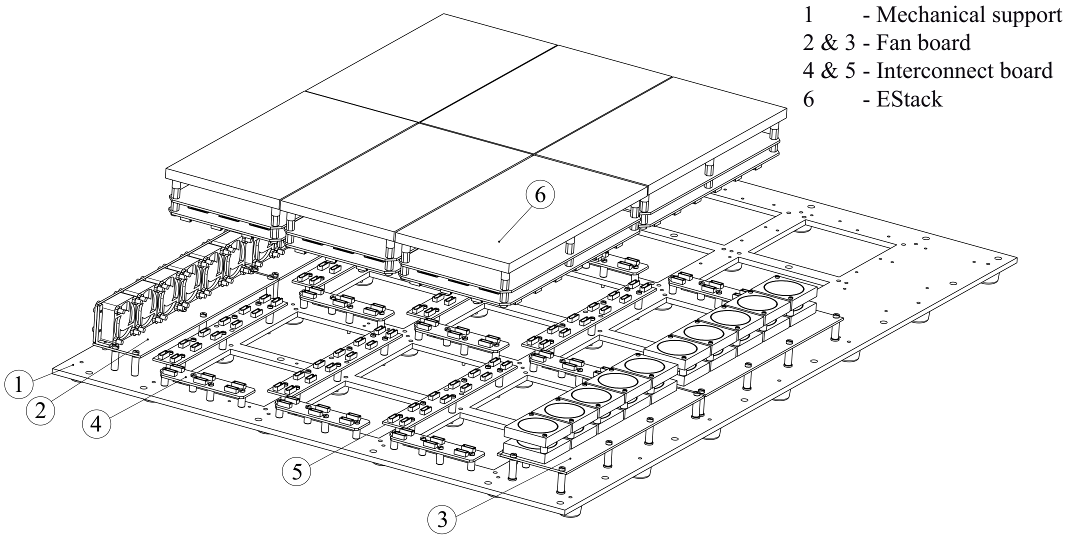


Figure 7.13: *Variant of the BioTissue: another arrangement of the* EStacks *boards.*

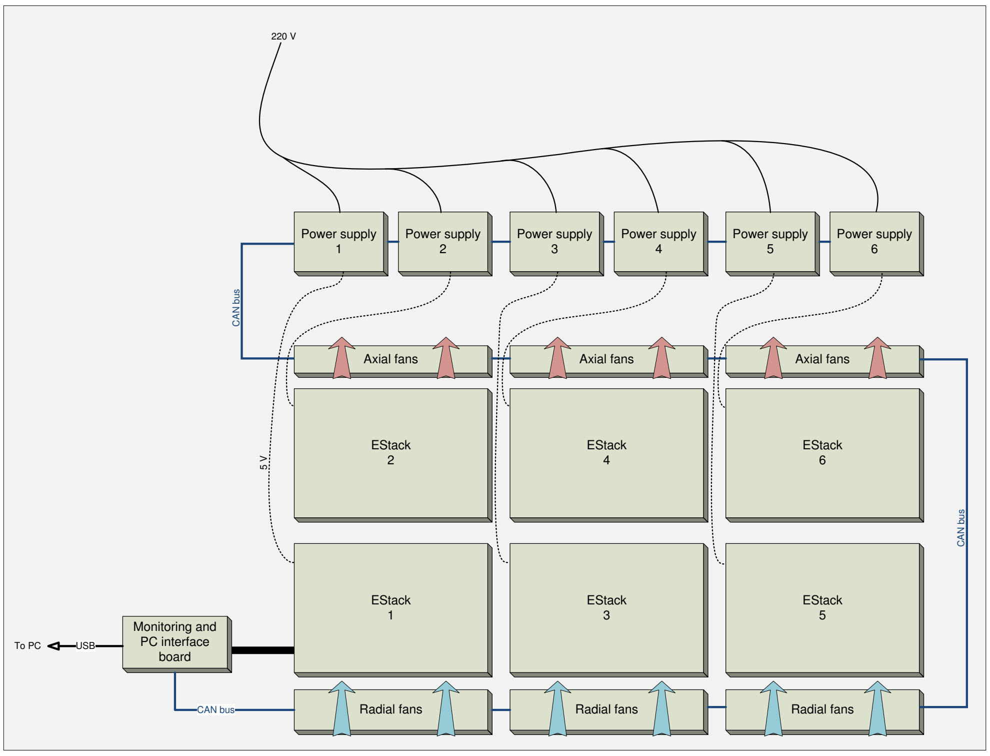


Figure 7.14: *The BioTissue with all its surrounding boards.*

comparison, the power estimations of 2.5 V and 3.3 V are very low, with a maximum 4 A for each of them. In addition to the FPGAs, the last significant energy consumer is the screen, which can use a maximum of 30 Watts. Thus, the total maximum consumption for one *EStack* can be up to 100 Watts. A 5 V input thus implies a maximum current of 20 A for one *EStack*, or 120 A for the BioTissue!

A power supply able to sustain a stabilized 5 V voltage up to 120 A is really expensive, and we chose the alternative of putting a commercial desktop PC power module for each *EStack*. In addition to being low-cost, such a module has the advantage of allowing an input voltage in the range of 85 V to 240 V with accurate output and can easily source high current up to 30 A. One such power supply for each *EStack* module, gives us the following kind of power bank (Fig. 7.15).

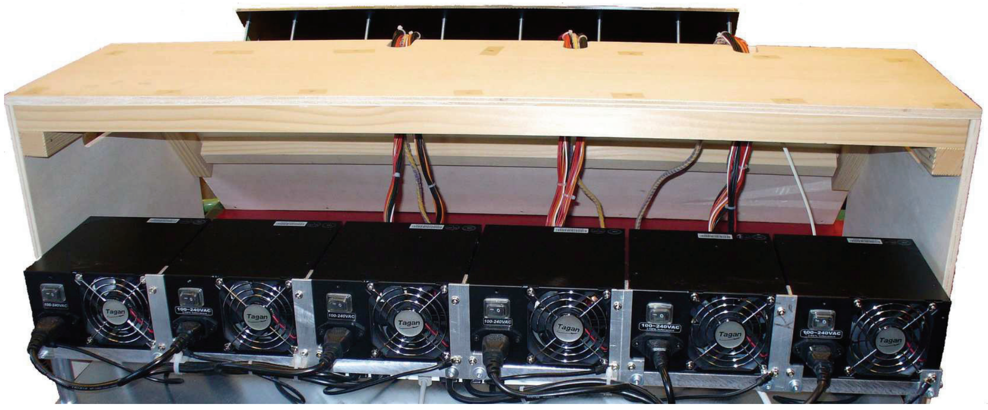


Figure 7.15: *The BioTissue power units.*

Because of the relatively high price of the components and the low number of *EStack* boards produced, we took some precautions to minimize the risks in component failure due to short circuits or thermal stress. Each power module in figure 7.15 is controlled by a dedicated board (Fig. 7.16) which controls the state of the power brick, and monitors the current drawn by the *EStack*. This information is transmitted to a supervisor unit using a Control Area Network (CAN) bus. In case of over-current (actually set to a 20 A limit) or voltage error, a failure message is transmitted and the power module is switched off to prevent damage to the *EStack*.

Heat consideration

Despite the fact that the reconfigurable circuits in the platform use a state-of-the-art fabrication technology that makes them less power hungry than the previous FPGA generation, the number of circuits involved in the whole system makes thermal management a real issue, as each *EStack* consumes a maximum power of 100 Watts. To solve it, we implemented an adequate cooling system in order to evacuate the generated heat. Thus, we integrated boards with fans on the top and bottom borders of the BioTissue. At the bottom of the system, there are three boards. Each of them are built with seven independently controllable fans to blow cool air inside the system, while on top, fans extract hot air away (Fig. 7.14). This two-stage fan construction creates an airflow inside the several layers of boards, allowing efficient cooling of the BioTissue.

Even if the chosen fan modules are almost silent, they still generate a light audible noise. A control process, which constantly monitors the internal temperature of *ECell*, *ERouting* and *ECell* boards, will only switch fans on when the temperature reaches

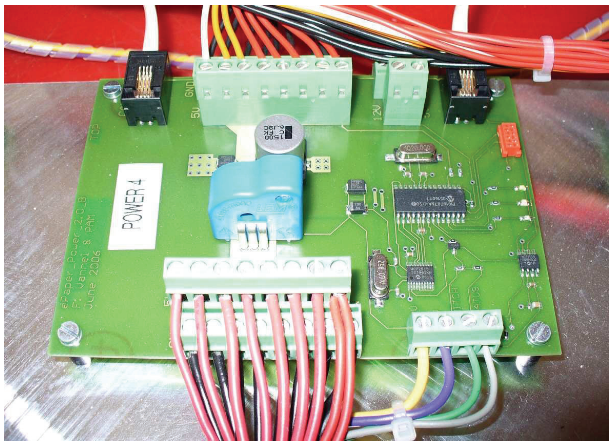


Figure 7.16: *Main power control boards; large current demand means several wires are needed to sustain these loads.*

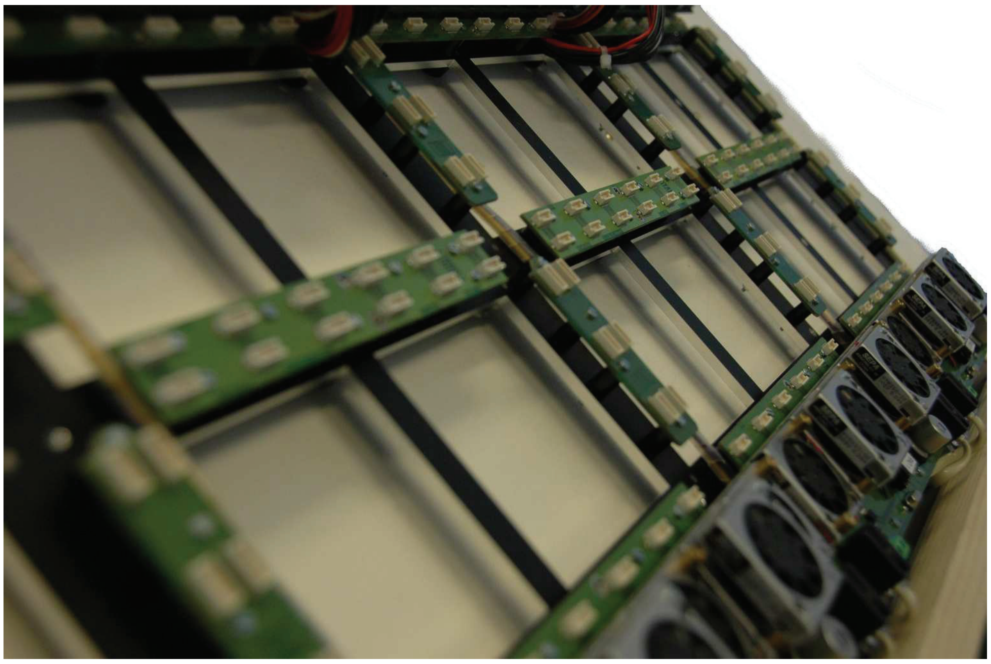


Figure 7.17: *The BioTissue support structure and fan boards.*

a predefined level. Moreover, as the temperature distribution inside the BioTissue depends on the application and on the FPGA speed, each fan can be switched on individually to target the cooling operation to the needed tissue region. It is thus relatively easy to detect thermal hot spots and turn on the necessary fans, a process which is operated by the *BioTissueCtrl* board.

The *BioTissueCtrl*: monitoring and PC interface

Although the whole system can be used in a configuration in which it is completely independent of any external control mechanism, a supervisor board has been developed in order to monitor the whole system and to help during the debugging phase of the system. This board, which lies next to the BioTissue system, as seen in figure 7.14, comprises the following elements:

- A SPARTAN® FPGA which carries out all the executed tasks of this board.
- A Universal Serial Bus (USB) interface chip between the external computer and the *BioTissueCtrl* SPARTAN® FPGA that allows transmission of configurations and data.
- A CAN protocol controller, managed by a microcontroller, that controls the different power supplies and fans of the system.

This board is responsible for the following tasks:

- Transferring FPGA bitstreams from the computer to the BioTissue. Currently, these streams are stored inside a FLASH memory, but other destinations could be conceivable, like on the fly configuration, or storing inside the *ECell* SRAM memory. This functionality is only performed when a requested bitstream is missing inside all BioTissue FLASH memories.
- Transferring data to the *ECell*.
- Getting computed data from the BioTissue in order to process or store them on a computer.
- Monitoring all elements of the BioTissue, in particular switching-on/off of power supplies, and fans.

This interface, unlike those of the BioWall and the BioCube, is a completely passive element that can be removed at any time without blocking the operation of the BioTissue. When it is disconnected, communications between the system and a computer are no longer available, but will resume as soon as this board is plugged-in again to the bottom-left *EStack* board. However, when the *BioTissueCtrl* board is disconnected, temperature monitoring is no longer available, therefore to prevent any overheating, we switch to a preventive mode, in which all fans are switched-on.

7.3 Firmware and software features

The previous section focused on the hardware structure of the entire BioTissue system. We showed that the *ECell*s are embedded in a huge and complex stack of electronic boards with the exception of the *ECell* FPGA, which is reconfigurable, all other *EStack* FPGA bitstreams and microcontroller codes are fixed. The firmware programmed inside these components simplifies the user application's design and adds several features, which will be discussed in this section.

7.3.1 Configuration

The various reconfigurable circuits used in BioTissue are all based on SRAM technology, and can be reconfigured an unlimited amount of times relatively quickly (typically 20 ms). One of the main interesting features of the *ECell* board is the dynamic reconfiguration of its FPGAs, so the system should be able to correctly handle this configuration process and address the correct bitstream to the desired *ECell* FPGA. This task is performed within the *ERouting* board, where each *ERouting* FPGA can access an adjacent 16 Mbits FLASH memory, typically used to store as many as sixteen different *ECell* FPGA configurations or serve as non-volatile memory available for applications. The *ECell* FPGA configuration process can be started by the *ECell* FPGA itself, thus allowing self-reconfiguration, or by another *ECell* FPGA located anywhere in the tissue, which can request new bitstreams to be loaded in one or several FPGAs at the same time. In order to be able to carry out these multiple operations, the FLASH memories need to have *ECell* bitstreams stored inside them. This can be achieved using a computer and the *BioTissueCtrl* external interface, which can modify the content of all FLASH memories located inside the tissue. This is currently the only way to store new applications to be executed by the *ECell*s. Of course, the configurability of the *ERouting* FPGAs coupled with the modularity of the whole system allows almost unlimited versatility in the configuration scheme. For example, it allows the implementation of applications that would update the FLASH contents using external memories, Ethernet or Wi-Fi connection, or retrieve the *ECell* configurations from sources other than the local FLASH memory.

As our SPARTAN®3 FPGA bitstream has a fixed size of $1'047'616$ bits and each application FLASH slot has a 1 Mbit size, 960 bits additional storage space is available for each application. We decided to use this free memory space for storing additional information like a unique ID, application characteristics, name, graphic icon, etc...

7.3.2 Cellular communications

In all cellular systems, connections between adjacent cells are really crucial and need high-capacity buses between them. Although our system can sustain several types of communication architecture or protocol, we made some IP cores that can emulate big sized buses that connect adjacent *ECell*s together. We decided to use high speed LVDS signals like those of the BioCube to ensure the connections. These have the advantage of reducing the number of wires between components. We then provide an IP core which emulates a direct wire link with the four cardinal neighbor *ECell* FPGAs. This very simple core has only one generic parameter n which set the bus width (Fig 7.18). The data multiplexing is similar to that of the BioCube as described on page 83, but is more complex, as the serial stream needs to go through two *ERouting* FPGAs. Hence a specific algorithm assures that no corrupted data can occur between cells due to the imprecision of the clock oscillator of two adjacent cells.

In the example of a cellular automaton where each cell needs to know the state of all its neighbors in order to update its own state, such a of configuration will be realized using an instantiated VHDL module inside each *ECell* FPGA. This module, which needs to be configured by setting the n bus width value, will provide four output buses with a width of n-bits, one in each direction (North, South, East and West) and

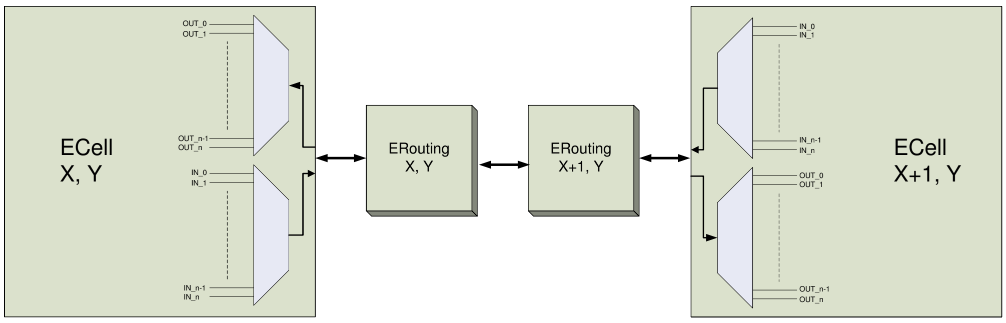


Figure 7.18: *Detail of cellular communication between two adjacent* ECells*.*

four such buses in input. Using the fast 500 Mbit/s lines for multiplexing all these signals, interface frequency for a 64-bit wide bus is 4 Mhz.

These kinds of bus multiplexing and demultiplexing buses allow the same kind of cell communications as for the BioWall, but are not really convenient for the BioTissue architecture. This kind of cellular communication continuously transmits the state of each bus to the next *ECell* module, but is not really optimized in terms of data amount transmitted, as it can only transmit bus changes or others parameters.

The BioTissue communication architecture allows more powerful types of connections between *ECell*'s FPGA, such as packet protocols which will be discussed below, but these are more complex to operate in the case of cellular automata applications.

7.3.3 Routing

While the above-mentioned high-speed links provide only very local communication capabilities, the *ERouting* FPGAs can obviously allow the implementation of more complex communication schemes such as broadcasting or point-to-point communication. Seen as a simple interface from the *ECell*, the routing network capabilities provided by the *ERouting* board are able to implement, at application level, complex data transfers between the *ECell*s. Of course, many different types of networking paradigms exist and could be implemented in our system [87, 140, 8]. As a first realization, we decided to try to use the *Hermes* framework [83], a powerful packet routing system, which provides many interesting functionalities for a relatively low hardware overhead. Available as a VHDL core, five *Hermes* switches may be implemented in each *ERouting* FPGA, providing a bandwidth of 500 MBits/s in every direction. This kind of routing protocol has been tested and is suitable for our BioTissue platform, but is actually not implemented and used by our developed applications (this can be done in the future for new developments on the BioTissue).

Depending on application and performance needs, this type of routing communication or the type previously described based on bus multiplexing can be chosen. Each has its advantages and disadvantages.

7.3.4 Display driving

A really important need for the user developing applications on the BioTissue hardware is to have simple interface for displaying data on the embedded screen. Each *ECell* can access an 8 x 8 screen area located over its PCB board. Several applications do not need to use color luminosity gradation, and switching on or off a LED is sufficient. We then provide a choice of two different IP cores that can be used in the *ECell* FPGA for accessing the display. The first one only gives the ability to address simultaneously each pixel with only the possibilities of switching it on or off. The second one gives the full color range to each pixel, but need to be fixed sequentially, as described below. These two cores are based on the same structure: they have on one side a user interface that fits well with the structure of the 8 x 8 dot display, and on the other side, a specific interface that must be connected to dedicated pins of the *ECell* FPGA. The function of these cores is to encode the display data information, and serialize them to the *ERouting* FPGA, where they will be buffered before being transferred to the *EPower* FPGA. This latter will reconstruct in its internal framebuffer, the image of the 18 *ECell*s needed to be displayed on the screen.

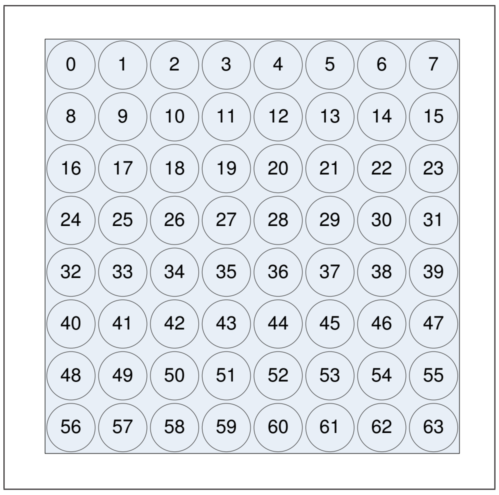


Figure 7.19: ECell *pixel ordering.*

Figure 7.19 shows the user view of the 8 x 8 pixel cell topology. Each one has a unique number and can be addressed individually. Table 7.3 shows the interface signal of the first core, which can only display eight different colors for each pixel (black, red, green, blue, and the colors composed of these three components: orange, pink, cyan and white).

If all of the RGB screen color ranges are needed, the second core, whose interface is described in table 7.4, must be used. Parallel addressing is no longer possible, as this kind of interface will need a 1536 (64 x 24) bits input interface, which would require too many logic gates inside the FPGA once synthesized. This problem is bypassed by using sequential addressing to this display array. Only the modified pixel will change on the screen. If some dots are not updated for a long time, their color will thus remain until new data are written on them. Table 7.5 gives the corresponding signification for each GRAM_adr.

Name	Bus size	Direction	Description
clk	1	IN	50.0 Mhz clock.
reset	1	IN	Module reset signal (active high).
red	64	IN	red signal; one bit for each pixel.
green	64	IN	green signal; one bit for each pixel.
blue	64	IN	blue signal; one bit for each pixel.
erouting_in	1	IN	signals used by this core to synchronize display data with the *ERouting* FPGA.
erouting_out	1	OUT	

Table 7.3: ECell *1 bit display interface.*

Name	Bus size	Direction	Description
clk	1	IN	50.0 Mhz clock.
reset	1	IN	Module reset signal (active high).
GRAM_adr	8	IN	address for the graphical RAM memory.
GRAM_data	8	IN	data for the graphical RAM memory.
we_GRAM	1	IN	write signal; store GRAM_data at GRAM_adr.
erouting_in	1	IN	signals used by this core to synchronize display datas with the *ERouting* FPGA.
erouting_out	1	OUT	

Table 7.4: ECell *8 bit display interface.*

Address	Content
0x00 - 0x3F	red composite. pixel 0 to 63.
0x40 - 0x7F	green composite. pixel 0 to 63.
0x80 - 0xBF	blue composite. pixel 0 to 63.
0xC0 - 0xFF	not used

Table 7.5: ECell *GRAM memory content.*

The data written to this module have an 8-bit range. The value 0 means LED off, and the value 255 (or 0xFF) means full luminosity output. The range between these two values must give all of the intermediate gradation levels in a linear way. Unfortunately, as for most luminary devices, the transfer function between the electrical and optical components of the display system is non-linear. If this non-linearity is not compensated for, high brightness regions are expanded and dim regions are compressed[4]. In section 7.2.3 we described the display as a 30-bit display, with a 10-bit range for each fundamental color. Hence we included in the *EPower* FPGA a gamma[102] correction Lookup Table (LUT) (8-bit wide input, 10-bit wide output), which compensates for the non-linearity of the LED display, and result in a linear transfer function.

7.4 Conclusion and future work

In this chapter, we described a novel hardware platform aimed at the realization of cellular computing applications ranging from massively parallel computing through the exploration of various routing paradigms to bio-inspired computing. The versatility of the platform along with the potential computational power it can provide offer very interesting perspectives for future developments.

For example, should more processing power be needed, the *ECell* boards could easily be replaced by a bigger reconfigurable circuit or a different kind of circuit. Similarly, from the software perspective, the system's modularity implies that few changes would be required, for example, to allow different types of *ECell* boards on the same *ERouting* substrate. Another example is of particular interest in the larger perspective of a prototyping board for unconventional computing: in the current implementation, the *ECell* boards consist of conventional programmable logic, but nothing prevents their replacement with more "exotic" or non-standard units that could potentially be of interest for research in bio-inspired mechanisms. The interest of a modular approach such as the one used for BioTissue would then be in the option of exploiting the routing resources of the *ERouting* substrate to connect the units and, probably more importantly, in the possibility of taking advantage of an existing design environment.

Aside from the computational aspect, the system is also open to several improvements related to I/O aspects. For example, it is clear that the display available on each *EStack* will not be sufficient for many types of application and, in this case, it would also be relatively simple to add an external screen to display more complex *ECell* computations. On a similar note, a planned improvement to the system is the introduction of high-speed I/O boards that, placed on the borders of the array, would allow the implementation of data-intensive applications (video streaming, for example).

In addition to hardware improvements, software tools like routing or interface tools, could be developed to facilitate usage and application development.

Chapter 8

BioTissue applications

WHEREAS the previous chapter focused on the BioTissue hardware description, this one details applications running on this tissue and solve the problem of selecting and starting applications. The BioTissue structure is particularly suited to the implementation of cellular computing applications illustrated by the two following examples: the Game of Life, whose implementation will be compared with the BioWall and the BioCube versions, and the self-replication of a cellular organism with self-healing properties. At the end of this chapter we will take a closer look at the BioTissue architecture to show that it is perfectly adapted to running multiple applications simultaneously.

8.1 μOS: a tiny operating system

Although the hardware structure is now well known, one question remains unanswered: what happens after power-on? We saw that the *EPower* board is responsible for the entire booting process, which means powering and checking the health status of all *EStack* boards. The *ERouting* FPGAs are then be configured with their predefined bitstream. At present the *ECell* FPGAs are still not configured, and no application is running. The start-up process is carried out automatically by forcing *ERouting* FPGAs to configure all *ECell* FPGAs with the bitstream number 0. At this point, we have our first application running on the BioTissue; however each time the BioTissue is started up, this same application will be loaded. This drawback is solved by implementing a small operating system, called μOS, which will allow the selection of different *ECell* FPGA configurations.

Before going any deeper into the explanation of μOS operations, it is important to give a definition of a BioTissue application. An application is responsible for achieving a specific task. This task is implemented within a set of several adjacent *ECell* Field Programmable Gate Arrays (FPGAs), defining a rectangular area called a *window*. Each FPGA can be configured with a specific configuration that can interact with the configuration of its different neighbors.

As each *ECell* FPGA gets the configuration 0 at the startup phase, to launch an application all required *ECell* FPGAs need to be reconfigured with another SPARTAN®3 configuration stored inside a FLASH memory.

8.1.1 Operation and reconfiguration

In the aim of handling and managing all of the *ECell* bitstreams, we created a simple operating system for the BioTissue, which is responsible for the following tasks:

1. Allowing the user to choose an application.
2. Defining a graphical area, called a window, in which the application will run.
3. Giving the user the possibility to kill an application.
4. Receiving reconfiguration commands from *ECell* FPGAs.
5. Computing and memorizing application parameters, like position and window sizes.
6. Transmitting previous memorized values to the reconfigured FPGAs.

Tasks (3), (4) and (6) are crucial and can be executed at any time independently of the application running. Tasks (1), (2) and (5), however, will be executed on request. The separation of these tasks will be detailed in section 8.1.2. First of all, it is important to mention that these requirements make it necessary to modify our firmware structure. We added to the *ERouting* FPGAs some modules which are always running, and which allow them to interact with the *ECell* configuration. These resident modules constitute our operating system. The volatile modules, which are in charge of the tasks (1), (2) and (5), are implemented inside an application named μOS. The operating system and μOS are the topics covered in this section.

The μOS application, which is the first one automatically loaded upon power-on, is an interactive application that will display on the BioTissue screen a list of all applications stored inside the BioTissue FLASH memories, and allow selection of the graphical area on which this one should run. Unlike actual commercial operating systems that can be found on computers or embedded systems, μOS is very limited in terms of functionalities and does not include any fine graphical features, nor any online help.

Figure 8.1 shows an example of the screen once μOS starts. Five 8 x 8 pixel icons are displayed on the bottom-left part of the screen, representing five different applications, as mentioned in the figure. The user presses one of these icons to select the desired application. Two other actions are then needed for selecting on which part of the screen the application should be loaded. The first push selects the bottom-left *ECell* of the applicative area and the second push selects the top-right *ECell*. The result is a window in which each *ECell* will be reconfigured using the selected application FPGA bitstream (Fig 8.2). In this figure, the highlighted zone is the new window in which the selected application will be loaded and run. The left zone, which has not been selected, is still loaded with the μOS application: this zone changed its layout to fit with its new dimensions.

Application parameters and reconfiguration

μOS is a special application: it is the only one able to communicate with a specific module inside the *ERouting* FPGA in order to set parameters used afterwards for application reconfiguration.

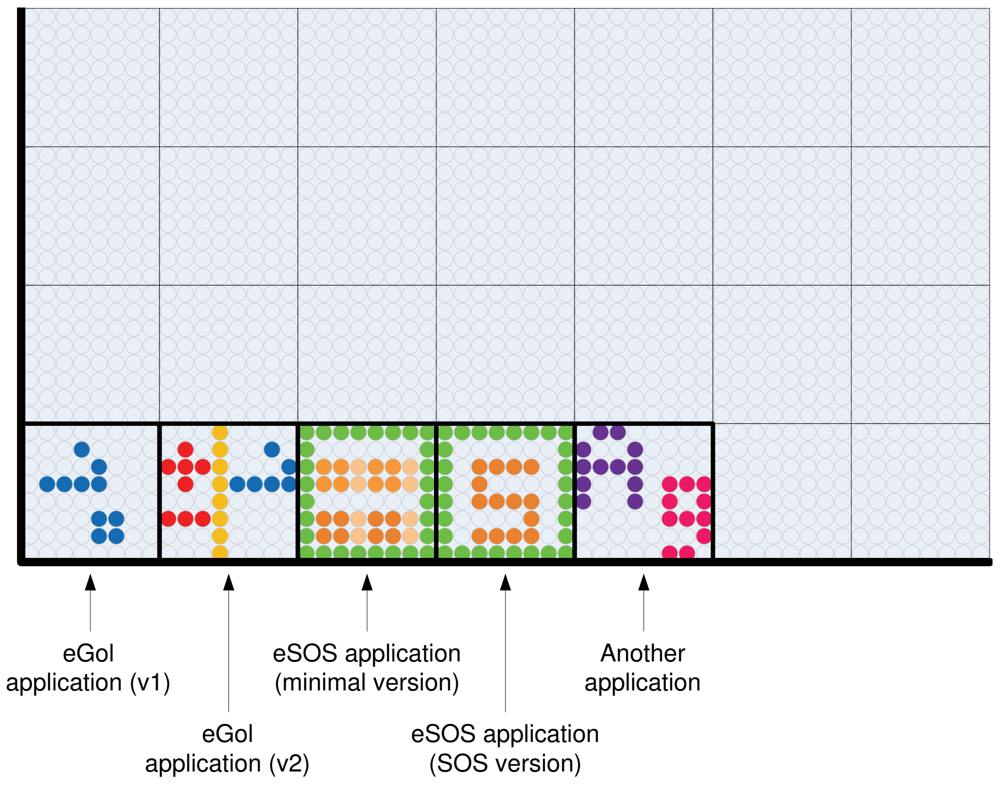

eGol application (v1)
eGol application (v2)
eSOS application (minimal version)
eSOS application (SOS version)
Another application

Figure 8.1: *μOS starting screen (detail of the bottom-left of application window).*

Just before reconfiguring the selected window with the new application, the μOS performs some computations (Table 8.1) whose results are stored inside the *ERouting* FPGA registers, and which are supplied to the next configuration as described in § 8.1.2.

Parameter	Description
X_{glo}, Y_{glo}	Cell absolute coordinates inside the BioTissue.
X_{win}, Y_{win}	Cell absolute coordinates inside the application window.
W_{glo}, H_{glo}	BioTissue width and height.
W_{win}, H_{win}	Application window width and height.

Table 8.1: *μOS computed parameters.*

These parameters are really important for the *ERouting* computational layer, as they will define the borders of the application area. Between two different, adjacent applications, communication channels will be severed in the aim of avoiding error and data corruption between two incompatible applications. This means that all communications between each *ECell* inside the window area of the application are possible, but communications with *ECell*s belonging to another application are impossible.

Once loaded, the new application can still interact with the *ERouting* layer and the

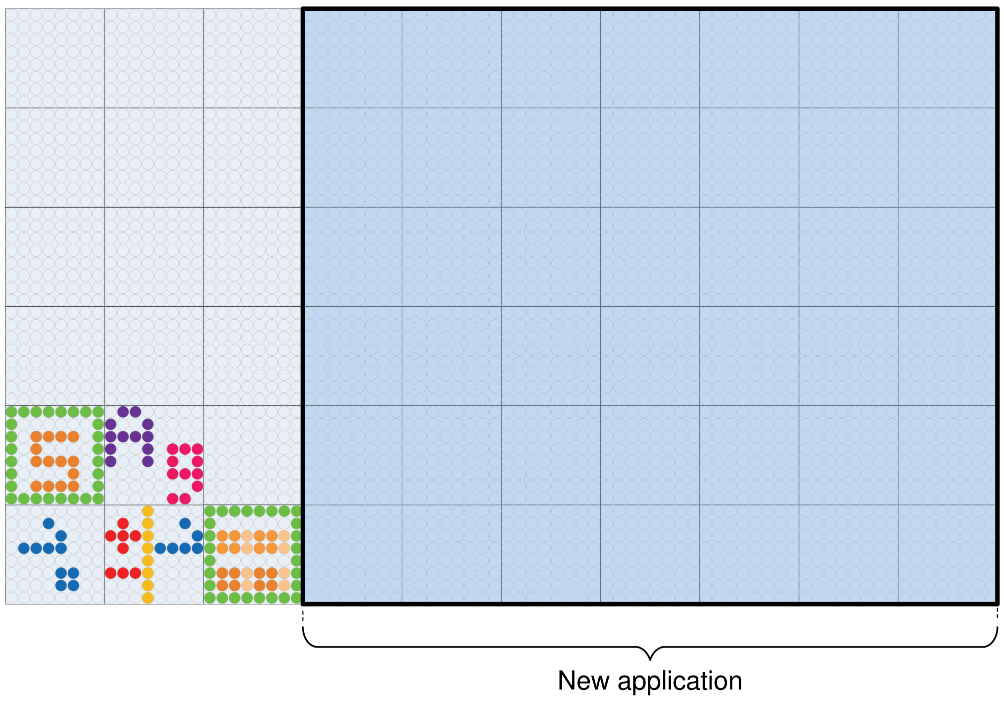


Figure 8.2: *Window delimiting the new application zone; the unselected area still displays the μOS application menu, with a reorganized layout.*

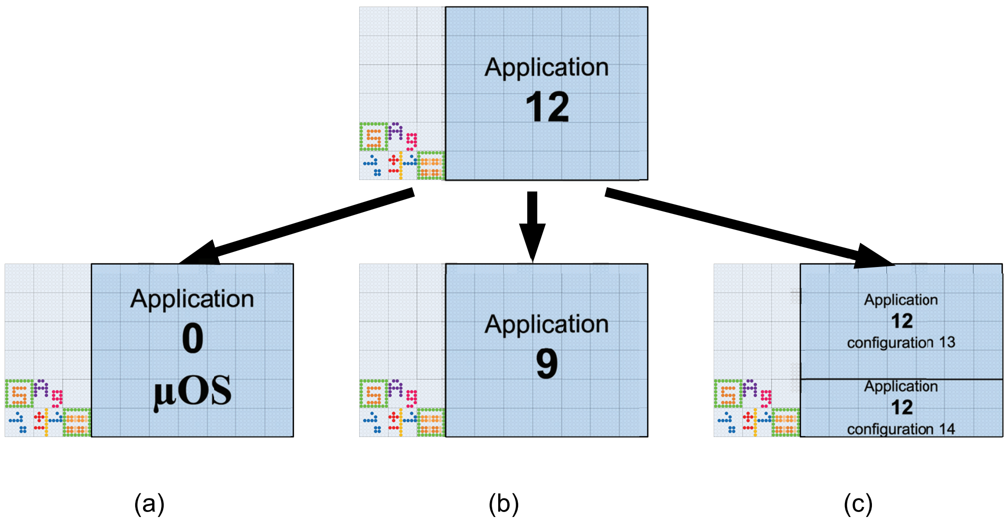


Figure 8.3: *Different types of* ECell *reconfiguration.*

application can request *ERouting* FPGA to change the content of the *ECell* FPGA. In this case, the current application can make several different requests:

- It reconfigures all the *ECell* FPGAs inside the window with μOS (Fig. 8.3(a)). This is similar to ending the application and restarting the μOS with its selective application process. The result of this configuration will be figure 8.1.
- It reconfigures all *ECell* FPGAs inside the defined window with a new application (Fig. 8.3(b)). This is similar to ending the current application and loading a specific new one.
- It reconfigures some *ECell* FPGAs inside the application window with a new configuration (Fig. 8.3(c)). In this case, several different configurations are running inside the application window space, and can communicate with each other using a communication bus. These new configurations set the "child" flag inside the *ERouting* registers, which indicates that a previous "main" configuration launched these new configurations.

8.1.2 μOS architecture

The operating system means that there is a resident module with which an application can interact at any time. The main tasks supported by this module are, as listed in § 8.1.1, to manage reconfiguration schemes and parameter data used by applications (table 8.1). Interpretation of these parameters sets some flags inside the *ERouting* FPGA, which block communications between adjacent *ECell*s. This feature is needed to manage the window concept at the hardware level. A last function, the "kill" feature, is implemented inside this resident module. When the bottom-left cell touchsensor of a window running a defective application is pushed for more than 5 s, a kill process occurs and reconfigures the whole of the selected window with the μOS configuration. This feature can be really useful during developing procedures and allows the termination of an application without resetting all of the BioTissue. Consequently, the *ERouting* FPGAs include only the necessary code needed for performing the above mentioned tasks, whereas the *ECell* FPGAs contain all of the interactive part, i.e. the application selection process, and the coordinate computation of all the parameters mentioned in table 8.1 (Fig. 8.4).

8.1.3 μOS task details

Once the BioTissue is started, all *ERouting* FPGAs configure their respective *ECell* FPGAs with the configuration number 0, labeled μOS. At this point, the μOS application finds all of its missing features inside the *ERouting* resident modules. They then interact in order to choose an load a new application inside a user defined window. The diagram in figure 8.5 shows all of the sequential tasks executed by our operating system.

Initialization

During the first phase, which takes place before an application has been selected by the user, several tasks occur sequentially. The first task is to locate each *ECell* inside the BioTissue, i.e. to give to each *ECell* board an unique number composed of the X- and Y-axis inside the tissue, where the bottom-left cell has the coordinates $0, 0$. This task

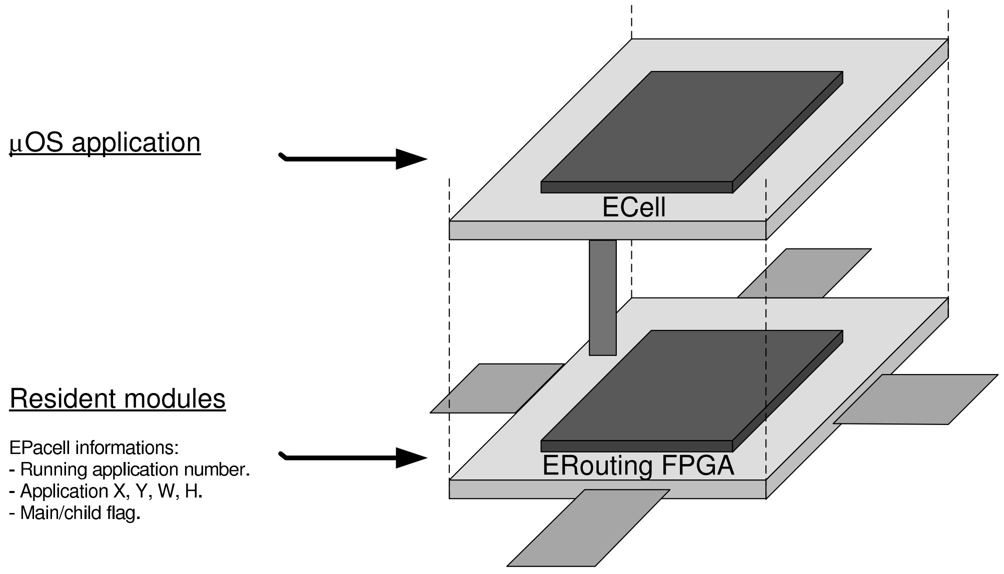


Figure 8.4: *Detailed structure of a BioTissue cell when μOS is running.*

is also executed inside the window. After power-on, there is only one window corresponding to the border of the BioTissue. In figure 8.2, we have two windows, with the first one on the left part of the screen, running μOS, while the right part of the screen will run another application. Here, local X- and Y-axes do not fit with global coordinates: in our example, the *ECell* located at global position $3, 0$ has the $0, 0$ coordinate inside its window. These enumerations are really important in our system, as they will help to manage different applications running simultaneously on the BioTissue. Once this localization process is achieved, μOS scans the contents of each FLASH memory of the *ERouting* layer, and sorts all available applications, whose icons are displayed in a consecutive manner on the screen (Fig. 8.1). If one of these icons is selected using the touchsensor interface, the desired application number is memorized, and the next steps are completed, as illustrated by the flow on the right side of figure 8.5.

Starting an application

The following procedure, executed by a human operator, selects the new window size and the position where the application will be loaded. The first step is to select, by pushing on a touchsensor membrane, the bottom-left corner of the new window. A second touch, which may be on the same sensor or on the top-right of the first selected *ECell*, then completes the procedure to define a rectangular area, setting the window boundaries for the new application. The minimal zone can be as small as an *ECell*, and the maximal size is the current window size, which is the BioTissue size after power-up.

Register file Once an application is selected with its new operation zone, all μOS cells compute the values of table 8.1. These values, which are stored in a register file

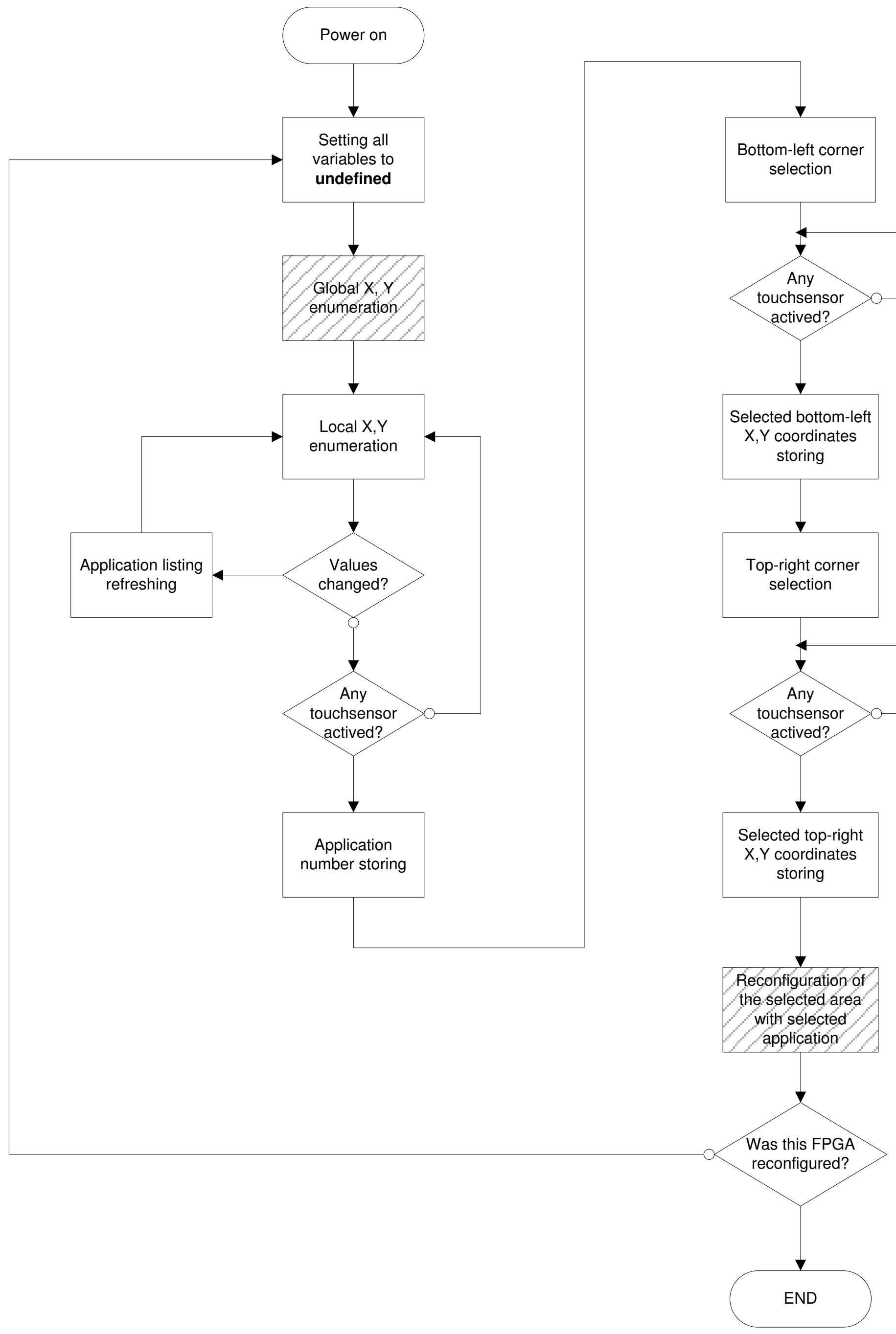

Power on
Setting all variables to **undefined**
Global X, Y enumeration
Local X,Y enumeration
Application listing refreshing
Values changed?
Any touchsensor actived?
Application number storing
Bottom-left corner selection
Any touchsensor actived?
Selected bottom-left X,Y coordinates storing
Top-right corner selection
Any touchsensor actived?
Selected top-right X,Y coordinates storing
Reconfiguration of the selected area with selected application
Was this FPGA reconfigured?
END

Figure 8.5: *μOS simplified functional diagram. Dashed blocks are tasks executed inside the* ERouting *FPGA and all other tasks are done inside the* ECell.

of the *ERouting* FPGA, help to configure the communication channels of the *ERouting* layer. Once the reconfiguration is completed, these same values will be copied from the

ERouting register file to a register file of the *ECell* FPGA. These parameters localize all of the FPGAs inside a window and can help the user in the application development process.

ECell reconfiguration

The last task consists in reconfiguring all of the *ECell* FPGAs from the selected window with the chosen application. After this point, the μOS functions are complete, and this new application becomes autonomous.

A question remains concerning the other FPGAs which were not selected to handle a new application and are still running μOS. The area that is left for these FPGAs is probably reduced, and they thus need to recalculate their boundaries before displaying the list of all applications (Figs. 8.2 and 8.5).

8.1.4 μOS specificity

μOS is the only application which could communicate with applications running in different window contexts. For other application, this is impossible. If two adjacent windows are both running the μOS application, the *ERouting* FPGAs detect this and dissolve the boundaries between these applications, resulting in an single, bigger window, still running the operating system. This feature is crucial, as it is the only way to break window limitations, and enlarge the global window size.

8.1.5 μOS conclusion and future work

This section was focused on detailing μOS, a minimal operating system allowing application selection, reconfiguration, and definition of window working zones.

A correct operation of the BioTissue system depends on the following rule: all *ECell*s always need to run a configuration belonging to one application. Breaking this rule could result in unconfigured *ECell* FPGAs, which are thus no longer controllable. This section shows that this will not occur, as when one application is ended or killed, μOS starts up again, to ensure the reliability of the whole platform.

This operating system application, designed to interact with the resident module of the *ERouting* FPGA, is the first step towards a more powerful and convivial system. Future evolutions of μOS could add nicer graphical interfaces and new features, like moving an application and its window somewhere else on the tissue. Another possible development could be to add pipe communications systems between two applications.

8.2 *eGol*: Game of Life application

In order to compare how the hardware architectures of our three systems (BioWall, BioCube and BioTissue) modify the application's internal structure, we need to have a common application that can run in a similar manner on the three machines. We have chosen the famous Game of Life automaton, as it has already been implemented on the BioWall and the BioCube. Section 4.1 describes the principle of the automaton and the way in which it runs on the BioWall. As we will see in this chapter, the application's implementation on the BioTissue is very similar to its implementation on the BioWall, the main differences being the number of cells inside each FPGA and on the control method used.

This cellular application is designed to implement 64 Game of Life cells in each *ECell*, which means that each pixel gets the state of an automaton cell. In the same way as for the BioWall, each time the touchsensor is pushed, a glider is generated in the selected *ECell*. When using the whole of the BioTissue surface, a maximum number of $6'912$ Game of Life cells can be simulated in parallel, which gives a density four times higher than on the BioWall (cf. § 4.1).

To communicate cell states to neighboring *ECell*s, 24 wires[1] are needed between each FPGA in each direction. Therefore we used the multiplexed bus module described in § 7.3.2 to complete this task.

The main difference between the BioWall and the BioTissue implementations resides in the electronic architecture. The BioTissue is entirely built on a Globally Asynchronous, Locally Synchronous (GALS) topology, which has the advantage of fitting the real biological world, but the disadvantage of being less suited to automata systems, in which a main clock signal is often needed. Unfortunately, the Game of Life is a synchronous system in which each automaton cell must be updated at the same time. The most suitable solution to solve this problem could be to use an asynchronous implementation of the Game of Life [72]. However, even though this approach is compatible with our architecture, we have decided not to use it, as this method is only suited to this automaton, and could not be applied to other cellular systems, nor to any globally synchronous application. Our solution is detailed in the next paragraph.

8.2.1 Automaton synchronization

To solve the synchronization problem we opted for the same methods as used for the BioCube, which emulates a global enable signal, defining the frequency of the automata. In our specific case, the bottom-left cell inside the window defining the running *eGol* area supports this task, and then broadcasts the synchronization signal to all other *ECell* FPGAs running the application. This signal, also called the automaton clock[2], has a 50% duty cycle and starts the automaton computation on the rising edge, updating these outputs on the falling edge. This method ensures that the correct values are computed at each step, but limits maximum speed to a slow frequency because this signal needs to be propagated from the bottom-left FPGA to the top-right FPGA. As seen in the previous chapter, our multiplexed bus modules allow a high number of channels to be transmitted over a single wire, but needs several clock steps to transfer the bus state from a FPGA to its neighbors. The main disadvantage of this methodology is to limit the toggling frequency of this enable signal to a maximum of tens of kilohertz, which exceeds our application's requirements, as visual changes are no longer visible above a hundred hertz.

[1]8 for direct links; 16 for diagonals.

[2]This signal is an enable signal for the main clock, for which the frequency is set inside the BioTissue to 50.0 Mhz.

8.2.2 *eGol* control

Whereas BioWall computing speeds are really easily controlled using the *BioSoft* tool on a Personal Computer (PC), BioTissue applications do not have any other way to control their states but by themselves. Our Game of Life implementation has an embedded control menu that allows the automaton's computation speed and the initial patterns to be changed. Figure 8.6 shows the application running with the control menu displayed in red. This menu automatically overlays in the *eGol* window when a touchsensor inside the application boundary is pushed for more than one second.

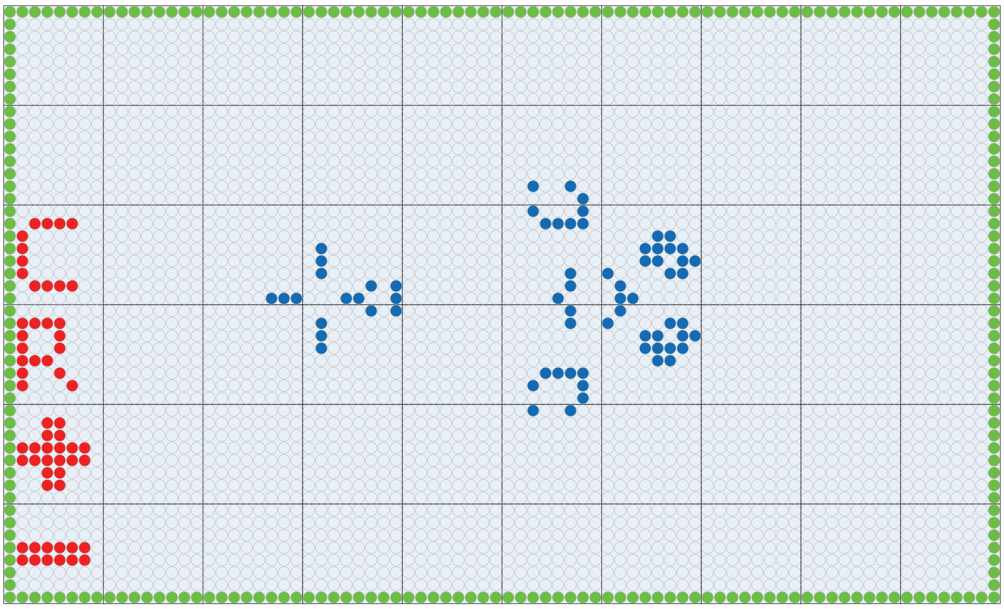


Figure 8.6: eGol *(v1) with a control menu and the game of life automaton.*

Clock speed

Once *eGol* is launched, the automaton's clock is by default set to 1 Hz. Using the *eGol* menu, the "+" and "-" buttons allow changes to its frequency, which ranges from 1 Hz to 1'000 Hz. Frequencies higher than a hundred hertz are not very useful, as the automaton's updates will be too fast for a human user to see. When the application is running at the 1 Hz frequency, the "-" button can be used to set the clock generator to the step-by-step mode (Fig 8.7), where each next pressure on this button generates a single automaton clock step. A simple push on the "+" restores the 1 Hz frequency.

Initial states

The two other buttons in this menu are responsible for setting the initial state of each Game of Life cell. The "R" button generates a global reset signal for all automaton cells and discards all blue pixels. The "C" button configures the Game of Life cells

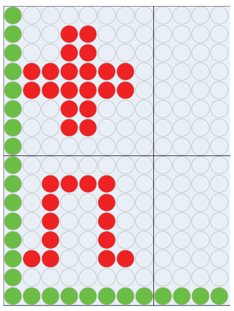


Figure 8.7: *Step-by-step mode.*

with a predefined pattern like the one in figure 8.6. Each successive touch switches between three predefined initial configurations. A push on any other touchsensor disables the menu and restores the sensitive area to its primary use, which allows interaction with the Game of Life automaton.

8.2.3 *eGol* architecture

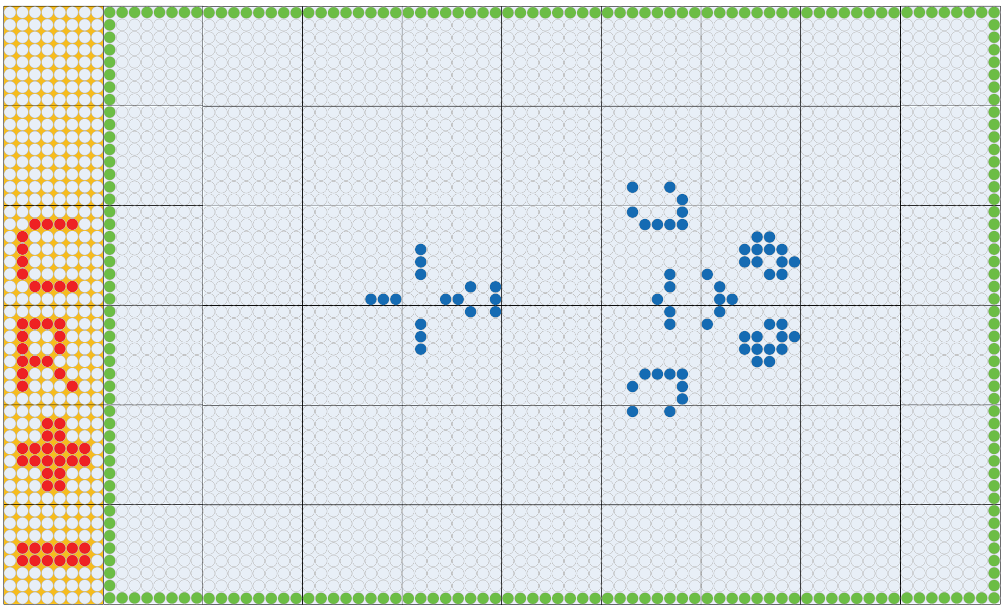


Figure 8.8: eGol *(v2) version with a control menu on the left and the Game of Life automaton inside the green borders.*

The *eGol* application includes, in the same FPGA configuration, two different features:

- the Game of Life automaton, made of 64 cells, and all the communication modules with their neighboring *ECell*s.

- The control menu allowing changes to the automaton clock's frequency and the clearing or loading of predefined patterns.

Thus each FPGA bitstream has the two features, even if the menu module will never be used by the majority of *eGol* FPGAs. Using the BioTissue reconfiguration features, we can assign these two tasks to different FPGAs. Figure 8.8 shows a similar implementation of the *eGol* application, where the left column (with the orange background) is only responsible for the control and all other *ECell*s only contain the Game of Life automaton.

Instead of loading one single bitstream for all the applications (Fig. 8.6), the configuration process is split into two sequential steps. Once this new version of *eGol* is launched by the μOS, all the FPGAs inside the application window run the application labeled number 5 (Fig. 8.9). This application remains loaded for a very short period, while it identifies and differentiates the FPGAs of the left column from the other ones, and reconfigures them with two different configurations. Figure 8.10 shows this specific task, during which the FPGAs on the left receive the bitstream for the control menu (label 7), and the remaining FPGAs receive the application 6, the Game of Life automaton. The result of these multiple steps is a window continually displaying the menu and the automaton as in figure 8.8.

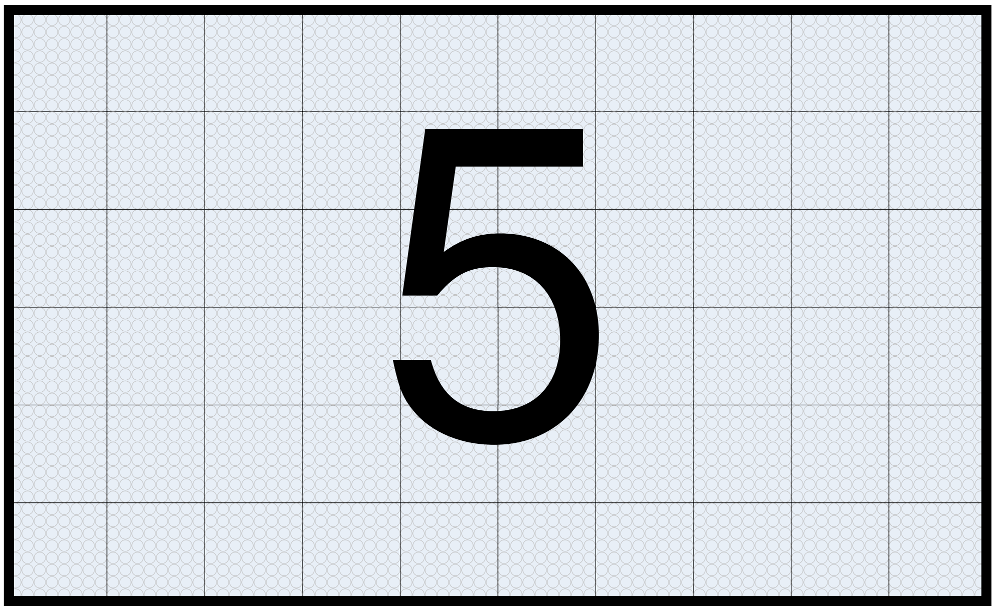


Figure 8.9: *Step 1: differentiation of* ECell*'s FPGAs.*

The main advantage of this procedure is to separate the tasks into two bitstreams and avoid having them present in the same configuration without being used.

Table 8.2 gives an overview of the resources used for the two *eGol* variants.

The main advantage of task separation resides in the possibility of running more complex applications without needing bigger FPGAs.

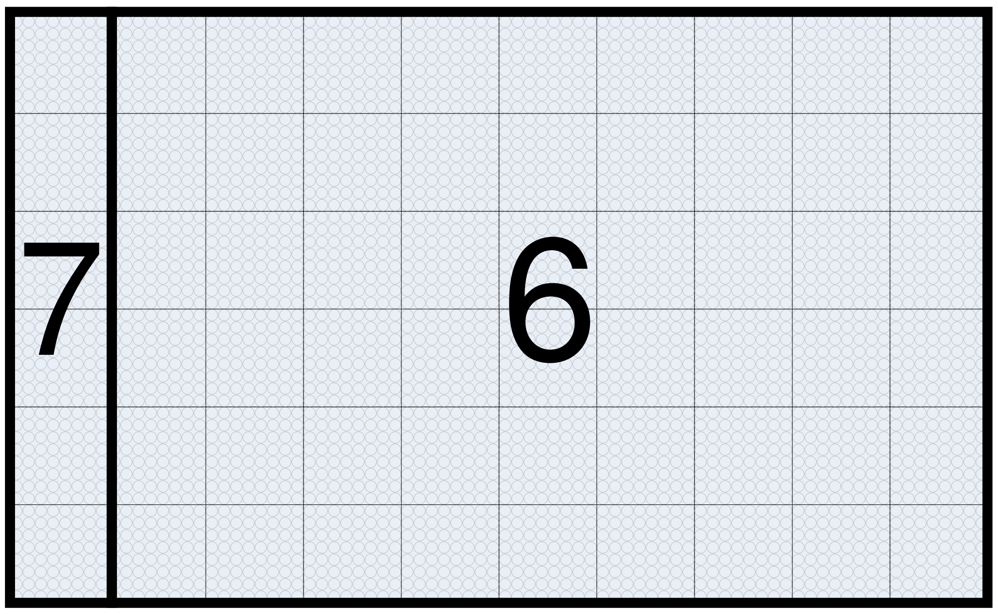


Figure 8.10: *Step 2: the first column gets the control menu configuration; all other FPGAs get the Game of Life automaton bitstream.*

Configuration	LUT used	% FPGA used
eGol v1	3'436	79.53
eGol v2 - configuration 5	187	4.33
eGol v2 - configuration 6 (automaton)	2'053	47.52
eGol v2 - configuration 7 (menu)	656	15.19

Table 8.2: ECell *LUT used for the two* eGol *variants.*

8.2.4 *eGol* conclusion and future work

Further optimizations (both in terms of array size and operating frequency) are indeed possible, but this example was sufficient to illustrate the kind of penalty experienced to achieve global synchronization in a system as large as the BioTissue platform. This penalty is not spurious: it reflects the conflict between the need to design scalable systems and the requirements of global synchronization, and clearly illustrates the need for asynchronous approaches once the size of the systems increases beyond a certain limit. Once again, it is interesting to note how biological systems have evolved to operate without the need for any sort of global synchronization between their components.

8.3 *eSOS*: self-organizing bio-inspired application

Borrowing three structural principles (multicellular architecture, cellular division, and cellular differentiation) from living organisms, we have already shown in previous chapters, how embryonic hardware is able to grow a bio-inspired system in silicon thanks to two algorithms: an algorithm for cellular differentiation, based on coordinate calculations, and an algorithm for cellular division, the Tom Thumb algorithm. Implemented as a Data and Signals Cellular Automaton (DSCA), this Embryonics application is endowed with self-developing properties like configuration, cloning, cicatrization, and regeneration.

The goal of this application, named *eSOS*, is to perform the configuration mechanisms (structural and functional growth), the cloning mechanisms (cellular and organismic growth), the cicatrization mechanism (cellular self-repair), and the regeneration mechanism (organismic self-repair) through simple processes like growth, load, branching, repair, reset, and kill. Starting with a very simple cell made of only six molecules, we will introduce hardware simulations of its DSCA implementation in order to describe these self-developing mechanisms. We will then define a small organism made of three cells, the "SOS" acronym, as an application example for the simulation of our mechanisms

8.3.1 Self-developing mechanisms

Structural configuration

The goal of the *structural configuration mechanism* is to define the boundaries of the cell as well as the living mode or spare mode of its constituting molecules. This mechanism is made up of a *structural growth process* followed by a *load process*. For a better understanding of these processes, we apply them to a minimal self-developing cell. This cell is made up of six molecules arranged as an array of two rows by three columns, the third column involving two spare molecules dedicated to self-repair (Fig. 8.11).

The *growth process* starts when a *growth signal* is applied to the lower left molecule of the cell at time $t = i$ (Fig. 8.11). Every four time steps ($t = i{+}1, i{+}5, ...$), according to the *structural configuration data* or *structural genome*, a molecule of the cell selects one of its four data inputs (Fig. 8.12) in order to create a data path among the molecules of the cell. This path allows a copy of the structural data to be captured in the memory positions of the cell and enables another copy of the structural data to move around the cell. At time $t = i + 24$, when the connection path between the molecules is about to close, the lower left molecule delivers a *close signal* to the nearest left neighbor cell. The path is really closed at time $t = i + 25$.

The *load process* is triggered by the *close signal* applied to the lower right molecule of the cell at time $t = i$ (Fig. 8.13). At each time step ($t = i + 1$ to $t = i + 3$), a *load signal* will then propagate westward and northward through the cell so that each of its molecules acquires a *molecular mode* (Fig. 8.14) and a *molecular type* (Fig. 8.15). At time $t = i{+}4$, we finally obtain an homogeneous tissue of molecules defining both the boundaries of the cell and the position of its *living mode* and *spare mode* molecules. This tissue is ready to be configured by the functional configuration data.

Functional configuration

The goal of the *functional configuration mechanism* is to store, the functional data needed by the specifications of the current application, in the homogeneous tissue al-

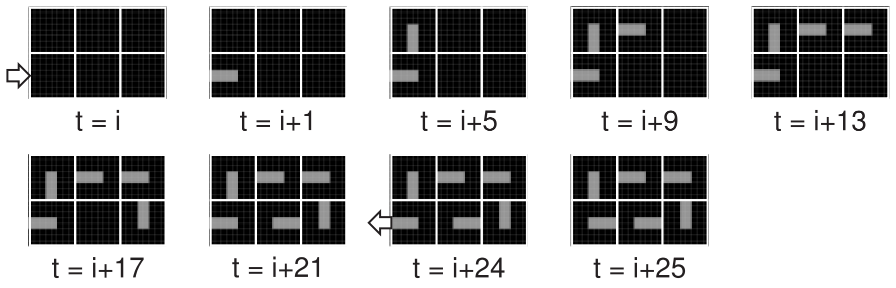


Figure 8.11: *Structural growth process of a minimal self-organizing cell made up of six molecules when a growth signal is applied to the lower left molecule at time $t = i$; when the path is about to close ($t = i + 24$), the lower left molecule delivers a close signal.*

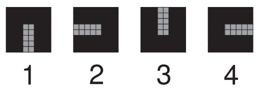


Figure 8.12: *Data input selection. (1) Northward. (2) Eastward. (3) Southward. (4) Westward.*

ready containing structural data (Fig. 8.13, $t = i + 4$). This mechanism is a *functional growth process*, performed only on the molecules in the *living mode* while the molecules in the *spare mode* are simply bypassed. Every four time steps ($t = i + 1, i + 5, ...$), according to the *functional configuration data* or *functional genome*, a path is created among the molecules of the cell in order to capture a copy of the functional data in the memory positions of the cell and to move another copy of the functional data around the cell. At time $t = i + 17$ (Fig. 8.16), the final cell is made up of four living molecules organized as an array of two rows by two columns, while one row of two spare molecules is bypassed. The final specifications of the cell under construction are now stored as functional data in its living molecules.

Cloning

The *cloning* or *self-replication mechanism* is implemented at the cellular level in order to build a multicellular organism, and at the organismic level in order to generate a population of organisms. This mechanism supposes that there exists a sufficient number of molecules in the array to contain at least one copy of the additional cell or of the additional organism. It corresponds to a *branching process* which takes place when the structural and the functional configuration mechanisms deliver northward

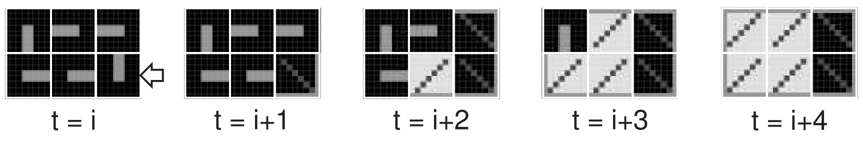


Figure 8.13: *Triggered by the close signal of the nearest right neighbor cell ($t = i$), the load process stores the molecular types and modes of the artificial cell.*

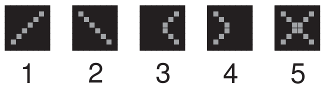


Figure 8.14: *Molecular modes. (1) Living. (2) Spare. (3) Faulty. (4) Repair. (5) Dead.*

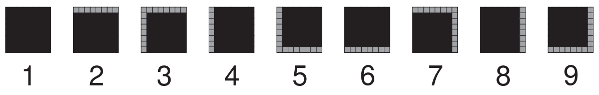


Figure 8.15: *Molecular types. (1) Internal. (2) Top. (3) Top-left. (4) Left. (5) Bottom-left. (6) Bottom. (7) Top-right. (8) Right. (9) Bottom-right.*

and eastward growth signals to the borders of the cell during the corresponding growth processes (Fig. 8.17).

Cicatrization

Figure 8.16, at time $t = i + 17$, shows the normal behavior of a healthy minimal cell, i.e. a cell without any faulty molecule. A molecule is considered as faulty, or in *faulty mode*, if some built-in self-test detects a lethal malfunction. Starting with the normal behavior of figure 8.16 ($t = i + 17$), we suppose that two molecules will become suddenly faulty (Fig. 8.18, $t = i$): (1) The lower left molecule is in *living mode*. (2) The upper right molecule is in *spare mode*. While there is no change for the upper-right molecule, which is just no longer able to play the role of a spare molecule, the lower-left one triggers a *cicatrization mechanism*. This mechanism is made up of a *repair process* involving *repair signals* (Fig. 8.18, $t = i + 1$ to $t = i + 2$) followed by a *reset process* performed with *reset signals* ($t = i + 4$ to $t = i + 6$). This tissue, comprising now two molecules in *faulty mode* and two molecules in *repair mode*, is ready to be reconfigured by the functional configuration data. This implies a *functional growth process* bypassing the faulty molecules (Fig. 8.19).

Regeneration

Our minimal self-developing cell comprises a single spare molecule per row and tolerates therefore only one faulty molecule in each row. A second faulty molecule in

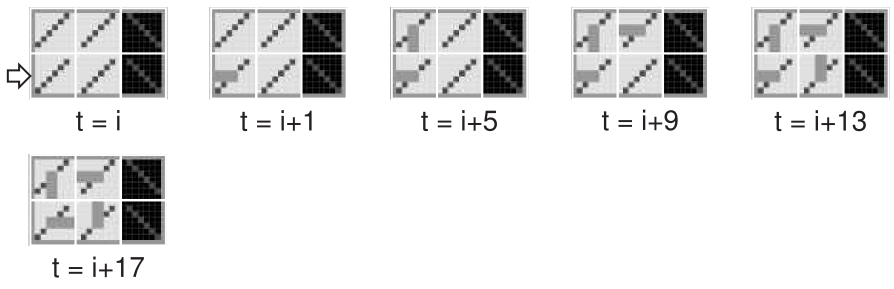


Figure 8.16: *Functional configuration of the living molecules.*

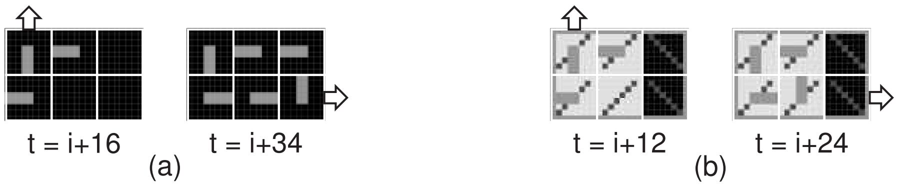


Figure 8.17: *Northward and eastward growth signals triggering the cloning mechanism. (a) Structural branching processes. (b) Functional branching processes.*

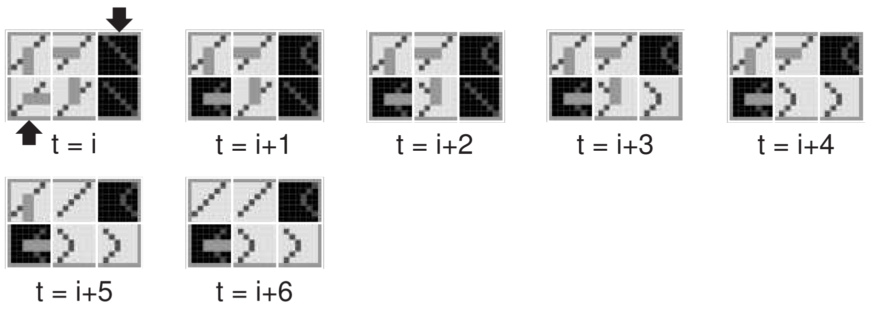


Figure 8.18: *Cicatrization mechanism performed as a repair process ($t = i+1$ to $i+3$) followed by a reset process ($t = i+4$ to $i+6$).*

the same row will trigger the death of the whole cell, and the start of a *regeneration mechanism*. Figure 8.20 illustrates the *repair process* and *kill process* involved in this mechanism. Starting with the normal behavior of the cicatrized cell (Fig. 8.19, $t = i + 17$), a new molecule, the upper middle one, becomes faulty. In a first step, the new faulty molecule sends a *repair signal* eastward, in order to look for a spare molecule, able to replace it ($t = i + 1$). In a second step, the supposed spare molecule, which is in fact a faulty one, enters the lethal *dead mode* and triggers *kill signals* northward, westward and southward ($t = i + 2$). In the next three steps, the other molecules of the array are given the *dead mode*. At time $t = i + 5$, the original minimal cell is dead.

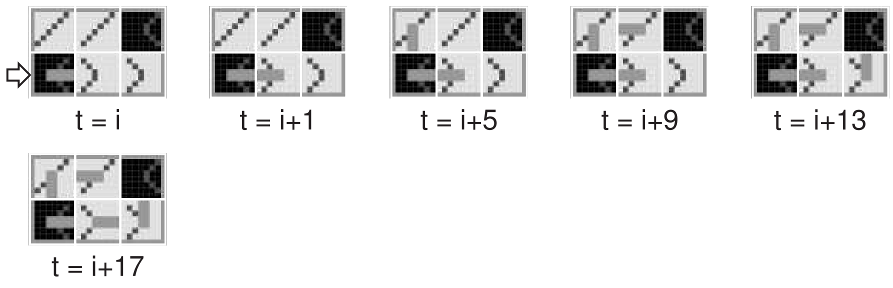


Figure 8.19: *Functional reconfiguration of the living and repair molecules.*

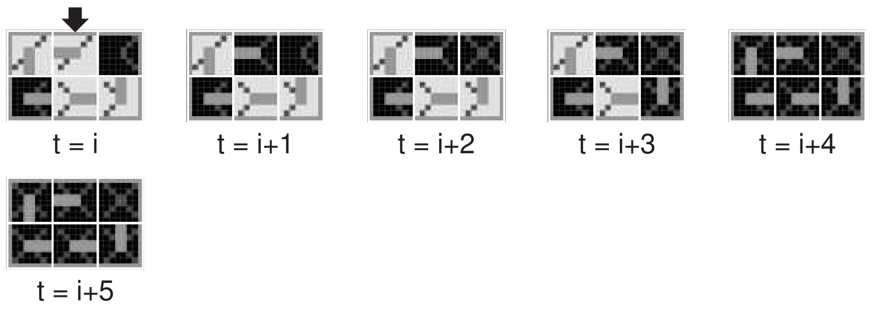


Figure 8.20: *Regeneration mechanism performed as a repair process ($t = i + 1$) followed by a kill process ($t = i + 2$ to $i + 5$).*

8.3.2 SOS acronym organism

Structural configuration, functional configuration and cloning

Although our final goal is the self-development of complex bio-inspired systems, in order to illustrate its basic mechanisms we will use an extremely simplified application example: the display of the "SOS" acronym. The system that displays the acronym can be considered as a one-dimensional artificial organism composed of three cells (Fig. 8.21). Each cell is identified by a X coordinate, ranging from 1 to 3. For coordinate values $X = 1$ and $X = 3$, the cell implements the S character, for $X = 2$, it implements the O character. Such a cell, capable of displaying either the S or the O character, is a *totipotent cell* comprising $4 \times 6 = 24$ molecules. An incrementer implementing the X coordinate calculation is embedded in the final organism.

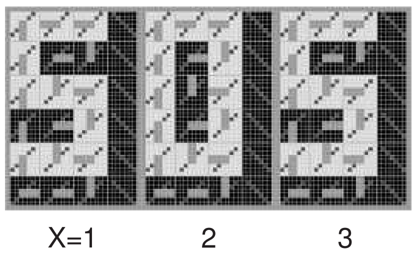


Figure 8.21: *One-dimensional organism composed of three cells resulting from the structural configuration, functional configuration and cloning mechanisms applied to a totipotent cell.*

In order to build the multicellular organism of figure 8.21, the structural configuration mechanism, the functional configuration mechanism, and the cloning mechanism are applied at the cellular level. Starting with the structural and functional configuration data of the totipotent cell, these mechanisms generate successively the three cells $X = 1$ to $X = 3$ of the "SOS" organism.

Cicatrization and functional reconfiguration

The cicatrization mechanism (or cellular self-repair) occurs when one column of spare molecules is introduced into each cell (Fig. 8.21), defined by the structural configuration of the totipotent cell, and the automatic detection of faulty molecules. Thanks to this mechanism, each of the two faulty molecules of the middle cell (Fig. 8.22) is deactivated, isolated from the network, and replaced by the nearest molecule to the right, which will itself be replaced by its nearest neighbor to the right, and so on until a spare molecule is reached. The functional reconfiguration mechanism then kicks in, in order to regenerate the O character of the "SOS" organism. As shown in figure 8.22, the regenerated character presents some graphical distortion.

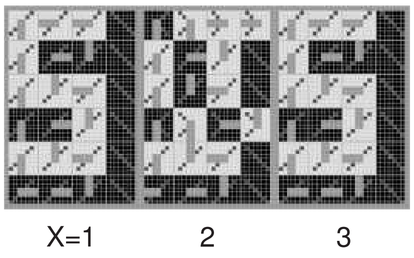


Figure 8.22: *Graphical distortion resulting from the cicatrization and reconfiguration mechanisms applied to the middle cell of the organism.*

Regeneration

The totipotent cell of the "SOS" organism containing only one spare column allows only one faulty molecule per row. When a second one is detected, the regeneration mechanism (or organismic self-repair) takes place and the entire column of all cells to which the faulty cell belongs is considered faulty and is deactivated (column $X = 2$ in figure 8.23; in this simple example, the column of cells is reduced to a single cell). All the functions (X coordinate and configuration) of the cells to the right of the column $X = 1$ are shifted one column to the right. Obviously, this process requires as many spare cells to the right of the array as there are faulty cells to repair. As shown in figure 8.23, the reparation of one faulty cell requires one spare cell to the right and leaves a scar in the "SOS" organism.

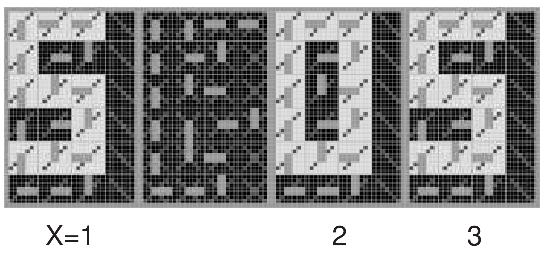
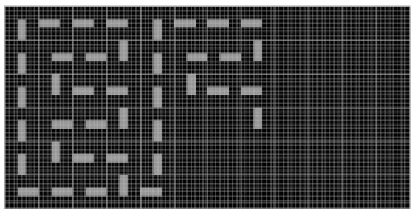
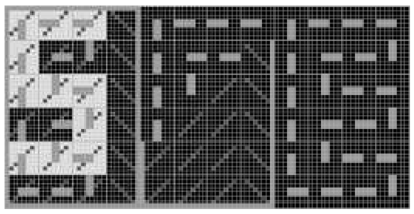


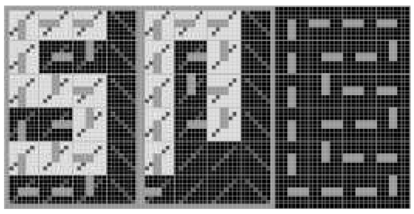
Figure 8.23: *Scar resulting from the regeneration mechanism applied to the organism.*

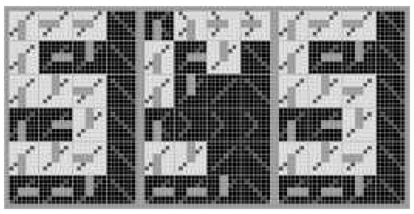
Basic processes

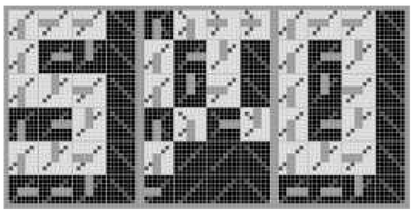
As defined in § 8.3.1, the self-developing mechanisms are made up of basic processes like growth, load, repair, reset and kill. Figure 8.24 illustrates these processes while they are performed on the middle cell of the “SOS” organism. During the structural configuration mechanism, the growth of the data path of the middle cell (Fig. 8.24(a)) leads to the definition of the boundaries as well as the mode of its constituting molecules (living or spare) (Fig. 8.24(b)). The growth process of the functional configuration mechanism expresses the O character which is part of the specifications of the current application (Fig. 8.24(c)). The reset process following the repair of the cicatrization mechanism (Fig. 8.24(d)) allows a functional regrowth process which bypasses the faulty molecules (Fig. 8.24(e)). The detection of a second faulty molecule in the third lower row of the middle cell triggers the kill process of the regeneration mechanism (Fig. 8.24(f)).


(a)


(b)


(c)


(d)


(e)

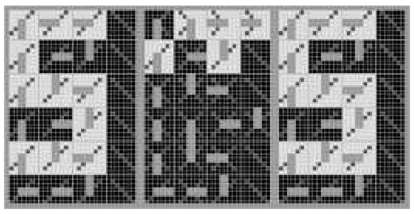

(f)

Figure 8.24: *Processes performed on the middle cell. (a) Structural growth. (b) Load. (c) Functional growth. (d) Repair and reset. (e) Functional regrowth. (f) Kill.*

8.3.3 *eSOS* application

The BioTissue platform comprises an amount of 108 *ECell*s organized as a rectangle of 6 rows by 18 columns. Using one *ECell* for each molecule, the complexity of the platform thus allows the implementation of four and a half totipotent cells of the “SOS” application.

Figure 8.25 shows the cloning of the totipotent cell in order to build a first multicellular “SOS” organism and sketches the cloning of this organism in order to define a population of them. The cloning of the organism rests on two assumptions: firstly, that there are a sufficient number of spare cells in the array to contain at least one copy of the additional organism, an assumption which is only partially satisfied here. Secondly, that the calculation of the coordinates produces a cycle $X = 1 \rightarrow 2 \rightarrow 3 \rightarrow 1$ implying $X{+} = (X + 1)$ mod 3. Given a sufficiently large space, the cloning of the organism could be repeated for any number of specimens in the X and/or Y axes.

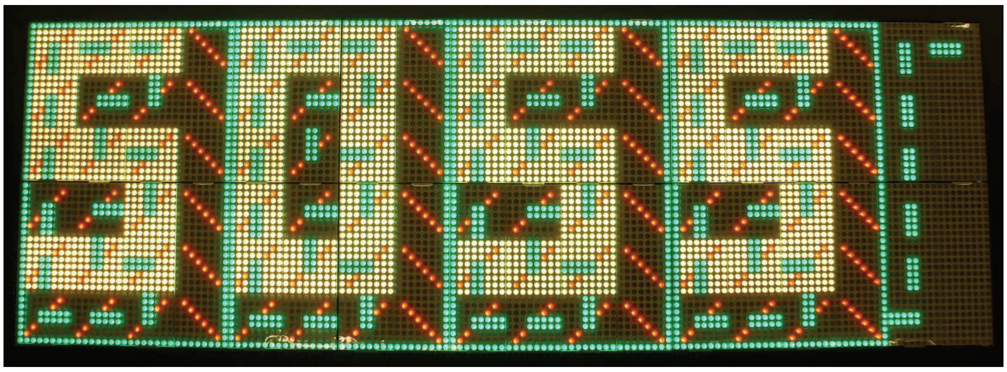


Figure 8.25: *Cloning of the "SOS" acronym, totally realized at the cellular level and partially achieved at the organismic level, on the BioTissue.*

Figure 8.26 illustrates cicatrization or repairs at the cellular level and regeneration or repair at the organismic level. The cicatrization of the cells containing at most one faulty molecule in each of their rows causes the graphical distortion of the characters S and O. The regeneration of the cell containing more than one faulty molecule in one of its rows leaves a scar in the "SOS" organism.

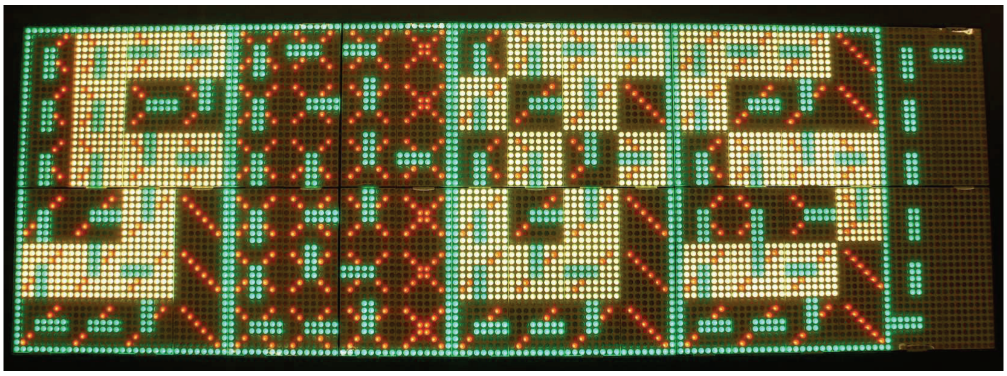


Figure 8.26: *Cicatrization and regeneration of the "SOS" acronym on the BioTissue.*

8.3.4 *eSOS* conclusion and future work

The self-developing mechanisms are mainly based on the Tom Thumb algorithm. These mechanisms are made of simple processes like growth, load, branching, repair, reset, and kill. They equip the cellular systems with bio-inspired properties such as:

- Cloning or self-replication at cellular and organismic levels.
- Cicatrization or self-repair at the cellular level.

- Regeneration or self-repair at the organismic level.

Starting with a very simple cell made of only six molecules, we realized hardware simulations of the application's DSCA implementation in order to describe these self-developing mechanisms. The "SOS" acronym, a small organism made of three cells, was introduced as an application example for the simulation of our mechanisms.

In order to improve this application, additional features could be investigated, for example:

- Automatic detection of faulty molecules, erroneous configuration data, and application dysfunction.
- Asynchronous implementation at the organismic level and synchronous implementation at the cellular level.

8.4 Conclusion

This chapter has described two bio-inspired applications and a simple operating system. Even if the BioTissue can run all kinds of applications, this is not an ideal platform for globally synchronous problems. Before developing new applications, the user should determine if these applications could be built using asynchronous communications. An asynchronous implementation would be more elegant, and more efficient, as it will be compatible with the GALS hardware architecture.

The window concept defined at the beginning of this chapter lets us split the BioTissue surface in several distinct areas, allowing different applications to run at the same time. At present, once a window is defined, all communications and interactions between FPGAs can only be done inside this window. Thus, each application can run at a different speed, manage buses in different ways, without disturbing other applications. Such a system prevents hazardous behaviors, and avoids malicious applications disturbing running ones.

Figure 8.27 shows an example of these possibilities with the *eSOS* application running on the left and right of the BioTissue, while *eGol* runs in the middle. If two adjacent windows are running the same application, they behave like two distinct windows with no communication and interaction possibilities between both of them. Thus, each one can be in a different state and tolerate different speeds. Future evolutions of the operating system could allow communication channels, or pipes between applications.

Finally, it should be noted that the complexity of the *eSOS* application based on the Tom Thumb algorithm with self healing properties was impossible to run on the simplified architecture of the BioWall, and thus constituted a motivation for designing and building a more complex system, the BioTissue.

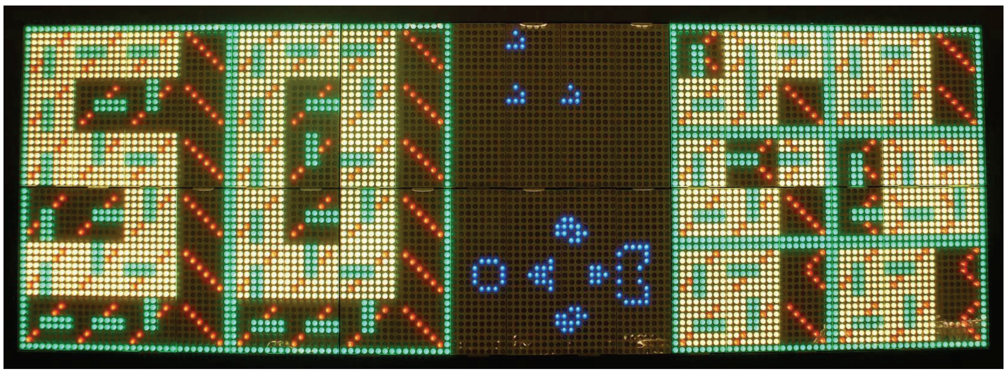


Figure 8.27: *The BioTissue running three different applications simultaneously:* eSOS *on the left with two cells of the “SOS” acronym,* eGol *in the middle, and again* eSOS *on the right with six cells of the minimal structure.*

Chapter 9

Towards electronic paper

It is the year 2054. A man sitting in a subway coach unfolds a piece of newspaper-sized electronic display. News headlines and advertisements start to flash in front of him…

This scene takes place in the science fiction movie *Minority Report*. It is exactly this concept of flexible and lightweight electronic paper that is cheap and light enough so that it can replace the current paper version of newsprint, that many researchers around the world are working to make a reality. Our future will certainly include several products made of electronic paper: newspapers as presented in *Minority Report*, thin and flexible laptops, wearable cellphones or computers integrated in clothes, and many new uses which need to be discovered. Technologies needed to build such an electronic paper device are not yet ready. However in this chapter, we show that much existing research could be combined to move towards a viable implementation.

Linking the results we have gotten in our thesis to the electronic paper concept constitutes the next and final step of our project. In the next section, we will describe in detail the concepts and the structure of the future electronic paper device, and show that the electronic paper terminology is misused in the media and marketing to describe what we call paper-like displays. Section 9.2 will give an overview of the technologies currently being developed by several researchers through the world and which will be needed for developing and manufacturing an electronic paper device based on the novel architecture that we will describe in section 9.3, the BioTissue.

9.1 What is electronic paper?

Electronic paper, also called e-paper, is a display technology designed to mimic the appearance of ordinary ink on paper. Unlike traditional displays, electronic paper can be crumpled or bent like traditional paper and is also capable of holding text and images indefinitely without drawing electricity. According to this definition, electronic paper refers to the display itself. For being usable, such a display needs to be driven by a microprocessor, probably with memory components, interfaces like a keyboard and a memory card reader. In the next paragraphs we will introduce a new terminology to name two products based on electronic paper displays, but mainly different in their physical structure and conception.

9.1.1 Products using a paper-like display

Nowadays, displays able to hold information once their power is shutdown are already on the market. Most of them are rigid and can not be bent, however flexible displays exist as prototypes and will be soon produced. The Readius® reader (Fig. 9.1) is a new Personal Digital Assistant (PDA) the screen of which can be bent and rolled around its case. This device needs a limited amount of power, since its display consumes energy only for changing its contents. A battery is situated in the case along with all the electronics needed to complete the features of the PDA. Many new products similar to the Readius® reader will be developed in the next years [73], and probably a device similar to the electronic newspaper seen in the *Minority Report* movie. Nevertheless, there is always the need for a rigid case holding batteries, microprocessors and interface electronics generally placed on one edge of the display (Fig. 9.2).

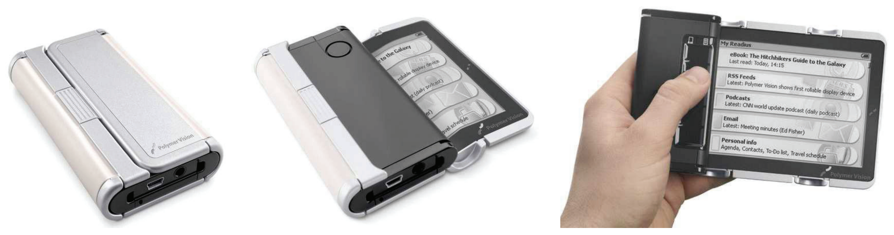


Figure 9.1: *The Readius® reader manufactured by Polymer Vision®. Source:[99]*

This concept does not fit my expectations of electronic paper: in my mind, electronic paper refers to a device looking like a sheet of paper endowed with some new features such as interactive display and processing capabilities. Consequently I have decided to name "products using paper-like displays" all of the current products using such an architecture.

9.1.2 Electronic paper

If "products using paper-like displays" are not electronic paper, then what is my definition? Firstly, if we think about paper, we can imagine a rectangular sheet, thin, flexible, with some text or pictures drawn on its surface. This paper sheet will be light and could be totally crumpled up. This means that the electronic sheet will be a display like the one described above, housing in its internal structure all the electronics, batteries and communication interfaces needed for its use (Fig. 9.3).

Many products could be built using electronic paper: from the newspaper, to the successor of the laptop, through new applications where a computer could be embedded into clothes [45].

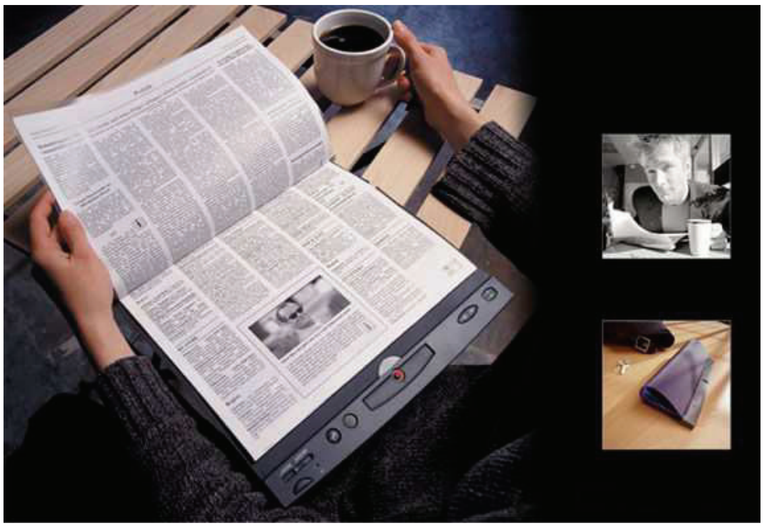


Figure 9.2: *Electronic newspaper; control case at the bottom with its flexible display. Source: IBM.*

Figure 9.3: *The future electronic newspaper will look like the traditional newsprint.*

9.2 Current research

In this section, we will present the research done in different fields, the results of which could be used for building a complete electronic paper machine, with a tactile interface on its surface, a thin display that could be bent, a source of energy, and the computing power to perform the requested tasks.

9.2.1 Displays

Multitudes of display techniques developed over flexible substrates exist. We will only focus on the most appropriated one, a technique developed by E Ink® and based on ultra small colored balls.

Figure 9.4: *Color screen, thin and bendable like a paper sheet. Source: Sony®.*

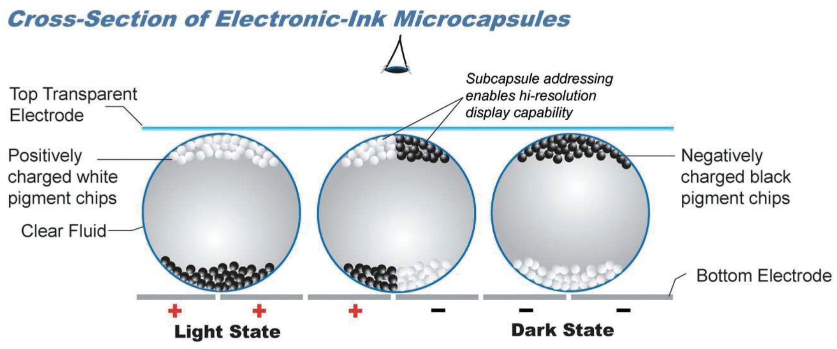

Cross-Section of Electronic-Ink Microcapsules
Top Transparent Electrode
Subcapsule addressing enables hi-resolution display capability
Positively charged white pigment chips
Clear Fluid
Negatively charged black pigment chips
Bottom Electrode
Light State
Dark State

Figure 9.5: *Cross section of E Ink® microcapsules. Source: E Ink®.*

Each tiny capsule contains white granules suspended in a dark, oily liquid (Fig. 9.5). When an electrode in the upper surface is given a negative charge, it attracts granules towards it, making the surface appear white. By reversing the polarity, the granules are pulled to the bottom, revealing the dark liquid and making the surface appear black. The spaces between electrodes are small enough to give a resolution of 300 monochrome Dots per Inch (DPI). To create a full color display (Fig. 9.4) a fine colored filter is laid across the top of the monochrome display - the same trick that lends color to Liquid Crystal Displays (LCDs). The filter makes each pixel appear either red, green or blue when the pixel below is white. When the pixel is black, the filter above reflects very little light so that no color is seen.

9.2.2 Input interfaces

Currently many techniques already exist and may be used for sensing the surface of our electronic display. The most adequate will be the capacitive sensors similar to those developed by Quantum Research [113], which could enable multi-touch screen behaviors like in the demonstrator of Perceptive Pixel [93].

9.2.3 Energy

The problem of energy storage is crucial, and several research are on their way to increase the energy capacity of batteries, while decreasing their size. Very thin and bendable batteries already exist and could be used for our needs.

Other important research involve energy generation. Solar cells could be good candidates for electronic paper. The back of the paper, which is not used, could be covered in flexible solar cells [18] allowing the battery to be recharged when using the e-paper near a luminous source like sunlight. Along the same lines, Pasquier and al. [90] have proposed a new kind of solar cell, made of carbon nanotubes, which has the main advantage of being transparent. Such a solar cell could be added on the top of the display, and power the electronic paper and/or recharge its battery.

9.2.4 Electronic components

A potential headache is the manufacturing of electronic components onto flexible substrates. The knowledge of techniques based on silicon Complementary Metal-Oxide-Semiconductors (CMOSs) is useless, as it needs a hard silicon wafer as a substrate. Much promising research is going on and interesting results have been obtained with transistors constructed on plastic substrates.

9.2.5 Manufacturing methodology

Since the manufacturing processes for building Integrated Circuits (ICs) are not adequate for e-paper, some researchers are looking for new ways to build IC and their transistors. Inkjet printers gave some really good hopes and results building transistors on flexible substrates [122]. Transistors are nevertheless extremely large compared to the nanometer scale and their performance is closer to that of the first silicon transistor.

Some other new manufacturing techniques have appeared notably the roll-to-roll methodology used by SiPix® to produce displays for electronic paper based on a technique similar to E Ink®. On a continuous plastic substrate, the display is created by coating, embossing, sealing and lamination processes. Figure 9.6 details this manufacturing process and the structure of the display. This method meets the high needs for producing a large quantity of display material that is reliable, extremely thin, and flexible.

9.3 The **BioTissue**: a basis for a new electronic paper architecture

In this section, we will describe the principle feature of the architecture of our electronic paper. Its structure will be mainly suited for the applications needing a computer structure; in fact, our electronic paper could be able to do the job of a laptop, once software is available.

Currently, all traditional systems using a microprocessor are built with several components placed on a Printed Circuit Board (PCB): the microprocessor itself, the memory, the power regulators, the display controller, the sensors... This structure consists of several completely different interconnected functionalities. However such an

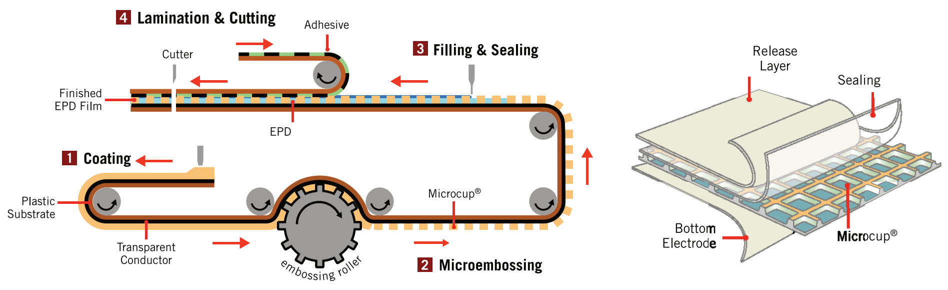


Figure 9.6: *(a) Roll-to-Roll manufacturing process used by SiPix® to produce their flexible display. (b) Structure of the SiPix® display. Source: [114]*

architecture does not easily fit into an electronic paper device. The other way would be to construct an electronic layer containing of these components. Of course, printed technology could be used, but may have the disadvantage of reducing performance since the microprocessor needs a rather large surface, much larger than its silicon counterpart. Once this electronic layer is manufactured, three additional layers should be glued on top of it: a thin battery at the bottom, a display on the top, and a tactile interface above the display with the connections between these four layers located on one edge.

This approach, has several disadvantages: printed technology will be difficult to use, performance may be reduced due to the increases size of printed components. Care should be taken with such electronic paper, it there is no fault-tolerant mechanism: if the foil is carelessly bent or crumpled a bit too much, irreversible failures could happen, since some transistors or connections between layers could break, and the whole sheet would be destroyed.

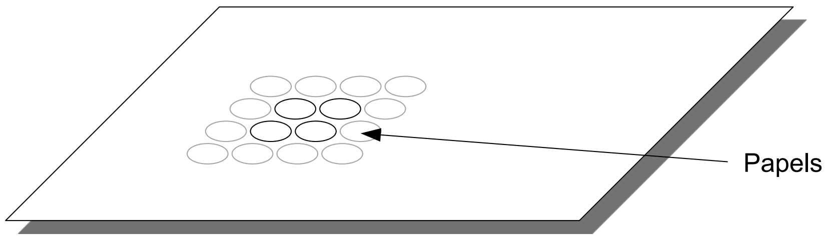
The answer to an architecture for an electronic paper device does not lie in this direction. Our thesis work has focused on bio-inspired cellular architectures, and the BioTissue is an autonomous self-repairable system able to compute powerful applications like those running on a computer. The BioTissue is thus the global answer for a new electronic paper architecture.

9.3.1 Structural definition

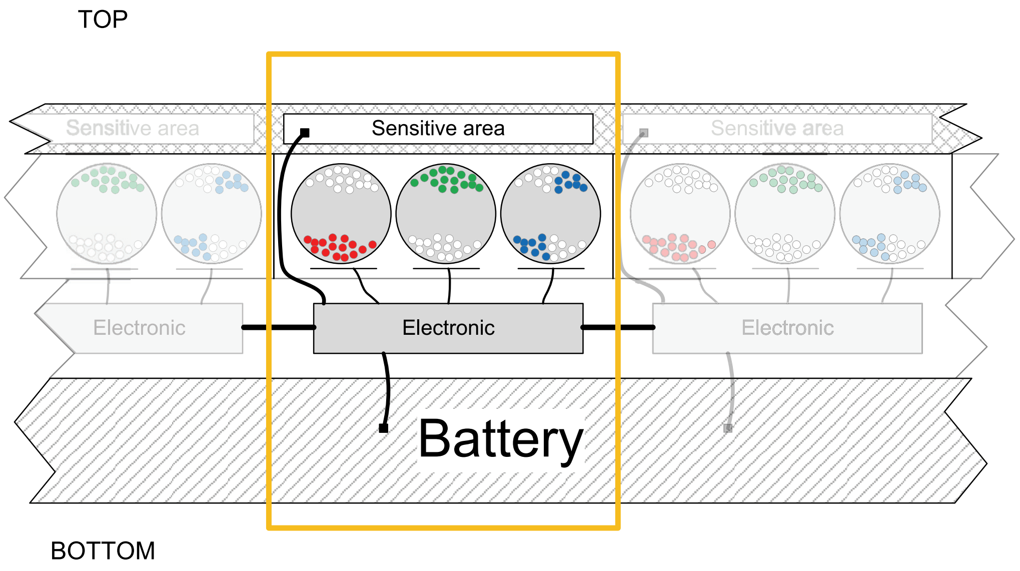
Based on the promising research done on printed displays, we chose to use the architecture of the BioTissue, with a reconfigurable element for each pixel. Thus, we defined first the following minimal structure, called a *papel* (for paper pixel) with, from top to bottom:

- A tactile sensor able to detect a human finger, or stylus.
- A pixel, which could be black and white, or full color. The technique could be based on e-ink display or similar.
- An electronic reconfigurable element. The number of logic cells will be limited by the size of the manufacturing process with the given pixel size.

- A connection to a thin film battery.



(a) Papels inside an electronic paper.



(b) Cross section view of a papel.

Figure 9.7: *Structure of a* papel.

Figure 9.7 presents the structure of a *papel*. Like in the BioTissue, each *papel* is connected to its nearest neighbors using point-to-point wires. The number of *papels* is defined by the size of the electronic paper, and its pixel number.

The manufacture of such electronic paper is simple: once the design of a *papel* is completed, this element should be replicated to fill in the desired electronic paper surface.

Variation based on nanoelectronic components

Such an electronic paper device built with a printed transistor may have a weak performance, giving to this product a limited market range. The electronic paper built upon a matrix of *papel* may be interesting if it could challenge laptop performances.

Rather than using a simple printed reconfigurable element in each *papel*, it may be better to have nanoelectronic components, whose performance could result in extremely fast microprocessors on a really small surface. In Chapter 5, we studied the

BioCube and stated that this system could be a perfect simulator for electronic reprogrammable circuits built in 3D with nanocomponents, like nanotubes. As soon as these technologies are industrially viable, it could be interesting to substitute the printed transistor of our electronic paper, with a nanochip. This kind of chip could be built in 3D to the size of a cube whose sides each measure a few micrometers. Such a small component, while having thousand transistors, could be inserted into the paper using DNA placement methods [69]. The configuration of each *papel* could use the results of the *3D Tom Thumb* application described in section 6.2.

9.3.2 Advantages

Since the architecture is based on a cellular structure, with fault-tolerance capabilities, a *papel* failure will not damage the whole system: consequences for the user will be one or several dead pixels, which is not critical.

During the manufacturing process, the device size could be easily parameterized. Before running the printing process, the desired electronic paper size can be changed by the operator, and for example some additional *papel* may be printed.

Since the system is based on the BioTissue architecture, the software running on the electronic paper will detect the device size and automatically adapt its functionality. Thus, if the manufactured device is an electronic newsprint, the contents of the newspaper will be automatically reorganized, as a web-browser does when its window size is modified. No engineering, nor human intervention is required when changing the machine size and its number of *papels*.

9.3.3 Disadvantages

There is unfortunately a major disadvantage compared to the classical architecture based on a microprocessor, memory, a keyboard and a display controller. In this classical architecture, all processing is computed sequentially, and methodologies, as programming tools, exist. In our new architecture, the system will operate a task in parallel on each *papel*. Thus, while developing an application, the whole methodology needs to be reconsidered. Development will take longer than for a microprocessor approach. A solution for this issue could be to consider the whole electronic paper sheet like a huge Field Programmable Gate Array (FPGA), and develop a microprocessor softcore that could be dynamically placed. However, performance should be estimated before going in this direction, and a solution needs to be found about how to distribute the softcore microprocessor over a potentially imperfect tissue, where defective *papels* could be present.

9.4 Conclusion

This chapter, which brings our thesis to a close, described how the bio-inspired approach we used as a common approach of three cellular machines, may constitute a solution for an electronic paper architecture. Given the low cost electronic printing techniques, which do not allow high component density, it makes senses to use a cellular structure for an electronic paper.

However, as we have said our architecture can not currently be implemented. Once the technologies needed for the integration of our structure are mature, the manufacture of a prototype should be considered.

It is certain that electronic paper will be a reality in the future. However, in year 2054, when reality has caught up with fiction, will our character sitting in the subway coach be reading this thesis on electronic paper built with the architecture and manufacturing process we just described?

Chapter 10

Conclusions

This thesis has presented our research in the field of cellular computing, involving the realization of three machines, each one composed of a cellular array. Each cell of our three inventions is based on the following common structure: a reconfigurable computing component, a display element and an external sensor allowing user interactions with each cell. The implementation in each system of two similar applications enabled us to compare and evaluate the performance of our three realizations. In addition to these developments, we have proposed an architecture for a novel computer: electronic paper.

We will firstly present some general conclusions for the whole project, and then give possibilities for future research based upon the current state of this thesis.

10.1 General conclusions

The evolution of computers over the last fifty years has been incredible. The first models were huge, slow, power hungry, extremely expensive and needed many operators and technicians for daily maintenance. They were very rare and could only be found in some major companies or universities, which used them for computing mathematical operations. The invention of the electronic transistor and the development of the integrated circuit had lowered the cost while increasing the computation power of the microprocessors. Twenty five years ago, the idea to put a computer into each home was launched by IBM®. This idea was seen as a marginal business opportunity at the time but turned out to be the greatest and most visionary idea for the future of the computer. Today, practically each person owns a PC at home and many devices are built around a microprocessor, like watches, credit cards and washing machines... Nowadays, microprocessors are extremely powerful, inexpensive and used in practically every electronic device. Future trends indicate further increase in the speed and performance of processors used in computers, and the use of processors in new targets such as clothing, or in the electronic newspaper which may replace the classic printed newspaper.

These evolutions will be possible through the reduction of the transistor size to nanoscale size, the development of flexible components and also the exploration of new computer architectures mainly based on multi-cellular systems.

Our research has mainly been based on the study of some cellular systems. We first concentrated on their hardware architecture, and progressed to the implementation of some applications on these machines.

10.1.1 Three cellular machines

The first realization, BioWall, is a novel hardware platform designed for bio-inspired cellular computing. This cellular machine is built upon a mesh topology of $4'000$ identical molecules each constituted of an FPGA coupled to an 8×8 bi-color LED display and a touchsensor. The massive parallelism of the BioWall along with the scalability of this platform provide possibilities for very powerful computation applications. However, the quite complex architecture of this machine needs to be improved to allow the system to behave autonomously. These improvements were integrated into the next development.

Several major improvements to the BioWall were added to develop the BioTissue, which is a similar machine still based on a cellular architecture. The core of the new cell is composed of a much more powerful FPGA. It is coupled to new components like FLASH and SRAM memories. Its three color RGB LED display give it a range of millions of colors. All these improvements make the BioTissue a cellular machine, whose specification allow it to run various complex applications and make it autonomous. Moreover, the Globally Asynchronous, Locally Synchronous (GALS) structure used for the architecture of the BioTissue brought it closer to the considerations of the current layout techniques used in integrated circuit conception, where multiple clock domains exist in the same circuit [81, 23, 121].

The BioCube is the expansion of our two 2D machines to the third dimension, which may be the future topology of microprocessors built on a nanometer scale. This machine is a novel hardware platform, created in the aim of realizing 3D bio-inspired cellular computing. The versatility and scalability of this platform allow it to simulate cell networks, where each cell is connected to its six neighbors.

10.1.2 Two common applications

We implemented two different applications on each of our three machines. The first application, the Game of Life cellular automaton, was chosen especially for the notable characteristics already mentioned, and for its simple implementation on a cellular tissue. It is interesting to note the difference between these three machines. Whereas the implementation of the application on the BioWall machine was extremely simple, since the structure of this automaton was close to the architecture of our machine, the implementation was more complex for the two other systems. The asynchronous behavior of the BioCube and the BioTissue made it difficult to reuse the developments already made to the BioWall. Even if solutions were proposed to allow synchronous applications like the Game of Life automaton to run on the machine, an asynchronous implementation of this automaton would have been more elegant, and more suited to the GALS hardware architectures of the BioCube and the BioTissue. Globally synchronous automata may soon be a thing of the past, current and upcoming research on cellular automata should be focused on asynchronous connections between cells: this approach will be closer to the reality of electronic hardware developments.

The second application, based on the Tom Thumb algorithm, is intended to propose a methodology for programming a cellular hardware structure. Unlike a current cellular programming methodology used in FPGA for example, where the dimensions of the array need to be known in advance, our algorithm can adapt itself over a cellular network whose dimensions are unknown or may vary. In the first implementation of our algorithm, onto the BioWall, we discussed how programming and replicating a configuration stream can be done on a basic cellular machine. The extension of the algorithm from 2D to 3D was the next step. This extended algorithm was tested on the BioCube. In the evolution of manufacturing processes the main goal is to decrease the lithographic size of transistors, to increase their number and their speed. This has the disadvantage of increasing the number of faults in a circuit. The size of a transistor gets smaller and smaller, to the point where the number of atoms constituting the gate of a MOS transistor could be counted on the fingers of two hands. The direct consequence is the increase in defective transistors at the fabrication process. To stay ahead of the current industrial layouts and manufacturing methodologies, we added to our algorithm self-repair concepts that could succeed in programming a cellular array, even if some cells are defective. The BioWall did not allow this new version of the Tom Thumb algorithm to be tested, but the BioTissue, whose architecture is more powerful was able to support the improved algorithm.

10.1.3 Electronic paper: where all the previous research converge

Current computer trends mainly focus on improving the speed of microprocessors by using new architectures, and decreasing the manufacturing process in the aim of having the smallest transistor gate as possible[1]. The other tendency is a decrease in the size and weight of the individual Personal Computer (PC). Currently, powerful computers could be found as laptops, pocket PCs, and are recently included in mobile phones. This trend is set to continue as powerful computers become more pervasive and work their way into new areas such as clothing and the written press.

The convergence of our three realizations allowed us to define and propose in chapter 9 an architecture, based on a cellular topology, for electronic paper. The Tom Thumb application as defined for the BioTissue in chapter 8 could be used for programming such a tissue, with the major advantage of succeeding even if some cells are defective.

10.2 Future work

Based on our current results, this section will present some ideas for further research.

10.2.1 BioCube nanoscale implementation

Nanotechnologies constitute a brand new research domain, which is very hot and full of possibilities. Although no logic component is ready yet for commercial industrialization, a large quantity of research and some prototypes are currently on their way.

[1]The advantages of a small transistor gate are the increase of the switching frequency of the transistor, the reduction in energy consumed and the increase in the number of transistors on an identical silicon surface.

This research is just beginning and it will be interesting to follow the trends and evolutions in this domain. It's a safe bet that the current evolution of 3D component stacking will be one of the keys of microprocessor architecture on a nanometer scale. The main challenge will be to check if the structure currently proposed by the BioCube could be used for manufacturing microprocessors constituted of identical cells stacked into three dimensions.

10.2.2 Evaluation of cell resources

The current BioTissue cell is endowed with a very powerful Field Programmable Gate Array (FPGA), and a lot of memory. In the electronic paper architecture, no size of reconfigurable array, nor memory size has been proposed. This question is quite difficult to answer at the present stage in our research. Based on an architecture where a cell drives only one pixel, many thousand of cells will be needed for a small "page" of electronic paper.

It may be interesting to conduct research to define the resources needed for each cell to obtain a desired goal. For certain applications such as an electronic paper page able to display the contents of a book, cells may not need a large amount of computing resources and still have enough memory to store the contents of several pages. A newspaper that could display texts, images, and also video on the other hand would need more computing resources.

A study based on several tasks (e-book reader, newspaper application, video decompression, diary, calculator...) and many different topologies (one pixel for each cell, or a matrix of several pixels for one cell) should be done. The results could allow some typical architectures to be defined along with their pros and cons. This could be similar to the current panel selection of microprocessors that could be bought from a distributor. Each one would have its advantages and disadvantages in performance, energy consumption, price, dedicated applications, multitask operations...

10.2.3 Operating system upgrade

We have developed an operating system for the BioTissue. It is really simple and allows several applications to be run over the same medium. At the moment a misuse of μOS could result in unexpected behaviors. These could be avoided by adding several security checks, thus giving a strong μOS application without any corrupted results.

Many other improvements would be welcome, such as allowing applications to be moved over the tissue, or resizing the window of an application.

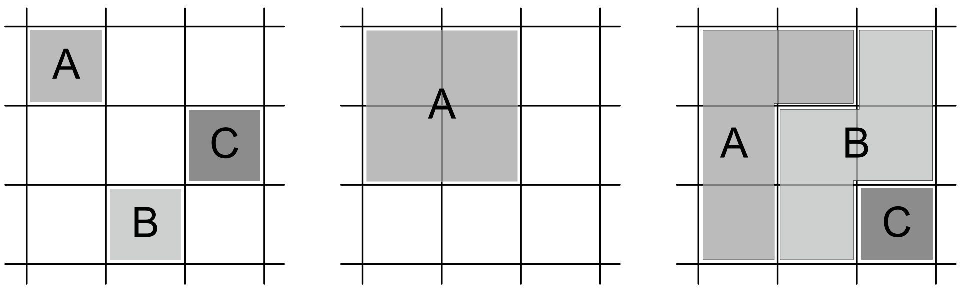
10.2.4 Self-organized microprocessor

The BioTissue is endowed with a huge processing capacity that is currently not completely exploited. An interesting and promising research subject is based on resource distribution of a task over such a cellular network. In the case of an application that is not based on a Cellular Automata (CA) approach, and rather than having only one cell executing a dedicated task (Fig. 10.1(a)), it may be interesting to look for an algorithm able to distribute this task over the cellular network of the BioTissue. This could be approached in several ways:

1. Distribution of a task over a predefined number of cells (Fig. 10.1(b)). The necessary number of cells needed to execute the specified task is known and has been precalculated by the developer. The surface where the task is executed is known and can not be adjusted.

2. Dynamic distribution of a task over an unused surface of cells (Fig. 10.1(c)). The application will calculate the number of cells it needs depending on the free resources of the network. The task will be executed by several cells whose location is dynamically calculated depending on the free cells available. When some of these cells are no longer being used, they can be released. (This is similar to the *malloc* and *free* functions in C++, where a block of memory can be allocated and accessed by a pointer, and then released when this memory block is no longer needed.)

We have summarized the advantages and disadvantages of these three methods in table 10.1.



(a) Algorithm 1: each task is executed by one cell.

(b) Algorithm 2: a task can be distributed over 2×2 cells. Impossibility in this example to run more than one task at the same time, since no 2×2 cell's surface is still free.

(c) Algorithm 3: a task can be distributed over a maximum of 4 adjacent cells. Disposition is dynamic and flexible. The task C does not have the place for running over 4 cells, and can still run in one cell.

Figure 10.1: *Three different methods for distributing tasks over a cellular network. Examples based on a small BioTissue of* 3×3 *cells.*

	Algorithm 1	Algorithm 2	Algorithm 3
Complexity	+	0	-
Execution time	-	0	+
Number of tasks	+	-	+
Surface lost	+	-	+

Table 10.1: *Comparison of task distribution algorithms. ("-" means bad results, "0" neutral and "+" good results)*

Currently algorithms 1 and 2 are used in the BioTissue, but also in many research fields like that of dynamic resources allocations in FPGAs [100]. An interesting research possibility could be the study of algorithm 3. Rather than speaking about tasks, it may be interesting to study how a microprocessor could be distributed over a cellular tissue like the BioTissue and how it could dynamically allocate new cells to compute some specific functions. The ultimate feature would be for a microprocessor to be able to replicate itself when needed in order to act like a multi-core microprocessor. All of the surface's utilization would be calculated online, and the best distribution chosen based on this distribution algorithm (Fig. 10.1(c)). In the case of microprocessor replication, our results from the *eSOS* application (cf. section 8.3) could be used, since they allow a complex structure to be replicated over a cellular network.

10.2.5 Detailed structure for the electronic paper

In this thesis work, we proposed a hardware architecture for electronic paper. This is based on a high level overview of the general structure. The cell functionalities and the main details are presented as are the interconnections needed to link a cell with its surrounding neighbors. In the near future, the plastic transistor technology will become mature and one possible manufacturing method to be expected is to place printed electronic paper on a plastic sheet.

A future research direction or thesis could be to propose a layout for an electronic paper machine based on our architecture. This layout will try to define the complete mapping of all the transistors for each cell. A display technology could be chosen and the stacking of the electronics with the display, the tactile sensor and the battery could be studied. Once the complete layout of a cell is accomplished, its interconnection with its neighbors should be detailed, without forgetting the edge where specific components could be located (WIFI interface, battery charger, audio outputs, memories card slots...)

Simulations and eventually prototyping would be carried out to give results about performance, energy consumption, and failure tolerance for different sizes of a complete layout of such a system.

In this thesis, we explored several cellular architectures. The different aspects of our research converge towards a proposal for a hardware architecture of a novel universal electronic paper machine. It is hoped that one day we will read our news or even a PhD thesis on an electronic sheet of paper built upon our research...

Bibliography

[1] 3D Game of Life simulators: `http://www.ibiblio.org/e-notes/Life/Game.htm`.

[2] H. Abelson, D. Allen, D. Coore, C. Hanson, G. Homsy, T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[3] A. Adamatzky, L. Bull, B. De Lacy Costello, S. Stepney, and C. Teuscher, editors. *Unconventional Computing.* Luniver Press, Beckington, UK, 2007.

[4] Agilent Technologies. *Introduction to driving LED matrices, application note 1216*, May 2001.

[5] B. M. Al-Hashimi. *System-on-Chip: Next Generation Electronics.* IEE Press, 2006.

[6] P. Alfke. Evolution and revolution: recent progress in field-programmable logic. Technical report, Xilinx, San Jose, 2001.

[7] L. Alvin and Jr. Pachynski. US Patent 3,995,120: Digital time-division multiplexing system, 1976.

[8] M. Amde, T. Felicijan, A. Efthymiou, D. Edwards, and L. Lavagno. Asynchronous On-Chip Networks. *IEE Proceedings Computers and Digital Techniques*, 152(02), March 2005.

[9] M. Amos. *Cellular computing.* Oxford University Press, New York, 2004.

[10] M. Barr. Pulse width modulation. *Embedded Systems Programming*, pages 103–104, September 2001.

[11] C. Bays. 3D Life (?). *Complex Systems*, 6(5):433–442, 1992.

[12] P. Beckett and A. Jennings. Towards nanocomputer architecture. In *CRPIT '02: Proceedings of the seventh Asia-Pacific conference on Computer systems architecture*, pages 141–150, Darlinghurst, Australia, Australia, 2002. Australian Computer Society, Inc.

[13] P. Bendito. Digital color design with the RGB color cube: Visualization and color coordination activities. *Journal of Design Communication, Northern Illinois University*, 2, Spring 2000.

[14] E. R. Berlekamp, J. H. Conway, and R. K. Guy. *Winning Ways for your Mathematical Plays*, volume 2, chapter 25, pages 817–850. Academic Press, New York, 1982.

[15] C. Bernard, D. Mange, and A. Stauffer. *Catalogue logidules*. EPFL, Laboratoire de systèmes logiques, Lausanne, 1994.

[16] T. Bjerregaard and S. Mahadevan. A survey of research and practices of Network-on-chip. *ACM Comput. Surv.*, 38(1):1, 2006.

[17] R. Boyce and M. Hoff. A history of microprocessor design at intel. *IEEE Micro*, 1:8–22, Feb. 1981.

[18] C. J. Brabec, N. S. Sariciftci, and J. C. Hummelen. Plastic solar cells. *Advanced Functional Materials*, 11(1):15–26, Feb. 2001.

[19] R. S. Braich, N. Chelyapov, C. Johnson, P. W. K. Rothemund, and L. Adleman. Solution of a 20-variable 3-SAT problem on a DNA computer. *Science*, 296(5567):499–502, Apr. 2002.

[20] C. Bruetsch. *BioWord*. EPFL, Laboratoire de systèmes logiques, February 2005.

[21] J. Byl. Self-reproduction in small cellular automata. *Physica D*, 34:295–299, 1989.

[22] R. Canham and A. M. Tyrrell. An embryonic array with improved efficiency and fault tolerance. In editor J. Lohn et al., editor, *Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH'03)*, pages 265–272. IEEE Computer Society, Los Alamitos, CA, 2003.

[23] A. Chattopadhyay and Z. Zilic. GALDS: a complete framework for designing multiclock ASICs and SoCs. 13(6):641–654, June 2005.

[24] C. Q. Choi. Qubit twist. *Scientific American*, 292(4):28, 2005.

[25] H.-H. Chou and J. A. Reggia. Problem solving during artificial selection of self-replicating loops. *Physica D*, 115(3-4):293–312, 1998.

[26] C. P. Collier, E. W. Wong, M. Belohradsky, F. M. Raymo, J. F. Stoddart, P. J. Kuekes, R. S. Williams, and J. R. Heath. Electronically configurable molecular-based logic gates. *Science*, 285(5426):391–394, Jul. 1999.

[27] P. G. Collins and P. Avouris. Nanotubes for electronics. *Scientific American*, 283(6):62–69, Dec. 2000.

[28] M. Coltheart. The persistences of vision. *Royal Society of London Philosophical Transactions Series B*, 290:57–69, July 1980.

[29] C. Constantinescu. Trends and challenges in vlsi circuit reliability. *IEEE Micro*, 23(4):14–19, 2003.

[30] Intel Corporation. Excerpts from a conversation with Gordon Moore: Moore's law. `ftp://download.intel.com/museum/Moores_Law/Video-Transcripts/Excepts_A_Conversation_with_Gordon_Moore.pdf`, 2005.

[31] Intel Corporation. Meet the world's first 45nm processor: Intel® 45nm transistor technology. `http://www.intel.com/technology/silicon/45nm_technology.htm?iid=homepage+42nm`, 2006.

[32] The Cubatron: `http://www.nw.com/nw/projects/cubatron/`.

[33] W. J. Dally and B. Towles. Route packets, net wires: on-chip inteconnectoin networks. In *DAC '01: Proceedings of the 38th conference on Design automation*, pages 684–689, New York, NY, USA, 2001. ACM Press.

[34] G. de Micheli and L. Benini. Networks on chip: A new paradigm for systems on chip design. In *DATE '02: Proceedings of the conference on Design, automation and test in Europe*, page 418, Washington, DC, USA, 2002. IEEE Computer Society.

[35] D. de S. Price. Gears from the greeks: The antikythera mechanism - a calendar computer from ca. 80 BC. *Transaction of the American Philosophical Society*, 64(7):1–70, Nov. 1974.

[36] A. Dehon and M. J. Wilson. Nanowire-based sublithographic programmable logic arrays. In *FPGA '04: Proceedings of the 2004 ACM/SIGDA 12th international symposium on Field programmable gate arrays*, pages 123–132, New York, NY, USA, 2004. ACM Press.

[37] A. K. Dewdney. The game of life aquires some successors in three dimensions. *Scientific American*, pages 8–13, Feb. 1987.

[38] S. Dorthi and R.L. Haggard. A survey of dynamically reconfigurable FPGA devices. In *Proceedings of the 35th Southeastern Symposium on System Theory*, pages 422–426, 2003.

[39] E. Downing, L. Hesselink, J. Ralston, and R. Macfarlane. A Three-Color, Solid-State, Three-Dimensional Display. *Science*, 273:1185–1189, August 1996.

[40] K. E. Drexler. *Nanosystems: Molecular Machinery, Manufacturing, and Computation*. John Wiley, New York, 1992.

[41] M. Dubash. Moore's Law is dead, says Gordon Moore. *Techworld*, Apr. 2005. `http://www.techworld.com/opsys/news/index.cfm?NewsID=3477`.

[42] Xilinx® EasyPath®: `http://www.xilinx.com/products/silicon_solutions/fpgas/easypath/index.htm`.

[43] W. R. Fahrner. *Nanotechnology and Nanoelectronics: Materials, Devices, Measurement Techniques.* Springer-Verlag, Berlin Heidelberg New-York, 2005.

[44] R. P. Feynman. There is plenty of room at the bottom. In *Engineering and Science*. California Institute of Technology, December 29th 1959. `http://www.zyvex.com/nanotech/feynman.html`.

[45] D. H. Freeman G. A. Freeman. US Patent 5,931,764: Wearable device with flexible display, 1999.

[46] Patterns for Conway's Game of Life: `http://www.radicaleye.com/lifepage/`.

[47] Martin Gardner. Mathematical games. the fantastic combinations of John Conway's new solitaire game "Life". *Scientific American*, 223:120–123, 1970.

[48] B. Gojman, R. Rubin, C. Pilotto, A. DeHon, and T. Tanamoto. 3D nanowire-based programmable logic. In *Nano-Networks and Workshops, 2006. NanoNet '06. 1st International Conference on*, pages 1–5, Sept. 2006.

[49] K. Govil, D. Teodosiu, Y. Huang, and M. Rosenblum. Cellular Disco: resource management using virtual clusters on shared-memory multiprocessors. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 154–169, New York, NY, USA, 1999. ACM Press.

[50] D. A. Grier. *When Computers Were Human.* Princeton University Press, May 2005.

[51] L. Hansen and T. Thomas. Complete FPGA and CPLD power analysis. *Xcell Journal*, 53:80–83, Second Quarter 2005.

[52] A. Harter. *Three-dimensional integrated circuit layout.* Cambridge University Press, New York, NY, USA, 1992.

[53] J. P. Hayes. *Computer Architecture and Organization.* McGraw-Hill Higher Education, 2002.

[54] J. Held, J. Bautista, and S. Koehl. *From a Few Cores to Many: A Tera-scale Computing Research Overview.* Intel Corporation, white paper, research at intel edition, 2006.

[55] T. Higuchi, Y. Liu, and X. Yao, editors. *Evolvable Hardware.* Genetic and Evolutionary Computation. Springer US, 2006.

[56] K. Imai, T. Hori, and K. Morita. Self-reproduction in three-dimensional reversible cellular space. *Artificial Life*, 8(2):155–174, 2002.

[57] Microchip Technoloy Inc. *PIC10F200/202/204/206 Data Sheet. 6-Pin, 8-bit FLASH Microcontrollers*, ds41239d edition, April 2007.

[58] Irvine sensors: `http://www.irvine-sensors.com/chip_stack.html`.

[59] ITRS. International technology roadmap for semiconductors. Technical report, 2005. `http://www.itrs.net/Links/2005ITRS/Home2005.htm`.

[60] D. Jilli. *BioCpu*. EPFL, Laboratoire de systèmes logiques, February 2003.

[61] J. A. Kahle, M. N. Day, H. P. Hofstee, C. R. Johns, T. R. Maeurer, and D. Shippy. Introduction to the cell multiprocessor. *IBM J. RES. & DEV*, 49(4/5):589–604, July/September 2005.

[62] T. Kamgaing, K. Ichikawa, X. Y. Zeng, K.-P. Hwang, Y. Min, and J. Kubota. Future package technologies for wireless communication systems. *Intel® Technology Journal*, 9(4):353–364, Nov. 2005.

[63] N. B. Karayiannis and A. N. Venetsanopoulos. *Artificial Neural Networks: Learning Algorithms, Performance Evaluation, and Applications*. Kluwer Academic Publishers, Norwell, MA, USA, 1992.

[64] S. Kawamura, N. Sasaki, T. Iwai, M. Nakano, and M. Takagi. Three-dimensional CMOS IC's fabricated by using beam recrystallization. 4(10):366–368, Oct. 1983.

[65] J. S. Kilby. US Patent 3,138,743: Miniaturized electronic circuits, 1964.

[66] Kingbright. *KAAF-5060PBESEEVGC RGB LED*, November 2003. Available from `http://www.kingbright-europe.de`.

[67] W. Knight. Most flexible electronic paper yet revealed. *NewScientist.com news service*, Jan. 2004.

[68] R. Kurzweil. *The Singularity Is Near: When Humans Transcend Biology*. Viking, 2005.

[69] J. W. Tseng L. A. Bodony, R. Bryan. US Patent 6,307,751: Flexible circuit assembly, 2001.

[70] P. K. Lala, editor. *Self-checking and fault-tolerant digital design*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 2001.

[71] C. G. Langton. Self-reproduction in cellular automata. *Physica D*, 10:135–144, 1984.

[72] J. Lee, S. Adachi, F. Peper, and K. Morita. Asynchronous Game of Life. *Physica D Nonlinear Phenomena*, 194:369–384, July 2004.

[73] H. A. Lichtfuss. US Patent 6,680,724: Flexible electronic viewing device, 2004.

[74] N. J. Macias and L. J. K. Durbeck. Self-assembling circuits with autonomous fault handling. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DOD Workshop Conference on Evolvable Hardware*, pages 46–55, Los Alamitos, CA, 2002. IEEE Computer Society Press.

[75] D. Mange, M. Sipper, and P. Marchal. Embryonic electronics. *BioSystems*, 51(3):145–152, 1999.

[76] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The Embryonics approach. *Proceedings of the IEEE*, 88(4):516–541, April 2000.

[77] D. Mange, A. Stauffer, G. Tempesti, F. Vannel, and A. Badertscher. *Evolvable Hardware*, chapter 5, Bio-Inspired Computing Machines with Artificial Division and Differentiation, pages 85–98. Genetic and Evolutionary Computation. Springer US, 2006.

[78] D. Mange and M. Tomassini, editors. *Bio-inspired Computing Machines: Toward Novel Computational Architectures*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[79] S. Mann. Smart clothing: The wearable computer and wearcam. *Personal and Ubiquitous Computing*, 1(1):21–27, Mar. 1997.

[80] Sharp Mebius PC-RD3D will be the world's first 3D laptop: `http://www.mobilemag.com/content/100/334/C2020/`.

[81] J. Monteiro, V. Tiwari, and P. Rakesh. *EDA for IC Implementation, Circuit Design And Process Technology*. CRC Press, March 2006.

[82] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8), April 1965.

[83] F. Moraes, N. Calazans, A. Mello, L. Möller, and L. Ost. HERMES: an infrastructure for low area overhead packet-switching networks on chip. *Integrated VLSI Journal*, 38(1):69–93, 2004.

[84] J. M. Moreno, Y. Thoma, and E. Sanchez. POEtic: A prototyping platform for bio-inspired hardware. In J.M. Moreno, J. Madrenas, and J. Cosp, editors, *Evolvable Systems: From Biology to Hardware (ICES 2005)*, volume 3637 of *LNCS*, pages 177–187, Berlin Heidelberg, 2005. Springer-Verlag.

[85] S. K. Nayar and V. N. Anand. 3D display using passive optical scatterers. *Computer*, 40(7):54–63, 2007.

[86] C. L. Nehaniv. Self-reproduction in asynchronous cellular automaton. In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, editors, *Proceedings of the 2002 NASA/DOD Workshop Conference on Evolvable Hardware*, pages 201–209, Los Alamitos, CA, 2002. IEEE Computer Society Press.

[87] A. Ngouanga, G. Sassatelli, L. Torres, T. Gil, A. Soares, and A. Susin. A contextual resources use: a proof of concept through the APACHES' platform. In *Proceedings of the 2006 IEEE Workshop on Design and Diagnostics of Electronic Circuits and Systems (DDECS)*, pages 44–49, April 2006.

[88] Nova: `http://www.nova.ethz.ch/`.

[89] R. N. Noyce. US Patent 2,981,877: Semiconductor device and lead structure, 1961.

[90] A. Du Pasquier, H. E. Unalan, A Kanwal, S. Miller, and M. Chhowalla. Conducting and transparent single-wall carbon nanotube electrodes for polymer-fullerene solar cells. *Applied Physics Letters*, 87, 2005.

[91] D. A. Patterson and J. L. Hennessy. *Computer Organization and Design: The Hardware/Software Interface, Third Edition.* Morgan Kaufmann, August 2004.

[92] L. D. Paulson. New electronic paper can display moving images. *IEEE Computer*, 36(12):26, Dec. 2003.

[93] Perceptive pixel: `http://www.perceptivepixel.com`.

[94] J.-Y. Perrier, M. Sipper, and J. Zahnd. Toward a viable, self-reproducing universal computer. *Physica D*, 97:335–352, 1996.

[95] C. Peterson. Taking technology to the molecular level. *Computer*, 33(1):46–53, 2000.

[96] Pfeiffer report: `http://www.pfeifferreport.com/trends/ett_eink.html`.

[97] D.C Pham, T. Aipperspach, and D. Boerstler et al. Overview of the architecture, circuit design, and physical implementation of a first-generation CELL processor. *IEEE Solid-State Circuits*, 41(1):179–196, 2006.

[98] D.C Pham, E. Behnen, M. Bolliger, H.P. Hostee, C. Johns, J. Kalhe, A. Kameyama, and J. Keaty. The design methodology and implementation of a first-generation CELL processor: a multi-core SoC. In *Proceedings of the Custom Integrated Circuits Conference*, pages 45–49. IEEE Computer Society, September 2005.

[99] Polymer Vision$^{\circledR}$: `http://www.polymervision.com`.

[100] A. E. Upegui Posada. *Dynamically reconfigurable bio-inspired hardware.* PhD thesis, EPFL, 1015 Lausanne, 2006. Thesis 3632.

[101] W. Poundstone. *Recursive Universe: Cosmic Consequences and the Limits of Scientific Knowledge.* NTC Publishing Group, 1985.

[102] C. Poynton. *A technical introduction to digital video.* John Wiley and Sons, 1996.

[103] D. Ratner and M. Ratner. *Nanotechnology: a gentle introduction to the next big idea.* P T R Prentice-Hall, Englewood Cliffs, NJ 07632, USA, 2003.

[104] J. A. Reggia, S. L. Armentrout, H.-H. Chou, and Y. Peng. Simple systems that exhibit self-directed replication. *Science*, 259:1282–1287, February 1993.

[105] P. Rendell. Turing universality of the Game of Life. pages 513–539, 2002.

[106] H. F. Restrepo and D. Mange. An embryonics implementation of a self-replicating universal Turing machine. In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, editors, *Evolvable Systems: From Biology to Hardware (ICES 2001)*, volume 2210 of *Lecture Notes in Computer Science*, pages 74–87. Springer-Verlag, Berlin, 2001.

[107] M. C. Roco and W. S. Bainbridge, editors. *Converging technologies for improving human performance. Nanotechnology, biotechnology, information technology and cognitive science.* NSF/DOC - sponsored report, Arlington, VA, 2002.

[108] D. Roggen, D. Floreano, and C. Mattiussi. A morphogenetic evolutionary system: Phylogenesis of the poetic circuit. In *ICES*, pages 153–164, 2003.

[109] R. Rojas and U. Hashagen, editors. *The first computers: history and architectures.* MIT Press, Cambridge, MA, USA, 2000.

[110] Samsung electronics develops 16-chip multi-stack package technology: `http://www.samsung.com/tr/PressCenter/PressKit/presskit_20061109_0000299907.asp`.

[111] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. Phylogeny, ontogeny, and epigenesis: Three sources of biological inspiration for softening hardware. In T. Higuchi, M. Iwata, and W. Liu, editors, *Proceedings of The First International Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, volume 1259 of *Lecture Notes in Computer Science*, pages 35–54. Springer-Verlag, Heidelberg, 1997.

[112] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations.* SpringerVerlag, 2004.

[113] Quantum research group: `http://www.qprox.com/`.

[114] SiPix®: `http://www.sipix.com`.

[115] M. Sipper. The emergence of cellular computing. *Computer*, 32(7):18–26, July 1999.

[116] M. Sipper. Kinematic self-replicating machines by Robert A. Freitas, Jr., and Ralph C. Merkle. *Artif. Life*, 12(1):187–188, 2006.

[117] M. Sipper, D. Mange, and A. Stauffer. Ontogenetic hardware. *BioSystems*, 1997. (to appear).

[118] M. Sipper and E. Sanchez. Configurable chips meld software and hardware. *Computer*, 33(1):120–121, January 2000.

[119] M. Sipper, E. Sanchez, 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*, 1(1):83–97, April 1997.

[120] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Pérez-Uribe, and A. Stauffer. The POE model of bio-inspired hardware systems: A short introduction. In J. R. Koza, K. Deb, M. Dorigo, D. B. Fogel, M. Garzon, H. Iba, and R. L. Riolo, editors, *Genetic Programming 1997: Proceedings of the Second Annual Conference*, pages 510–511. Morgan Kaufmann, San Francisco, CA, 1997.

[121] S. Sirowy, Wu Yonghui, S. Lonardi, and F. Vahid. Clock-frequency assignment for multiple clock domain systems-on-a-chip. *Design, Automation and Test in Europe Conference and Exhibition*, pages 1–6, April 16-20 2007.

[122] H. Sirringhaus, T. Kawase, R. H. Friend, T. Shimoda, M. Inbasekaran, W. Wu, and E. P. Woo. High-resolution inkjet printing of all-polymer transistor circuits. *Science*, 290(5499):2123–2126, Dec. 2000.

[123] D. D. Spencer. *The timetable of computers*. Camelot Publishing Co., Ormond Beach, FL, USA, 1997.

[124] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. Biowatch: A giant electronic bio-inspired watch. In D. Keymeulen, A. Stoica, J. Lohn, and R. S. Zebulum, editors, *Proceedings of the Third NASA/DOD Workshop on Evolvable Hardware*, pages 185–192, Los Alamitos, CA, 2001. IEEE Computer Society.

[125] A. Stauffer, D. Mange, G. Tempesti, and C. Teuscher. A self-repairing and self-healing electronic watch: The biowatch. In *ICES '01: Proceedings of the 4th International Conference on Evolvable Systems: From Biology to Hardware*, pages 112–127, London, UK, 2001. Springer-Verlag.

[126] A. Stauffer and M. Sipper. Biomorphs implemented as a data and signals cellular automaton. In W. Banzhaf, T. Christaller, P. Dittrich, J. T. Kin, and J. Ziegler, editors, *Proceedings of the 7th European Conference on Artificial Life (ECAL 2003)*, volume 2801 of *Advances in Artificial Life*, pages 235–241, Berlin Heidelberg, 2003. Springer-Verlag.

[127] A. Stauffer and M. Sipper. Data and signals: A new kind of cellular automaton for growing systems. In J. Lohn, R. Zebulum, J. Steincamp, D. Keymeulen, A. Stoica, and M. I. Ferguson, editors, *Proceedings of the 2003 NASA/DOD Conference on Evolvable Hardware*, pages 235–241, Los Alamitos, CA, 2003. IEEE Computer Society.

[128] G. Tempesti. A new self-reproducing cellular automaton capable of construction and computation. In F. Morán, A. Moreno, J. J. Merelo, and P. Chacón, editors, *ECAL'95: Third European Conference on Artificial Life*, volume 929 of *Lecture Notes in Computer Science*, pages 555–563, Heidelberg, 1995. Springer-Verlag.

[129] G. Tempesti, D. Mange, E. Petraglio, A. Stauffer, and Y. Thoma. Developmental Processes in Silicon: An Engineering Perspective. In *J. Lohn et al., Eds., Proceedings, 2003 NASA/DoD Conference on Evolvable Hardware*, pages 255–264. IEEE Computer Society, Los Alamitos, Calif., 2003.

[130] C. Teuscher. *Amorphous membrane blending: from regular to irregular cellular computing machines.* PhD thesis, EPFL, 1015 Lausanne, 2004. Thesis 2925.

[131] C. Teuscher, D. Mange, A. Stauffer, and G. Tempesti. Bio-inspired computing tissues: Towards machines that evolve, grow, and learn. *BioSystems*, 68(2–3):235–244, February–March 2003.

[132] Y. Thoma. *Tissu Numérique Cellulaire à Routage et Configuration Dynamiques.* PhD thesis, EPFL, 1015 Lausanne, April 2005. Thesis 3226.

[133] Y. Thoma, G. Tempesti, E. Sanchez, and J.-M. Moreno Arostegui. POEtic: An electronic tissue for bio-inspired cellular applications. *BioSystems*, 74(1-3):191–200, August-October 2004.

[134] J. M. Tour. *Molecular Electronics: Commercial Insights, Chemistry, Devices, Architecture and Programming.* World Scientific Publishing Company, New Jersey, 2003.

[135] J. M. Tour, W. L. Van Zandt, C. P. Husband, S. M. Husband, L. S. Wilson, P. D. Franzon, and D. P. Nackashi. Nanocell logic gates for molecular computing. In *Nanotechnology, IEEE Transactions on*, volume 1 of *2*, pages 100–109, June 2002.

[136] J. von Neumann. *Theory of Self-Reproducing Automata.* University of Illinois Press, Illinois, 1966. Edited and completed by A. W. Burks.

[137] B. Warneke, M. Last, B. Liebowitz, and K. S. J. Pister. Smart dust: Communicating with a cubic-millimeter computer. *Computer*, 34(1):44–51, 2001.

[138] R. Waser, editor. *Nanoelectronics and Information Technology: Advanced Electronic Materials and Novel Devices.* John Wiley & Sons, Inc., New York, NY, USA, 2003.

[139] J. D. Watson and F. H. C. Crick. A structure for desoxyribose nucleid acid. *Nature*, 171:737–738, 1953.

[140] D. Wiklund and D. Liu. SoCBUS: Switched network on chip for hard real time embedded systems. In *IPDPS'03: Proceedings of the 17th International Symposium on Parallel and Distributed Processing*, page 78.1, Washington, DC, USA, 2003. IEEE Computer Society.

[141] Xilinx. SPARTAN® *and* SPARTAN®*-XL Famillies Field Programmable Gate Arrays*, DS060 (v1.7) edition, June 2002.

[142] Xilinx. SPARTAN®*3 FPGA Familly: Complete Data Sheet*, DS099 (v2.2) edition, May 2007.

[143] Xlife program download page: `http://surf.de.uu.net/zooland/download/packages/xlife/xlife-3.0.tar.gz`.

[144] X. Yao. Evolving artificial neural networks. *Proceedings of the IEEE*, 87(9):1423–1447, Sep. 1999.

Acronyms

ACE	Automatic Computing Engine
ALU	Arithmetic Logic Unit
ASIC	Application-Specific Integrated Circuit
CA	Cellular Automata
CAN	Control Area Network
CISC	Complex Instruction Set Computer
CLB	Configurable Logic Block
CMOS	Complementary Metal-Oxide-Semiconductor
CPLD	Complex Programmable Logic Device
CPU	Central Processing Unit
DCM	Digital Clock Manager
DNA	Deoxyribonucleic Acid
DPI	Dots per Inch
DSCA	Data and Signals Cellular Automaton
DSP	Digital Signal Processor
EDVAC	Electronic Discrete Variable Automatic Computer
ENIAC	Electronic Numerical Integrator And Computer
EPIC	Explicitly Parallel Instruction Computing
FPGA	Field Programmable Gate Array
GALS	Globally Asynchronous, Locally Synchronous
GND	Ground

I/O	Input/Output
IC	Integrated Circuit
IP	Intellectual Property
MCU	Microcontroller unit
MSB	Most Significant Bit
LCD	Liquid Crystal Display
LED	Light-Emitting Diode
LSB	Least Significant Bit
LSL	Logic Systems Laboratory
LUT	Lookup Table
LVDS	Low Voltage Differential Signaling
NoC	Network-On-Chip
OLED	Organic Light Emitting Diode
OS	Operating System
PC	Personal Computer
PCB	Printed Circuit Board
PDA	Personal Digital Assistant
PLA	Programmable Logic Array
PWM	Pulse-width modulation
RAM	Random Access Memory
RGB	Red Green Blue
RISC	Reduced Instruction Set Computer
ROM	Read-Only Memory
SiP	System in Package
SoC	System-On-Chip
SP-CSP	Stacked Package-Chip Size Package
SR	Shift Register
SRAM	Static Random Access Memory
TDM	Time Division Multiplexing

USB	Universal Serial Bus
VHDL	VHSIC Hardware Description Language
VHSIC	Very-High-Speed Integrated Circuit
VLIW	Very Long Instruction Word
VLSI	Very-Large-Scale Integration

List of Figures

List of Tables

Contents