UNIVERSITATEA POLITEHNICA DIN TIMIŞOARA

PHD THESIS

# SELF-REPAIRING MEMORY ARRAYS INSPIRED BY BIOLOGICAL PROCESSES

LUCIAN PRODAN
COMPUTER ENGINEERING DEPARTMENT
POLITEHNICA UNIVERSITY OF TIMISOARA
TIMISOARA, ROMANIA

**Thesis Director:** Prof. Mircea VLADUTIU, UPT

OCTOBER 14, 2005

# ABSTRACT

Maintaining the strong momentum of the scientific and technological advances from the past several decades proves to be especially difficult when computing is concerned. Its evolution of modern computing equipment has to pursue a twin track: some applications require *speed* above anything else, while others require the highest possible *reliability*. However, regardless the priority target, classic systems appear to have approached their limits, therefore fueling the need for new computing paradigms and architectures. Emerging technologies are set to explore the potential of biologically-inspired computing and quantum computing, which will hopefully lead to new avenues for progress in computing, both from a paradigmatic and an architectural standpoint; these emerging technology vectors were acknowledged in a recent ITRS (International Technology Roadmap on Semiconductors) report.

As the computing systems gain in architectural complexity, with an ever increasing number of processing units being integrated onto the same silicon die through technology scaling, the idea of drawing inspiration from biology unveils a range of potential benefits. On one hand, how to design and manage such complex computing systems in order to provide both performance and fault tolerance could find appropriate answers through studying the living systems. Their complexity is apparent both in their sheer numbers of parts (for instance, a human being is made of $10^{13}$ cells), and in their behaviors: living beings are endowed with *robustness* to damage (they are able to develop and operate correctly despite assaults from the environment or genetic mutation) and are highly *dynamic* (cells die and are replaced, they fit the environment through different phenotypes). Exploring the mechanisms that underlie these attributes of the living systems in computing is the major goal of the Embryonics project, a long-term research initiative that provides the framework for this thesis.

The research carried over the Embryonics project led to the development of a new, architecturally uniform, bio-inspired FPGA, called MuxTree, with built-in self-test and self-repair. While allowing universal computation and featuring a hierarchical strategy of achieving fault-tolerance, this architecture was ill-suited for the implementation of memory structures required by micro-programmed machines. The first main goal of this thesis was therefore to expand the MuxTree architecture in order to allow an efficient and flexible way of data storage, while preserving its bio-inspired self-repair strategy.

The main challenge of introducing a memory structure in the MuxTree design was integrating it and the required additional features with the already proven mechanisms of growth, differentiation, self-test and self-repair. The functional characteristics of each MuxTree programmable element are determined by the genetic information stored inside a special purpose configuration register. We chose to better exploit this resource by introducing a new operating mode, which we called the *memory mode*. The previous operation was preserved under the name of *logic mode*, while in *memory mode* the MuxTree element allows the implementation of memory structures of variable dimensions, through chaining the configuration registers into what we called a *cyclic memory*. Access to genetic information is sequential (as is in living beings) and there is a choice between maximum data storage and no information routing (*long memory mode*), and minimum data storage with information routing capability (*short memory mode*). However, the addition of the new operating mode raises some issues:

- the original self-testing mechanism preserved from the MuxTree design, cannot just simply extend to obtain a self-testable memory structure;
- an original fault tolerance strategy has to be developed in order to expand the robustness of the Embryonics concepts over the memory structures.

The second goal of this thesis was to provide the newly introduced memory structures with a mechanism of self-test that could be integrated with the extant two-level self-repair strategy. The motivation lies in the different nature of what was to be protected: information recovery, required by the protection of data (when in memory mode), is ensuring a correct functionality (when in logic mode). Choosing the appropriate error recovery strategy in case of memory structures is argued by the occurrence of soft errors, which are subject to influence normal operation of applications targeted by the Embryonics project. A considerable challenge represented the integration of the self-testing mechanisms, for the logic and the memory modes, with the hierarhical strategy of self-repair.

After evaluating several strategies for error recovery in memory structures, we decided in favor of protecting each memory structure through a Hamming-type single error correcting code. A thorough analysis over the reliability of MuxTree machines is also provided, for the considered strategies. This is further extended through the assessment of the accuracy threshold, a parameter borrowed from another fault-rich computational environment represented by quantum computing, shown to also work well for the Embryonics project. In fact, the accuracy threshold parameter can be used in order to estimate the upper bound of the error frequency that would still allow a successful error recovery. Furthermore, a methodology of implementing concatenated coding in Embryonics for the purpose of further extending the robustness of computational processes is presented.

The initial goals of the thesis were met and the circuit was implemented in actual hardware, using the Embryonics demonstrator platform. Several memory configurations were tested in order to demonstrate both a correct functionality and the successful integration with legacy mechanisms.

# REZUMAT

Menținerea puternicului avânt înregistrat în ultimele decenii, atât din punct de vedere ştiinţific, cât şi din punct de vedere tehnologic, devine o sarcină în mod particular complicată în ceea ce priveşte echipamentele de calcul. În mod necesar, evoluția calculatorului modern trebuie să abordeze două direcții: în timp ce o categorie de aplicații necesită cu prioritate *viteza* maximă de calcul, altele revendică în primul rând caracteristici optime de *fiabilitate*. Indiferent de prioritate însă, sistemele clasice de calcul se prezintă ca fiind foarte aproape de limitele lor, alimentând în consecință necesitatea investigării unor noi paradigme și arhitecturi computaționale. În acest context, tehnologiile emergente țintesc explorarea potențialului calculului bioinspirat si a celui cuantic, direcții prin care este de presupus că se va ajunge la noi posibilități de progres în tehnica de calcul, atât din punct de vedere paradigmatic, cât şi din punct de vedere arhitectural; vectorii reprezentând aceste tehnologii emergente sunt menționați în recentul raport ITRS (International Technology Roadmap on Semiconductors).

Pe măsură ce sistemele de calcul câştigă în complexitatea arhitecturală, un număr mereu crescând de unități de procesare putând fi integrate pe acelaşi substrat de siliciu prin scalare tehnologică, ideea de a beneficia în domeniul calculatoarelor de inspirație din domeniul biologiei dezvăluie o întreagă paletă de potențiale beneficii. Pe de o parte, modalitatea de a realiza şi gestiona un design al unui sistem de calcul de o asemenea complexitate astfel încât să fie întrunite dezideratele de performanță şi toleranță la defectare ar putea fi inspirată de studiul sistemelor vii. Complexitatea acestora transpare atât prin numărul masiv de componente (de exemplu, se estimează că o ființă umană conține aproximativ $10^{13}$ celule), cât şi prin comportament: ființele vii sunt *robuste* (sunt capabile să se dezvolte şi să opereze în mod corect în pofida acțiunii nefaste a mediului înconjurător sau a mutațiilor genetice) şi sunt extrem de *dinamice* (celulele mor şi sunt înlocuite de altele noi, prezintă o mare capacitate de adaptare la mediu prin diverse fenotipuri). Explorarea mecanismelor care determină toate aceste atribute ale ființelor vii reprezintă ținta majoră a proiectului Embryonics, proiect de cercetare de lungă durată care asigură platforma acestei teze.

Cercetările întreprinse asupra proiectului Embryonics au condus la dezvoltarea unui nou circuit programabil de tip FPGA, denumit MuxTree, având o arhitectură uniformă şi înglobând facilități de auto-testare şi auto-reparare. Deşi permițând procese de calcul universal şi implementând o strategie ierarhică în scopul obținerii unui grad superior de toleranță la defectare, această arhitectură este mai puțin adaptată implementării unor structuri de memorie, care sunt necesare pentru orice sistem de calcul microprogramat. De aceea, prima țintă majoră a acestei teze a fost extinderea arhitecturii MuxTree în scopul de a ingădui o stocare flexibilă şi eficientă a datelor, concomitent cu păstrarea neafectată a strategiei ierarhice de autoreparare existente.

Principala provocare a introducerii unei structuri de memorie în designul MuxTree a fost integrarea acesteia, împreună cu mecanismele adiționale necesare, în peisajul mecanismelor existente, cu funcţionalitate anterior verificată, reprezentate de creştere şi diferenţiere celulară, auto-testare şi auto-reparare. Caracteristicile funcţionale ale elementului programabil MuxTree sunt determinate de infomaţia genetică stocată în interiorul unui registru de uz special, numit de configurare. Am ales să exploatăm mai eficace această resursă prin introducerea unui nou mod de operare, denumit *modul memorie*. Modul anterior de funcţionare a fost conservat sub denumirea de *modul logic*, în modul memorie fiind posibilă implementarea unor structuri de elemente MuxTree de dimensiuni variabile, prin asamblarea registrelor de configurare în ceea ce am numit *memorie ciclică*. Accesul la informaţia genetică este realizat în mod secvenţial (într-un mod asemănător celui din fiinţele vii) iar flexibilitatea este asigurată prin posibilitatea de selecţie între două modalităţi, modul lung de memorie (spaţiu maxim de stocare, rutare limitată a informaţiei) şi modul scurt de memorie (spaţiu redus de stocare, rutare normală a informaţiei). Introducerea acestui nou mod de operare a fost însă însoţită de apariţia unor probleme:

- mecanismul original de auto-testare, prezervat din precedentul design MuxTree, nu poate fi extins în mod direct pentru a obţine o structură de memorie auto-testabilă;
- este necesară o strategie originală de obţinere a toleranţei la defecte pentru a extinde robusteţea conceptelor Embryonics asupra structurilor de memorie.

Cea de-a doua ţintă majoră a tezei o constituie adăugarea asupra noilor structuri de memorie a unui mecanism de auto-testare care să poată fi integrat în strategia existentă de auto-reparare pe două niveluri. Motivaţia este argumentată de natura diferită a ceea ce se doreşte a se proteja: refacerea informaţiei, necesitată de structurile de memorie determină, în acelaşi timp şi o corectă funcţionalitate a structurilor operând în mod logic. Alegerea unei strategii corespunzătoare pentru revenirea din eroare, in ceea ce priveşte structurile de memorie, a fost determinată de caracteristicile erorilor de tip soft, susceptibile de a influenţa corecta funcţionare a aplicaţiilor-ţintă ale proiectului Embryonics. O provocare considerabilă a fost constituită de integrarea mecanimelor de auto-testare, pentru modurile memorie şi logic, în strategia ierarhică de auto-reparare.

După evaluarea diferitelor strategii de revenire din eroare în ceea ce priveşte structurile de memorie, decizia luată a fost protejarea acestora prin intermediul unui cod Hamming, corector al erorii singulare. O analiză completă asupra caracteristicilor de fiabilitate ale arhitecturii MuxTree a fost prezentată pentru strategiile considerate. Aceasta este extinsă prin investigarea unui parametru numit prag de acurateţe a computaţiei, transferat dintr-un alt mediu computaţional cu o bogată susceptibilitate la erori, şi anume calculul cuantic, care s-a dovedit a fi potrivit inclusiv proiectului Embryonics. De fapt, acest parametru poate fi utilizat în scopul cuantificării limitei superioare a frecvenţei erorilor pentru a permite în continuare desfăşurarea cu succes a proceselor de revenire din eroare. Mai mult, teza prezintă şi o metodologie de implementare a codurilor concatenate în Embryonics, în scopul de a extinde robusteţea proceselor computaţionale la nivelul acestei platforme.

Ţintele iniţiale ale tezei au fost îndeplinite, iar noul design MuxTree a fost implementat in hardware, utilizând platforma demonstrativă existentă. Au fost realizate şi testate câteva configuraţii de memorie pentru a demonstra atât corecta lor funcţionare cât şi integrarea de succes în paleta de mecanisme prezente în designul anterior.

# ACKNOWLEDGEMENTS

The coming of a PhD thesis is granted by a mixture between two essential ingredients: the culmination of several years of study and the ever understanding uphold from family, teachers and friends. Since it would be impossible to individually thank all the people who deserve it, I feel compelled to make a final appeal to their goodwill. I will, however, acknowledge my gratitude to those who made this thesis possible and adorned these years with their unconditional support.

First, of course, my parents, who have always been willing to endure considerable deprivations to provide me with the best in life. My wife, who beared calmly with my frequent periods of being totally immersed in research. They kept me surrounded constantly with all their love and caring; needless to say, without their efforts, it would have been impossible to stand where I am today.

My mentors, from two different countries, united by the same dedication for the academia, who contributed to my becoming both through their friendship and professional excellence:

Professor Mircea Vladutiu, the thesis director and a true connoisseur of seeding the spiritual urge that drives any research. It is he whom I owe the introduction to the joyful realms of computer hardware design.

From abroad, the extraordinary team of the Logic Systems Laboratory, led by Professor Daniel Mange, who fathered my access to the exquisite world of the Embryonics project during my research activity at the EPFL – the Swiss Federal Institute of Technology at Lausanne. Professor Gianluca Tempesti, who shared with me the ups and downs walking along the MuxTree development path, both as my direct supervisor and, later, as my colleague. My integration in the "ontogenetic team" unveiled, apart from some of the more "secret" insights of the project, a more profound perspective over human relations, which made the research efforts in their company never a burden, but always a privilege.

The experts that stood the burden of reading and appraising the effort put into this thesis: Professor Mircea Petrescu from the "Politehnica" University of Bucharest, and Professor Sergiu Nedevschi, from the Technical University of Cluj-Napoca.

If this thesis builds on scientific results, it is because means of encouragement were found in critical moments. I would like to take this opportunity to thank the rector of our university, Professor Nicolae Robu, for the significant support he provided in order to attend key conferences.

Last, but not least, my colleague Mihai Udrescu, with whom I shared the office and catalyst of many fruitful exchanges of ideas. My appreciation also goes to our students, especially to Nicola Velciov. If tight deadlines could be met, it is also thanks to their contribution.

The research carried out throughout this thesis was supported in part by:
- the Swiss National Foundation under grant 21-54113.98;
- the Consorzio Ferrara Richerche, Università di Ferrara, Ferrara, Italy;
- the Leenards Foundation, Lausanne, Switzerland;
- the Romanian Ministry of Education and Research through grant CNCSIS nr.643/2005.

During times when funding becomes more and more scarce for everything that does not seem to have an immediate industrial application, it is inciting to know that long-term research still remains possible.

# TABLE OF CONTENTS

# CHAPTER 1

# INTRODUCTION

It is difficult, if not impossible, to think of modern ages without considering computers. Throughout their relatively short history computers have shown an evolution both dramatic and dynamic. Born as a consequence of the ever-inquisitive human spirit, computing systems have since shown a continuous development process driven by necessity.

The modernity of a certain computer is sometimes measured by referring to its *generation*. First generation computers, based on electronic tubes, were unique machines, occupying quite a large area; they are now obsolete and, compared to present days computers, they certainly appear as ridiculous. But they will forever be present in computing history as its first milestone. A crucial moment that allowed spectacular progress came was the introduction of the transistor. This set another milestone in computing, showing a green light for a second, more powerful generation of computers. By this time, the concepts of designing and implementing a computer were also emerging with John von Neumann setting another milestone with the architecture that now carries his name. Present days computers have reached such levels of design complexity that could only be dreamed of in the past. The strong momentum of technological advances in the last three decades allowed a continuous process of shrinking the transistors on the silicon wafer, thus enabling a larger crowd of devices onto the same chip. What initially started as experimental technology has now reached sub-micron levels (with the 90nm process now available) and is still going down, unveiling new problems that, according to Gigascale Silicon Research Center, can be grouped into problems of the small (caused by the transistor dimensions on the silicon wafer), problems of the large (caused by the process of designing and testing a device containing such an enormous number of transistors), and problems of the diverse [30]. The industry's focus on problems of the small, which have been thus far dominant, is now shifting on problems of the large, as they need to be efficiently solved in order to sustain further growth.

Though solidly set towards evolution, the difficulty computers experience along this track comes from it being manifold: some applications require speed above anything else, while others require the highest possible reliability; unfortunately, computers themselves can fully fulfill none of the requests, thus fueling the need for different and better suited designs. But stretching the limits of human skill and creativity to make better computers has nowadays become a twofold process, by necessity: while solving technological problems involved is certainly essential, this is pursued in parallel with a quest for new inspiration in their *design*, both in software and in hardware.

In its continuous strive to make computing systems run faster and be more reliable, mankind is now looking for new, as yet unexplored, computing paradigms as classic systems appear to have approached their limits. Two inspiring sources seem to be within our grasp.

Nature may very well provide such inspiration, exhibiting biological solutions that are omnipresent and, considering the time spent for evolving them, as close to perfection as possible (though the optimality of nature's mechanisms is perhaps arguable from an engineering perspective). In a context of scientific rush at both design level (to build new, innovative computers) and paradigmatic level (generating new algorithms, running on new principles), nature presents a wealth of inspiration. The idea of importing biological features into human-made machines is not at all new, a variety of robots being part of everyday life: able to meet requirements such as brute force, flexibility and reliability, they have become indispensable. Yet all living organisms are provided by nature with at least two special features – self-repair and self-replication – that still remain inaccessible to current machine design. At the paradigmatic level, a new focus on the laws of physics could once again be put to work in order to achieve quantum computing, which could further enhance the computing forces at our disposal. These are two directions that show the potential of setting new and awaited milestones by producing, if not revolutionary, then at least some evolutionary effects over modern computing. This thesis is concerned with the first one, as part of the development of a much larger, bio-inspired research project.

This introductory chapter will briefly present the motivations that lie behind this thesis and some of the basic features we wish to introduce, together with a brief outline of the overall structure of the thesis itself.

## 1.1 Motivations

The current state-of-the-art in computing stands, undoubtedly, for the quest to achieve dependable, fault tolerant systems while preserving the raw performance. Human-built machines tend to show little flexibility and brittle responses, not being quite able to cope with the exposure to a dynamically changing environment, all these in deep contrast to the wonderfully adaptable biological systems built by Nature. The surrounding environment, a huge repository of various solutions constantly tested along many million years, made engineers aware they could draw some inspiration from it. History records indicate there was a clear separation between the two categories of scientists and engineers – while scientists aimed to a better understanding of things, and ultimately Nature, engineers tried to imitate it when creating tools. However, the modern era is narrowing the gap by forcing them to work together, in a symbiotic process: during research activity, scientists need tools created by engineers, while engineers use scientific knowledge in the process of tool creation [67].

There is no doubt Nature has developed and thoroughly tested its solutions in time; such a long time that it would be impossible for engineers to replicate the process and allocate the same amount for testing. And considering dependable, fault tolerant systems, there is no closer source of inspiration than Nature itself, proven and endorsed by the uncountable variety of living beings.

However, engineers face a dilemma when trying to implement Nature's solutions: while studying natural sciences is an essentially an *analytic* process, creating devices that would exhibit some of the most interesting features found in living organisms happens to be essentially a *synthetic* process [119]. There are at least three major issues:

- would adapting mechanisms from nature in engineering also replicate the results?
- were exactly would the two of them, Nature and science/engineering, meet?
- finally, is the process of exporting biological features in computer engineering technically possible?

These are some of the questions both engineers and scientists are strongly arguing about, for the benefits seem to be very well worth the effort [99]. The potential of biologically-inspired and quantum computing architectures is acknowledged by the ITRS report on emerging technologies (see Figure 1-1) [169]:

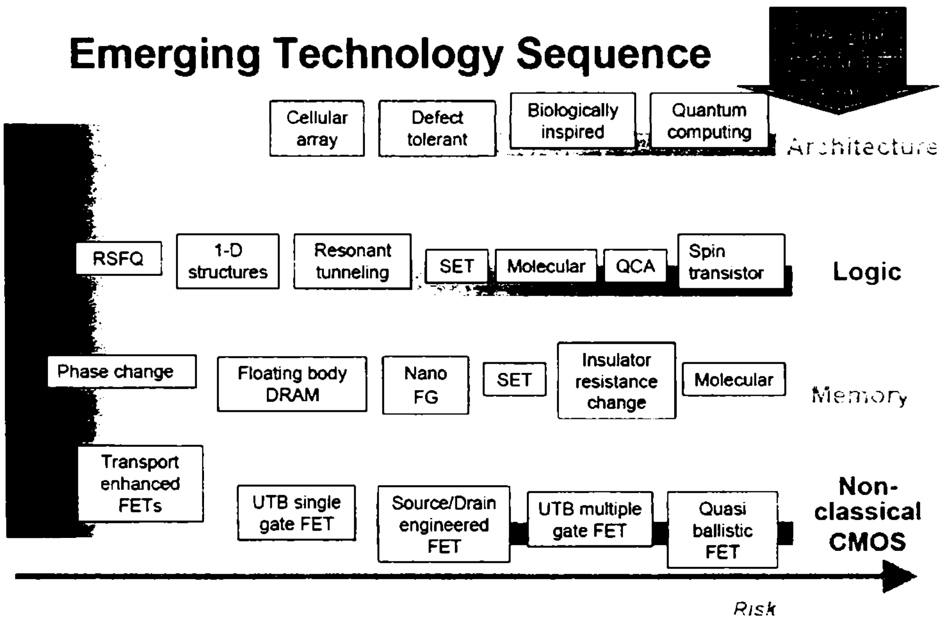


Figure 1-1: Emerging technology sequence [169].

## 1.2 The Legacy

Answers to the aforementioned issues came along with the advent of bio-inspired digital systems, pioneered by John von Neumann as a brilliant scientist who will mark the history of mankind with his achievements. Not only a gifted mathematician, he extended his ingenious, sharp mind over a variety of fields, including him being part of the Project Manhattan and later the father of the modern computer architecture, the IAS machine. He also considered the aspects involved by the reliability of computing systems [80, 81] and later in his life he found a great interest in what he called the *theory of automata* [56, 61, 79, 112, 120], inspired by the similarities and differences between the *artificial automata* (computers) and *natural automata* (biological organisms).

Parenting the concept of the stored program at the Institute for Advanced Study in Princeton was driven by his perception that computers could be used successfully to applied mathematics for specific problems. Aware of the vast parallelism involved in the biological systems, but also of many issues that would have to be solved in order to take any advantage of it, he preferred to design a sequential architecture for the IAS machine.

Always looking for a synergy between a range of fields (in which von Neumann certainly didn't lack expertise), his last research effort before his untimely death in 1957 focused on the theory of automata which led to a set of principles an artificial system with biological attributes would have to follow. He began investigating the similarities between a computer and a nervous system [78] and later proposed an architecture for a self-replicating system consisting of two parts [79]:

- *A universal constructor.* Von Neumann's view was that self-replication should come as a particular case of construction universality. In other words, such a self-replicating machine should be able not only to construct identical copies but any machine, given an appropriate description.
- *A universal computer.* The computing capabilities of the machine should extend over any finite program; A. Turing defined such a class of automata, now known as the universal Turing machines [80].

Von Neumann's research was never completed, but his theory of automata may be considered even today as an inspiration in developing bio-inspired systems.

### 1.2.1 The POE Model

Intricate behavioral patterns are exhibited by all members of the biological world, the marvel of life hiding inside them being so different and yet so similar to each other. All the features exhibited by living beings come as the result of a continuously running evolutionary process that can be considered as taking place along three axes [67, 119]:

- The very first level of organization concerns the temporal evolution of the genetic program. Called phylogeny (P), it is the result of a non-deterministic, low-error rate reproduction process of the genome, thus giving rise to biological diversity.
- The second level of organization is concerned by the temporal evolution of a multicellular organism, from the early stage of the primordial cell (the zygote) to the final, mature organism. Called ontogeny (O), this is essentially a deterministic, low-error rate process that includes two distinct processes: cellular division and cellular differentiation.
- The third level of organization arises from the apparently insufficient capacity of the previous levels to integrate complex structures such as the nervous, immune or endocrine systems. Furthermore, the changing topologies of these systems throughout a normal individual's life also support the concept of such a superior level of organization, called epigenesis (E).

If each level is represented by an axis (Figure 1-2), this leads to partitioning the space of all living beings, the result being what is called *the POE model.* All of the above considered, the taxonomy of artificial, bio-inspired systems may be regarded as falling over the POE model, containing the same three directions along which natural, biological evolution guides itself, in a quasi-perfect analogy [107, 119].

Using the taxonomy provided by the POE model, one can investigate the existence of a variety of bio-inspired systems that belong mostly to a certain axis or plane. Phylogenetic processes can be observed under the form of evolutionary processes, such as genetic algorithms (GA), genetic programming (GP), widely known as evolutionary algorithms (EA). There exists both software (in the form of a population of artificial individuals, used to evolve an acceptable solution by using genetic operators such as cross-over and mutation) [35, 41] and hardware (evolutionary algorithms applied to the synthesis of digital circuits) implementations [109].

Epigenetic processes are implemented, both in software and hardware, as artificial neural networks (or ANNs) of two types: learned systems (that feature an instinctive behavior and limited generalization capability) and learning systems (that adapt continuously to a dynamic environment and feature a high generalization capability). Though hardware implementations of ANNs remain a very small minority compared to software, it is worth mentioning that new momentum in designing bio-inspired systems endowed with epigenetic processes in hardware was gained through using the latest generation of programmable circuits, the field programmable gate arrays (the FPGAs) [89, 90].

Ontogenetic processes can be primarily assimilated to growth or construction, through the processes of cellular division and cellular differentiation. These show a critical importance to the world of digital electronics since one of their direct consequences is the ability of self-repair. Pioneered by John von Neumann, the research along the ontogenetic direction can be viewed over several stages [67]; while the majority of implementations concern unicellular automata, there is a long-term research project that adapts the cellular processes by proposing an ontogenetic hardware architecture, called *Embryonics*.

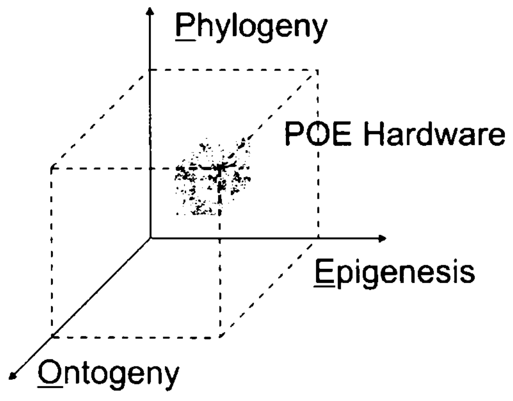


Figure 1-2: The three axes of the POE model.

### 1.2.2 The Embryonics Project

The main purpose of Embryonics (a contraction of *embryonic electronics*) is to attempt to build a bridge over the existing gap between the worlds of biology and electronics. Perhaps seeming totally disjoint at a first glance, the two worlds are in fact not that dissimilar; as argued in the previous section, each of the three directions of the POE model were actually considered already (at least theoretically) for implementations. Therefore any of the insights revealed from this profound exploration process should be carefully analyzed and considered for a potential adaptation and/or insertion into new mechanisms over silicon devices. The entire Embryonics project is generously dimensioned to accommodate the design and implementation of novel bio-inspired hardware, including (but not limited to) the ontogenetic axis [118].

Ontogenetic processes that take place in all multicellular living beings are believed to be driven by the *genetic program*, a copy of which is present into each cell, be it muscle, nerve or any other tissue. Its relentless execution expands the runtime period over the entity's entire lifetime, and is by itself an example of perfection. The genetic program is not executed in its entirety, a complex mixture of factors partitioning it into groups of components called *genes*, and also determining which genes will be executed by one particular cell. By this process of selecting the genes to be interpreted and executed, nature has implemented a mechanism that transfers the partitioning from the genetic program to the cellular level. As a consequence, by executing different parts of the genome, cells develop different types of functionalities through a process known as cellular differentiation. Other natural mechanisms such as healing or reproducing may be regarded as facets of the same genetic program.

The ideas that lie behind the Embryonics project extract their substance from the fascination of this most powerful program also known as the *genome*. After years of continuous research and refinement, the Embryonics project has reached a mature stage, its central dogma being a four-level architecture (shown in Figure 2-2). The most basic brick in Embryonics is the *molecule*, essentially a reconfigurable circuit capable of universal computation; molecules are assembled to make *cells*, which in turn make up *organisms*, thus achieving multicellular organization, and finally, a *population* of organisms. The strength of Embryonics comes from the fact that each cell stores a complete copy of the genetic program of the organism it belongs to, the only differences between two different cells being the portions of the genome each executes, through a coordinate-based mechanism that ensures cellular differentiation.

There are several key features that are vital for Embryonics:

- *The multicellular architecture* allows the construction of complex entities, built of identical, simple cells that are built by even simpler molecules. As the levels increase in the organization axis (Table 1-1), so does the complexity of the corresponding entities. The situation is borrowed from biology but there are, of course, many aspects that prevent these *similarities* from actually becoming *identities*: we do not have access to the complex chemical plant Nature uses to assemble its molecules and cells; instead we have to *provide* the artificial environment as an empty, non-functional array of reconfigurable logic, that will be subject to pseudo-biological mechanisms in order to assemble our artificial cells from artificial molecules. Furthermore, an artificial organism's performance is given by the sum of the functionalities of its own cells that operate in *parallel*; this is also borrowed from biology where living beings exploit a massively parallel operating of their components.
- *The ontogenetic processes* give biological entities their incredible degree of robustness; during its normal life, a living organism suffers temporary illnesses and wounds that are normally cured in time. The temporality of these environmental aggressions is ensured by the two processes mentioned before, namely cellular division and cellular differentiation. Dead cells from broken tissues are successfully replaced by newly-born ones, whether they originate directly from an identical cell or indirectly by a process of modified behavior as a result of a differentiation process that takes place in the gene

expression process: given that any cell stores a *complete* copy of the genome, a cell can become of *any* type by simply executing the right genes.

While the similarity with their biological counterparts is one of the main purposes of the entire project, these vital features of Embryonics also share the names from the world of digital computerware. Therefore, one can say that the biological mechanism of cellular division has as its direct consequence in Embryonics the mechanisms of self-replication, while the cellular differentiation from biology takes the form of self-repair mechanisms.

| Biology | Electronics |
|---|---|
| Multicellular organism | Parallel computer system |
| Cell | Processor |
| Molecule | FPGA Element |

Table 1-1: Analogies present in Embryonics [128].

### 1.2.3 A Plea for Bio-Inspiration

Emergent behaviors observed in biological organisms demonstrate their intrinsic robustness. On one hand, wounds and illnesses (*faults* and *errors* in artificial systems) are not rare at all, while on the other hand the overall activity of the organism (the *functionality* of the artificial system) remains virtually unaffected. All these happen because of the self-diagnosing and self-healing capabilities that take place ceaselessly inside living beings (the power of replacing damaged cells and tissues with newly fabricated ones has the consequence of being able to heal quickly). While this was successfully implemented and then perfected by Nature over a very large period of time, the power of fabricating new things is something that human-made machines certainly lack (even considering this as feasible, dedicating the same time to adapt and perfect the necessary mechanisms as Nature did is obviously impossible). On the other hand, bringing a robustness degree to artificial systems, i.e. incorporating fault tolerance, might not necessarily have to rely on actually fabricating the required basic bricks: they could lie inside the system, passively assisting its normal operations and only becoming active when there is a fault detected, as part of what is called *redundancy*.

Self-healing mechanisms from nature have a direct correspondent in artificial systems where they are called self-repairing mechanisms. The redundancy feature cannot be separated from the possibility of reconfiguring the system. Depending on how redundancy is achieved two types can be distinguished: *hardware* redundancy, making use of multiple replicated resources, and *time* redundancy, making use of multiple tasks running in parallel for the same goal [53, 65, 67, 128]. Bio-inspired redundancy is one of the researched paths [85, 86], while a bio-inspired fault detection known as *immunotronics* successfully implements a mapping of the biological immune system to the world of silicon [6, 40, 144, 157].

The key aspects and differences between bio-inspired alternatives such as Embryonics [60, 71, 72] and classical designs for fault-tolerance lie in the distributed nature of the reconfiguration process. In hardware redundancy, the faulty resources are taken over by identical spare ones, the physical re-routing processes being usually decided and initiated by some kind of centralized processing unit. While this approach has a proven effectiveness, its centralized nature presents some major disadvantages concerning the scalability and reliability. Centralized controls do not scale well to arbitrarily large networks. Moreover, the entire system is no more reliable than its centralized processing unit; any fault arising here could potentially jeopardize its normal operations and thus leading to the failure of the entire system [39, 44, 46].

## 1.3 Features

All organisms from nature (be them unicellular or pluricellular) store in each of the composing cells their individual and unique genetic program, whose ceaseless operation constitutes the supreme act of living. The features that differentiate living beings from each other are heavily influenced by the genome itself, by its structure and by its size. While our artificial organisms are also genome-driven, expanding bio-inspiration is a dynamic process that not only means adapting biological mechanism in digital electronics but also constantly updating the structure of the genome itself in order to ensure the degrees of efficiency and flexibility required by all newly imported bio-inspired features.

The complexity that can be achieved by using our architecture reflects upon the complexity of the genome's structure, which is made of thee distinct components. The *polymerase genome* is the information required for the initial process of delimiting the space occupied by one cell. This is done by a special mechanism called the *space divider*, which effectively attaches a geographical blueprint to each molecule, the result being a rectangular cluster made of the molecules that make up our cell. The entire process of encoding the cellular boundaries is similar to what can be encountered in nature under the form of the cellular membrane.

The *ribosomic genome* determines the functionality of the entire cell; since a cell may be considered as the sum of its molecular components, the ribosomic genome is an assembly of all binary strings that are used to configure each MuxTree molecule's internal logic [67, 100, 128, 131]. Through a proper configuration of the ribosomic genome, our artificial molecules and cells may implement complex digital machines. There is however an important limitation concerning implementations that make use of any memory structure: the only memory resource present inside a molecule that might be used for such a purpose has a storage capacity of a single bit [128, 131]; as a natural consequence, any form of memory would need a huge number of molecules just for storage purposes, thus all the remaining resources are but a significant waste. A typical example where the ribosomic genome shows insufficient abilities would be any microprogrammed machine.

As a measure of extending the limitations of the ribosomic genome, the *operative genome* [63] has been introduced. It was designed to establish an efficient way of implementing more flexible memory structures, also providing the means to a more complete utilization of internal resources at the molecular level. Although the introduction of the operative genome brings new possibilities in building a memory structure within an Embryonics machine, the operation is far from being a straightforward graft, as there are several issues arising that have to be settled. The existing hierarchical model imposes a distributed-type memory, that is, the memory structure has to be composed of basic memory units. A larger storage space would imply a correspondently larger number of such units; also, it should be possible for independent memory structures to co-exist within the same organism. Furthermore, the memory allocation should offer a reasonable balance between storage capacity and functional attributes (if storage is more important than functionality, then a reduced number of internal resources would be available; the opposite situation is when functionality is most important and the use of internal logic would have to be maximized).

From these observations we can outline the basics of the new structure. We decided in favor of a cyclic-type memory architecture, based on a new operating mode introduced at the molecular level called the *memory mode*. Together, all these make possible the use of a new memory structure, created by effectively chaining molecules operating in memory mode into an extended shift register, structure that we will call *macro-molecule*. In order to provide flexibility in using the macro-molecules and to address the functionality-storage balance issue mentioned previously, two memory operating sub-modes are available for any molecule. The *long memory* mode locks the molecule's communication resources to the north-south, and east-west fixed paths while providing the maximum possible storage space, while the *short memory* mode keeps the communications resources available but offers half the storage space.

The artificial organisms in Embryonics offer superior robustness due to a quite capable self-repair mechanism that stretches over both the molecular and the cellular levels. Such a hierarchical approach for self-repair allows for an effective way of tolerating faults: it has the capacity of reconfiguring according to different severity levels, by addressing the least severe first (represented by faulty molecules), and then the most severe (represented by faulty cells), while it inflicts minimal resource loss through reconfiguration (replacing one faulty molecule is certainly less expensive than replacing an entire cell). Although covering a large area of possible faults, the self-repair mechanism currently expands over the strictly functional parts of each molecule only. Since these are driven by the ribosomic genome, the operative genome (and of course, the macro-molecule) is left aside with virtually no error protection. While ensuring self-repair over the *functionality* is certainly important, the newly introduced memory structures represent per-se a hole in the robust package of Embryonics, as the integrity of the *entire* genome (that also governs over functionality) is no longer fully protected. It is therefore imperative that the existing mechanisms of fault-tolerance be *extended* in order to accommodate the operative genome.

Finally, the introduction of new mechanisms that expand bio-inspiration is also envisaged. The biological process of healing presumes periods of time during which the wounded entity is at least partially incapacitated. We developed a similar mechanism, based on repeated attempts of re-powering a totally incapacitated organism or cell, which we called *unkill*. Based on the fact that repaired faults are often of a transient nature, after a cell "dies" it might be brought to life again by simply re-charging the genome, the success of this process indicating the disappearance of such faults.

## 1.4 Outline

The complexity of such a vast research project as Embryonics has powerful repercussions over the description effort. As the introduction of the new type of genome influences the self-repair, while aspects closely related to self-repair influence the genome, one can safely predict that such a reciprocal dependence will be hard to describe step-by-step, in a linear fashion. However, we will try to keep an as clear separation as possible between the issues this thesis deals with.

Being part of a much larger research direction, many of the conceptual and design choices made in this thesis come as consequences of a continuous development process over the existing design; such a process requires this thesis be a building brick pertaining to a scientifically engineered edifice, rather than a standalone one by itself. Chapter 2 will present an Embryonics overview, expanding section 1.2.2, but also narrowing the focus over the actual design concepts. The essential issues concerning bio-inspiration in digital hardware are introduced in a more detailed context of Embryonics, with examples on how an artificial organism is built out of artificial cells and molecules. Since the Embryonics project always was, and continues to be, a collective effort, we cannot claim originality for the contents of this chapter though we feel such a deeper introduction towards this thesis' goals is absolutely necessary.

Chapter 3 is entirely dedicated to the new memory implementation under the form of the operative genome, whose need was previously justified [128]. Its development and implementation are original and required a substantial amount of research. The chapter begins by introducing the motivation of the problem and then presents the arguments, which favored our decision towards a cyclic-type memory, as opposed to other, more classical architectures. After a description of the new memory organization, the chapter goes deeper into the implementation details and gives examples on the ways storage data can be routed. Details on how the process of growing the cellular membrane (which is achieved with the special mechanism called space divider) is achieved are also given here. The end of the chapter contains coverage of the self-repair mechanism in the new context of the memory-operating mode (with examples of the actual reconfiguration process provided) and the mechanism used for controlling the memory.

Chapter 4 is dedicated to the key aspects concerning memory fault tolerance. It will start with a comprehensive description of the causes and effects of the transient faults and, in particular, of those known under the name of soft fails. Details of the particle physics that may lead to soft fails are presented; though not an original contribution in its essence, we believe the review of soft fails' influence presents significant importance with respect to potential Embryonics applications in hostile environments. As a natural step, we will next describe the process of designing an upgraded self-repairing mechanism (which would protect *both* the ribosomic and the operative genome). Furthermore, an analysis of the methodology of configuring the operative genome in order to obtain a fault tolerant memory is presented. An example of a complete cell, with the operative genome using a Hamming-type, error-correcting coding is also provided.

Chapter 5 contains our conclusions, analyzing the final design with respect to the initial requirements previously motivated in section 1.1 and set in section 1.3. An original view over the bio-inspiration degree of the Embryonics project is given in the light of the last scientific findings in modern biology, concerning the human DNA and the stem cells. The final words are reserved for several considerations on Embryonics; while providing an outsider's view over possible real-life applications may prove considerably challenging for someone deeply involved in such a vast project, we make a last attempt on portraying the future of this project. We hope our ambition would some day be rewarded.

The body of thesis will be followed by an appendix describing details of the implementation.

# CHAPTER 2

# BIO-INSPIRED COMPUTING SYSTEMS

## 2.1. A Brief History of Bio-Inspiration

### 2.1.1 Introduction

Ever since the advent of modern computing systems, the quest for performance seems to have taken intriguing paths: in the beginning it was the rush for more and more raw computing power; as this need was increasingly fulfilled, computing power continued to remain a top priority. Techniques were developed in order to maintain performance progress for various types of digital systems [32, 50, 88, 153, 159]. The performance achieved by computers seems these days somewhat sufficient, transforming the aim of computer designers from a *single* purpose – performance – to *a list* of top priorities. An essential requirement of modern computing systems and part of this list *dependability*, which is a synthetic term involving (as stated by IFIP's working group WG 10.4) a list of parameters such as reliability, fault tolerance, availability, performability, safety, and others; in real world, a dependable system will operate normally over long periods of time before experiencing any fail (reliability), will recover quickly from it (testability, fault tolerance), while the performance level won't drop under a certain, acceptable, level (performability).

The need for dependable computing systems cannot be considered exactly new, but until recently, building such systems was almost always a secondary target. By attaining a sufficient level in terms of performance, an entire range of new applications made it obvious that performance by itself was no longer enough. Such is the case of space applications, which represent a special category for digital computing; long-term exposure to aggressive (even hostile) environments prohibit or limit any human intervention, therefore endorsing the quest for dependable systems, which has turned into a vital requirement instead of a negotiable quality indicator.

Designing dependable and reliable computing systems is one of the modern-day designer's tasks, and any literature survey reveals enormous amounts of work carried in order to ensure these much needed characteristics both at the conceptual level [3] and at the hardware level [22, 23, 93], in particular in FPGAs [31, 43, 77, 131]. An essential part of the dependability equation is played by fault tolerance (covering a variety of aspects such as fault detection and fault recovery). Unfortunately, a sufficiently dependable (this being, of course, application dependent) computer remains an idealistic target so far, proven by a series of unsuccesses of on-board computers at high altitudes and in the outer space (Japan's Nozomi and UK's Beagle-2 probes, both lost on the way towards Mars, are perhaps the most recent pieces of evidence that mankind still has some improvement room left where building dependable equipment is concerned). With current designs being insufficiently dependable, other sources of inspiration may be worth exploring.

A very generous, yet not fully researched and understood, source of inspiration lies very close to each and every one of us: nature. Nature exhibits omnipresent biological solutions, which, considering the time spent for evolving them, are as close to perfection as possible. In a context of scientific progress at both the design level (to build new, innovative computers such as based on bio-inspired architectures) and the paradigmatic level (generating new algorithms, running on new principles, such as molecular and quantum computing), nature presents a wealth of inspiration.

Biological organisms are the most intricate structures known to man, with a highly complex behavior, due to massive, parallel cooperation between huge numbers of relatively simple elements, the cells. And considering dependable, fault tolerant systems, there is no closer source of inspiration than Nature itself, proven and endorsed by the uncountable variety of living beings, with a life span up to several hundreds (for the animal regnum) or even thousands (for the vegetal regnum) of years. Trying to exploit nature's results only seems a natural step.

Present-day technology has pushed the architecture of computing systems towards such levels of complexity that the design of new, innovative computers has become a challenge for human intelligence. Despite not being a new concept, biological inspiration in the field of designing artificial machines still represents a research activity. Recent technological advances combined with the focus shifting from the mechanical world to the realms of information led to re-evaluating the biological inspiration when designing computer hardware. The dream of creating artificial machines featuring the robustness and the efficiency of living beings is even closer to becoming a reality [119].

### 2.1.2 The Road to Bio-Inspiration

The analogy between biology and electronics might appear fuzzy at a first glance [16, 67, 107, 118, 119]. But considering the facts that the function of a living cell is determined by the genome, and that a computer's functionality is determined by the operating program, then the two may be regarded as sharing a certain degree of similarity.

Of course, carbon-based biology is different enough than silicon-based computing, making it quite difficult to establish a straightforward similarity between them (except, perhaps, at a very superficial level). It might be worth mentioning that, while the environment simply *exists* for the biological entities in nature, which depend on it in order to survive and replicate, any artificial mechanism *needs* a suitable environment in order to operate, which is also artificial. Furthermore, no artificial mechanism is as yet capable of a similar performance as everyday natural processes of birth and growth; what in biology is possible with the aid of complex chemical processes must be provided initially in electronics. However, despite a variety of aspects that sever the connections between the two worlds, some basic biological concepts may appear to encourage their coming closer to each other, and may prove extremely interesting from a hardware designer's point of view (such as robustness, evolution, and healing), thus making the research for adapting them into digital devices a real challenge.

Considering the field of designing electronic circuits, bio-inspiration was not overlooked by one of the founders of modern computer engineering, John von Neumann, the parent of the first self-replicating computing machines [61, 69, 105]. Several years before his untimely death he began to develop a theory of automata, which was to contain a systematic theory of mixed mathematical and logical forms, with the aim of contributing to a better understanding of both natural systems and computers [79, 120].

Shortly after John von Neumann passed away, Francis Crick, one of the discoverers of the DNA's structure, enounced the central dogma of molecular biology, that *proteins are not made directly from genes*, the intermediary being the RNA [11, 120]. The DNA is the *container* for the information needed by a biological organism to carry out each of its functions; it is the DNA's *data* that allows the zygote to divide and differentiate to grow the organism as a multicellular identity that will interact with the environment, adapt to its conditions, heal, reproduce, and eventually die. Thus we assist to a certain layering of the roles played by the essential biological molecules [120]:

- DNA is the *carrier* of information;
- RNA is the *messenger* (the elements decoding the RNA genetic information to produce the necessary proteins are called ribosomes [106]);
- proteins are the *executors*.

Considering the biological findings, if any piece of digital hardware is to achieve some degree of bio-inspiration, several biological features would have to find their correspondents in the world of electronics. Similarities exist already: the DNA might be regarded as equivalent to a memory, the RNA as a communication unit, and the proteins as equivalent to parts from a logical unit. However, there is an essential problem, which was solved by Nature from the very beginning, that has yet to find a solution in electronics: proteins are *synthesized* by living organisms from raw materials that are readily available and offered by the surrounding environment, a task that is, at least for the moment, impossible to implement by silicon devices. Instead of synthesizing any proteins, the only available solution is that their electronic equivalents *be supplied* by the artificial environment, thus mimicking the synthesizing process by choosing only the useful parts from the environment through a process of digital configuration. In order to implement such mechanisms, bio-inspired machines have to be able to change their hardware functionality by using information. By consequence, it can be concluded that a piece of bio-inspired hardware would present the following features:

- a memory structure carrying the genetic information and equivalent to the biological DNA;
- a decoding/routing unit that would manage genetic information in a similar way the biological RNA does;
- a functional unit that would execute the genetic program, similar to the biological proteins.

The central dogma of modern molecular biology entails the formula "genotype + ribotype = phenotype". As such, von Neumann could not have explicitly thought about this equivalence, which makes it astounding that the system he developed with no prior knowledge of the DNA structure, still meets this equation. His universal constructor self-replicates similar to the natural process: the tape stores the description for the entire machine (the genotype), it is interpreted by a ribosome (the ribotype), and together they make the machine itself (the phenotype) [61].

### 2.1.3 The Technology

The key technology available, which today allows the development of bio-inspired hardware architectures, can be found in the programmable logic devices, usually referred to as FPGAs (Field Programmable Gate Arrays) [7, 140]. Composed by a two-dimensional array of identical elements, each being capable of implementing a variety of different functions, FPGAs can be used to play the role of just about any kind of digital circuit by individually configuring the functionality of the programmable elements, as well as all the connections between them. They seem to be the ideal platform for developing bio-inspired hardware, which requires the layout of the circuit be changed by mechanisms implementing self-replication, evolution or healing (self-repair) and adaptation to the environment.

#### 2.1.3.1. Characteristics of the FPGAs

Any FPGA device [87] may be decomposed into two essential layers (see Figure 2-1) that give its power and flexibility: the functional layer and the communication layer. The functionality of the device comes from a number of structurally identical elementary units, usually composed of a few gates and flip-flops that can be configured to allow for any kind of combinational and/or sequential implementations (bottom left in Figure 2-1). In order to feed the elementary units and also drive required signals inside the device, the communication layer consists of input-output busses (bottom right in Figure 2-1); furthermore, the direction of driving a signal may be changed by configuring the bus connections, which are themselves configurable. A special case of elementary units are those situated at the boundaries, as these will likely play the role of user-defined IO ports, thus allowing the FPGA to interact with its exterior environment at the expense of a limited functionality.

The internal architecture of an FPGA allows unleashing the inherent parallelism in operating its building bricks. They are small and simple enough (architectural uniformity being also possible) to achieve massively parallel operating (able to perform bit-level operations), yet they are complex enough to allow further organizing, layering, and communicating as separate entities of various shapes and roles (also able to perform systolic operations). Designing with FPGAs is considered in present days one of the most efficient ways of prototyping, testing and evolving new architectures; a special bonus is offered by the possibilities of on-line modifications in a FPGA configuration, with on-line hardware evolution being a very attractive tool in the near future [68]. Much work has been carried out over methods of designing efficiently with FPGAs for the purpose of implementing testable and self-testing computing machines [1, 2, 31, 43, 77, 126].

Another key advantage of the FPGAs lies in the configurable interconnections, thus making them especially suited for bio-inspired design. Efficient and flexible (at least in theory), bio-inspired fault-tolerance requires some sort of hierarchy when dealing with resource reconfiguration. Such a hierarchical mode of configuring the interconnections can be offered by FPGAs; for instance, the 6000 series from Xilinx has been involved in evolvable hardware research because of its suitability for such applications [19, 26]. One of its strong points is a mechanism that provides abundant, hierarchical connectivity between groups of elementary logic blocks (FastLANE™ routing technology) [171], being provided as follows:

- each elementary *cell* connects directly with the four adjacent neighbors;
- an upper layer provides interconnections between *blocks* consisting of 4×4 elementary cells (length 4 FastLANE™);
- another communication layer provides interconnections between *tiles*, i.e. 4×4 blocks, that is, 16×16 elementary cells (length 16 FastLANE™);
- finally, information can be routed chip-wide, between structures of 64×64 elementary cells, or 4×4 tiles (length 64 FastLANE™).

### 2.1.3.2. Bio-Inspiration and Robustness

Inspiration from natural systems in engineering seems appealing due to their extraordinary resilience: living organisms are continuously immersed into a dynamically changing environment (which, to make matters worse, is also non-deterministic), yet they manage to survive, prosper, and reproduce. They succeeded in taming the sometimes harsh and aggressive environment in order to live for such extended periods of time any design engineer would wish to build systems operating for such long and with such efficiency. Therefore, *dependability* would provide in engineering a definition close to what *robustness* actually means in nature.

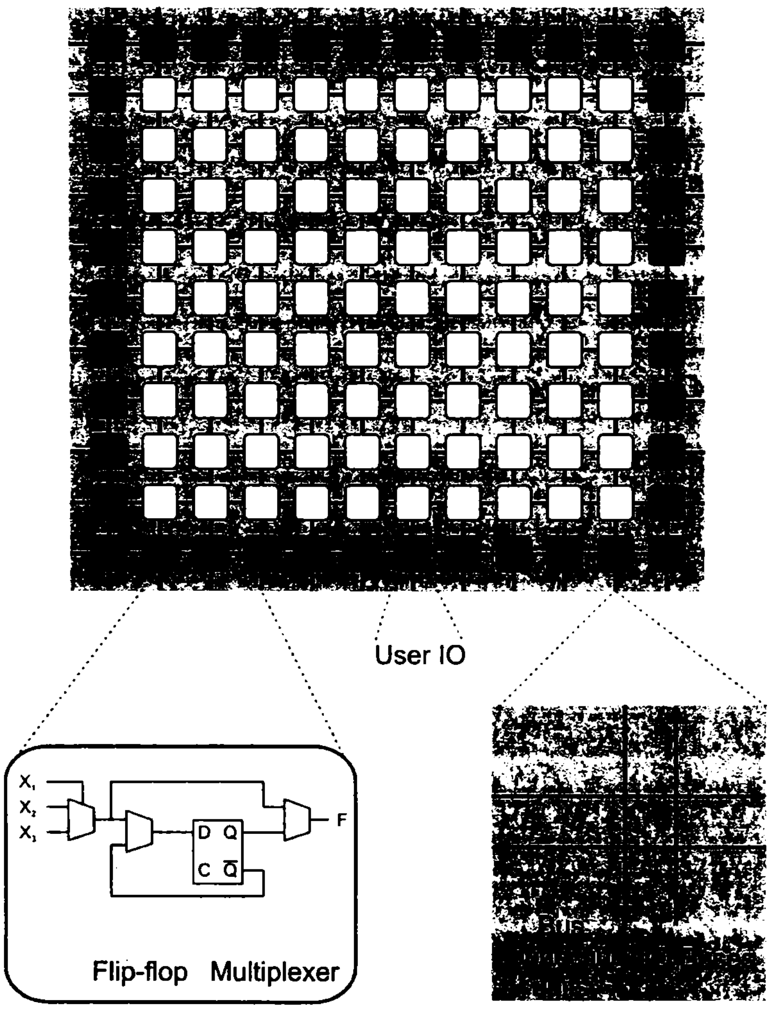


Figure 2-1: Basic structure of an FPGA.

657.637

246 E

In order to implement bio-inspired, dependable systems that would match the robustness witnessed in any biological organism, one has to assess its driving forces by answering at least the following questions:

- what mechanisms make biological entities robust, and
- how can these be implemented in engineering so as to build systems with comparable dependability levels.

Answering the first question requires one to observe how the natural healing processes occur in living organisms, as the key quality allowing them to overcome the damaging effects of not only the surrounding environment, but also of interactions with other organisms, lies in their capacity of *regeneration*. A majority of illnesses and wounds that leave the organism partially incapacitated are successfully cured through regeneration in time. This remarkable curing potential lies in the capability of all living beings to grow new cells that, in the end, will completely replace the cells composing the damaged organ or tissue. An engineering approach, however, would encounter here its first serious obstacle to implement natural healing processes in electronics; if replacing faulty circuits and devices can be implemented from an operational standpoint (through reconfiguration, majority voting or other strategies), *fabricating* new replacement circuits on-line or "regenerating" them remains a matter of fiction. Therefore, bio-inspired digital design would have to content with reconfiguration strategies for *self-repairing*, as only these are, as of yet, available; this also provides a strategy for answering the second question.

However, the ability of all living beings to grow new cells in order to repair damaged organs and tissues includes an additional feature of locating the exact boundaries of the region that needs healing; such a mechanism would be similar to what is called *self-testing* in digital design. But in nature growing new cells is not sufficient *per se* for the healing to be complete; new cells may replace damaged cells only if they are the same, i.e. they share the same structure and functionality. By consequence, bio-inspired self-testing and self-repairing will not be sufficient in order to produce a close imitation of natural healing; faulty artificial cells can be replaced only by new units with the same characteristics.

### 2.1.4. Bio-Inspired Computing

The first computing system that might be regarded as to comprise bio-inspired concepts comes from the very dawn of the computer era and is actually the von Neumann architecture, developed at the Institute for Advanced Study (IAS): the entire system is built around a central processing unit (which may be seen in analogy with a brain), which is connected with a range of peripheral devices (analog to sensory organs and limbs) through a network of buses (analog to the nervous system) and also accesses a memory system (somewhat similar to a genetic memory). The validity of the structural concepts of this architecture was proven by the fact they are used, with little or no change, even today by modern computers. However, if roots of bio-inspiration can be found from the very beginning of the classical computation, new momentum was gained with the technological progress, bio-inspired applications now covering all three axes that make up the POE framework [119].

The phylogenetic axis is represented mainly by software implementations based on genetic algorithms [35, 41]. Formally introduced in the United States in the 1970s by John Holland (University of Michigan), genetic algorithms are very attractive for some optimization problems, where classical algorithms may get stuck over local optima; they encode and combine the problem's possible solutions using genetic-like operators such as crossover (or recombination) and mutation in order to evolve a better result. Both these operators are non-deterministic by nature, and therefore fundamentally different than classic algorithms, thus offering a potential edge in digital computing. However, the evolution in the space of potential solutions pursues a predefined goal, the artificial population has no material existence, and individuals do not interact simultaneously, as opposed to natural systems [49]. Furthermore, the artificial phylogenetic implementations require a fitness calculation based on some characteristics of the pursued goal (which is obviously not an open-ended evolution), also very different than natural processes where evolution is open-ended (it does not appear to target a specific goal) [67].

A recent emerging field has been significantly influenced by the progress in reconfigurable hardware design and evolutionary computing. Called *evolvable hardware*, it is a special category that allows phylogenetic operations to be actually run inside computer hardware [116, 117, 158]. There is still an ongoing debate on how to define it properly; it may be regarded as a hardware solution to some evolutionary techniques but it may also be regarded as performing online bio-inspired adaptation processes. As an example of the latter, the Firefly machine is a hardware implementation of evolving cellular automata in order to solve the synchronization task, illustrating the phenomenon of a firefly population that reaches the state of pulsating synchronously [108].

According to the POE model (presented in Chapter 1, section 1.2.1), at a certain level, living matter continuously reorganizes itself based on learning experiences that take place by interacting with the surrounding environment. Therefore, along the epigenetic axis, one can find implementations involving artificial neural networks (ANNs) as attempts of building computing systems endowed with the capability of learning. Another category is expert systems, which try to emulate the processes that are linked to human intelligence. Of the three known epigenetic systems, namely the nervous, endocrine and immune systems, epigenetic artificial processes are usually inspired by the first [24]. There has been a vast scientific effort covering the research in this area, on both *learned* and *learning* systems, but there are still few hardware applications allowing true, online learning [89, 91]. The immune system is also providing bio-inspiration, as there are some implementations for error detection purposes [157] and even attempts towards the implementation of an artificial immune system [6, 40]; computer viruses were also found to integrate attacking strategies used by their biological correspondents [122].

Ontogenetic characteristics are specific to another organization level of the living matter, comprising all the developmental processes that take place in a multicellular organism. In contrast to the phylogenetic processes, ontogenetic processes are deterministic by nature, any occurring error therefore producing a severely handicapped or non-viable organism [155]. John von Neumann was the first to formalize the theory of self-reproducing automata capable of universal computation and universal construction [79]. Unfortunately, the complexity of such machines prohibited any physical implementation at that time; today, self-reproducing automata capable of universal computation have been implemented both in hardware [92, 111, 118] and software [19, 47, 129] but implementing the universal construction feature continues to elude modern science.

Ontogenetic processes include cellular division (starting from the moment of inception, with the mother cell or the zygote) and the specialization of daughter cells according to their position within the ensemble, known as cellular differentiation. Therefore, systems that implement ontogenetic processes in hardware need to be based on a hierarchic, modular (cellular) architecture, features made possible with the arrival of the reconfigurable devices. A representative example is the BioWatch [108, 124, 125], a time keeping machine that exhibits a hierarchical strategy of tolerating a range of possible faults by implementing both cellular division and cellular differentiation processes, and implemented recently over the artificial tissue of BioWall. Essentially, BioWall offers a bio-inspired platform that draws its powers from an array of 5700 Spartan FPGAs [134, 135]; apart from BioWatch, the BioWall demonstrates implementations of self-replicating loops, Turing neural networks, solving the synchronization task (the original FireFly machine was also ported on the BioWall platform), and string comparison computing.

Both the Firefly and the BioWatch machines are bio-inspired attempts to explore the potential of ontogenetic processes applied to computer hardware. While they each contribute to the establishing of evolvable hardware as a new field of research in computing, they are both offsprings of a much larger research project, focusing on the exploration of the POE model's implications in digital devices: it is known as the Embryonics project and this thesis is part of the collective research effort involved.

## 2.2. The Embryonics Project

Embryonics stands as a contraction for *embryonic electronics* and is the name of a long-term research project launched by the Logic Systems Laboratory at the Swiss Federal Institute of Technology, Lausanne [62]; it aims at establishing a bridge between the world of biology and the one of electronics, in particular the world of digital circuits. The main goal is to use biologically inspired mechanisms – borrowed from nature and adapted to electronics – and to draw inspiration from two distinct sources [66, 123, 128, 131]. The first is the biological mechanism of multi-cellular organization: the complex biological behavior is a result of massive parallel operation of a multitude of simple elements, the cells, each containing a complete description of the organism itself, which is the genome. The second is von Neumann's concept of self-replication of a universal computer, a mechanism allowing the automatic creation of multiple identical copies of a machine from a single instance [79, 128].

As a general research project, Embryonics is conceived not so much to achieve a specific goal, but to explore the uncovering insights by applying new concepts (inspired from biology) to a known field (digital computing). It tries to determine if interesting results can be obtained by applying biological concepts and mechanisms to the world of electronics [128] and to assess the potential of bio-inspired digital design in order to build innovative computing systems featuring superior robustness (i.e. fault tolerance in computer science) and beyond. As any original concept, Embryonics is also law protected by a number of patents [57, 58, 59].

### 2.2.1 Scope of Embryonics

The POE model presented in Chapter 1 provides a general framework for all bio-inspired implementations in digital computing; however, simultaneous exploration of all three research directions (phylogeny, ontogeny and epigenesis) would not prove to be an effective approach: Nature itself perfected the biological mechanisms over a very long time, and the process is still ongoing. Rather than attempting such a task, Embryonics offers the perspective of studying the feasibility of bio-inspiration in hardware design, along any individual direction of the POE model. In particular, the current focus of Embryonics is over the ontogenetic processes, as they may be regarded as constituents of the first level of organization of the living matter, offering a foundation for achieving robustness.

By demonstrating the technical possibility of modifying hardware by the use of information – and this is the case with the FPGAs – the feasibility of creating bio-inspired computer hardware was proven. Drawing inspiration from biological organisms has therefore led us to define the electronic organism as a two-dimensional array of processing elements, all identical in structure (similar to biological cells) and each executing a different part of the same program (the genome), depending on the position in the array (again, similar to biological cells) [25, 128].

### 2.2.2 Bio-Inspired Robustness

There are two essential biological mechanisms that can be considered as strong providers of reliability: multicellular organization and cellular division and specialization. Rather than following a centralized approach, if viewed from an "administrative" perspective, the organization in Embryonics is distributed over multiple entities that share the same basic structure: the cells. There are important advantages over computing processes also, since such an organization provides the platform for dynamically reconfigurable architectures, the processing power stemming from the distributed, massively parallel operations that are tailored for a specific application. Furthermore, the multicellular organization allows establishing an architectural hierarchy that will prove useful for robust computing and will also allow a sense of scale when analyzing such a system.

After thorough consideration of the aspects involved by creating an artificial ontogeny landscape, Embryonics adopted a quasi-biological, four-level architecture (see Figure 2-2) that allows a hierarchical separation between artificial entities that are different in complexity. The topmost level, similar to what is found in nature, is the *population* level. Essentially, the totality of organisms that "live" within a certain artificial environment (which is the entire computing system), make up what is called a population; individuals from a population (which may be viewed as computers operating in parallel) interact both with each other and with the environment itself.

One level lower in this hierarchy, the *organismic* level describes the internal structure of one organism. Each organism corresponds to a parallel computer system, member of a population (which is the higher level of organization), and may be decomposed into its more basic components. Its processing power is given by the massively parallel operation of its components, the cells; they are basic processing units and represent the *cellular* level. In nature, cells are the smallest biological entities that carry the entire genetic program of an organism, meaning they carry sufficient information to allow them to specialize in order to perform any required cellular activity; being basic processors, cells in Embryonics are similar to their biological correspondents, being able to perform any processing duty specified by their program.

The flexibility of biological cells comes from an even smaller scale perspective: they are built of molecules and possess the ability of fabricating new molecules according to their need. Therefore Embryonics' lowest level is the *molecular* level, represented by the simplest element that is part of the cellular structure; essentially, it is the element of a reconfigurable circuit capable of universal computation. Since this appears very much similar to the elementary unit of a fine-grained FPGA, Embryonics opens the way to designing a new type of fine-grained FPGA device that would feature bio-inspired mechanisms in order to deliver superior robustness.

Each of the four levels in Embryonics is shown in Figure 2-2. Going through the hierarchy, one may observe how the building bricks are assembled to form increasingly complex entities: the most basic parts, the molecules, are put together in order to make up the basic processor, or the cell. A single cell may also play the role of a unicellular organism, while a multicellular organism will be an ensemble containing more cells. Be it uni- or multi- cellular, such an artificial organism is a parallel computer inside which cells operate individually and cooperate with each other for a higher, organismic, purpose. At the top of the hierarchy lies the group of all organisms from within the artificial environment, the population.

### 2.2.3 Bio-Inspired Functionality

The reconfigurable nature of the Embryonics framework requires a piece of configuring software in order to define the functionality of the hardware, through a process also inspired from actual biological processes. In nature, living entities are driven by genetic information encoded in the DNA. Therefore, the Embryonics project also uses genetic information under the form of an artificial *genome*, which actually contains a low level encoding of the entire organism's hardware features (our genome is a binary configuration string analogous to the configuration bitstream of any FPGA).

In nature, the genome contains information regarding specifications of the entire organism; stored in each cell, it is organized in smaller fractions of the code called *genes*, which encode characteristics of a certain organ or tissue. In Embryonics cells perform independent tasks, despite all cells from an organism sharing the same hardware architecture and the same genome. The differences in cellular functionality arise from the fact that not all genes are executed by all cells; according to the specified task, a cell executes a set of genes through a process called *gene expression* (genes A to F are shown in Figure 2-2 at the organismic level).

The history of the Embryonics project did not see from the very beginning the hierarchy shown in Figure 2-2 as based on four different levels. It is worth mentioning how this hierarchical architecture was established, as this has a crucial influence over the functional aspect of the entire framework.

### 2.2.3.1 Artificial Organisms and Cellular Coordinates

Storing a copy of the genome in each cell might not seem very efficient; however, this has the appearance of a fair price for achieving some of the biological strengths. As a most interesting feature exhibited by biological cells, robustness is a consequence of the redundancy we find to be wasteful: each cell contains a copy of the entire genome and, theoretically, might replace any other cell by using a coordinate-based mechanism [151]. Since the gene expression uses the coordinate mechanism, a healthy cell can take over the functionality of a faulty one by changing gene execution through a process of updating its coordinates.

Theoretically, the design of an electronic organism could be implemented using any of the commercially available FPGAs [7, 140]; in practice however, this is difficult to achieve, since a processor, however simple, is a relatively complex circuit and restricting the amount of programmable logic to a single chip could rapidly raise limits when implementing the two-dimensional array of cells. An electronic organism therefore requires a completely homogenous, self-repairable FPGA, which could easily be configured as an array of identical elements. No commercial FPGA has these features, the conception and development of a new FPGA, capable of self-repair and self-replication being actually one of the main challenges of the Embryonics project [55, 63, 67, 128, 129, 131, 133].

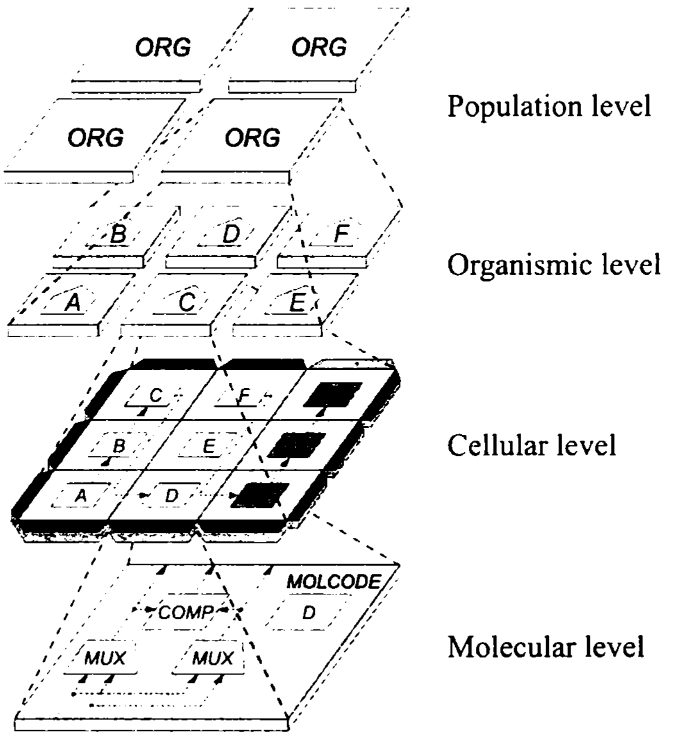


Figure 2-2: The four levels of organization in Embryonics [63].

Figure 2-3 presents an organism that suffers two errors, within two different cells. At the beginning (Figure 2-3, left), all cells operate normally, the organism consisting of active cells (marked as blue) and spare cells that would become active should any error be encountered (marked as yellow). The functional status of a cell at any particular moment is given by the genetic specifications contained by the genome, with respect to its geographical location given by its coordinates. The coordinate mechanism starts the numbering from the lower left (or the south-west) cell, which will be labeled as having the coordinates (1,1), the first being the horizontal and the last being the vertical coordinate. Each cell will therefore calculate its own coordinates (X, Y) based on the coordinates communicated by its neighbors to the west (with coordinates (WX, WY)) and to the south (with coordinates (SX, SY)), as follows:

$$X := WX + 1,\ Y := SY + 1.$$

At a certain moment, errors are detected in two different cells (having the coordinates (2,3) and (3,2)) and the self-repairing mechanism is triggered. Cellular universality plays an important role here: since the organism possesses the ability of recovering (there are spare cells available for repairing), the faulty cells are marked as "dead" thus triggering a re-evaluation of coordinates for each cell. The genome stored by each cell contains genetic information regarding *all* the cells from within an organism; each cell will express a certain set of genes, in accordance with a specific set of coordinates. When the new coordinates are computed due to the reconfiguration implied by the self-repairing process (the direction being from bottom to top and from left to right), the gene expression mechanism will force a new set of genes to be expressed by each cell positioned at the right of a faulty one; this appears as shifting an entire row of cells one position to the right, starting with the right neighbor of the faulty cell. The organism recovers by activating spare cells, as required by the self-repairing process (shown in Figure 2-3, right) [133].

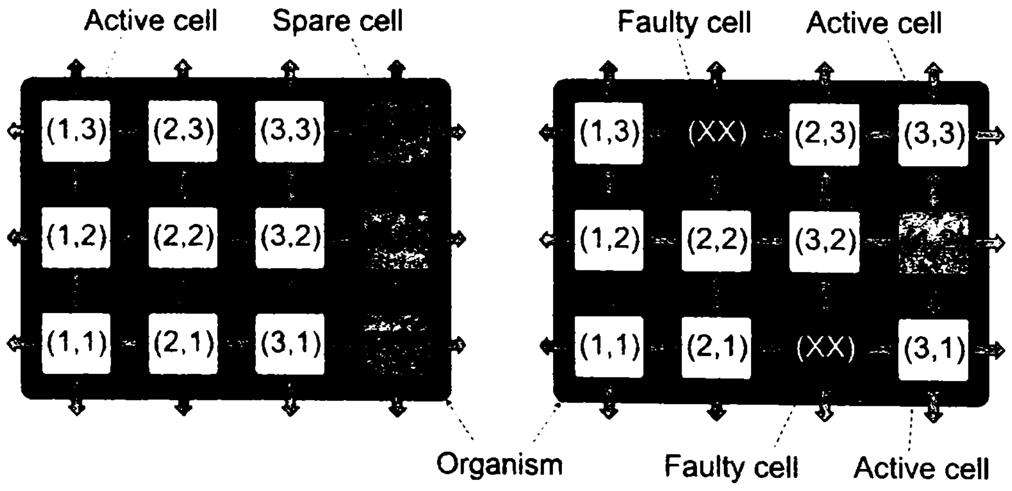


Figure 2-3: A small (4x3 cells) artificial organism with no faulty cells (left) and reconfiguration (self-repairing) after detection of a faulty cell (right) [128].

The redundancy introduced by having a copy of the genome in each cell provides an intrinsic support for self-repair. By providing a set of spare cells, i.e. cells that are inactive during normal operation, the organism is able to reconfigure its structure to avoid using faulty processors [56]. However, there are situations when the number of spare cells is outcome by the number of faulty ones; since faulty elements are too many, this will lead to the impossibility of the organism to successfully reconfigure itself, the result being the death of the entire organism.

The coordinate mechanism presented previously also proves to be useful when the colonization of an electronic environment with identical organisms (in nature, such a process is called *self-replication*) is required. Because the function of a cell depends on its coordinates, it is possible to implement the self-replication mechanism by simply cycling the coordinates [115]: when the configuration of an organism is complete, the process is repeated, the result being the configuration of new, identical organisms to the east and north of the original one. This way multiple copies of the same organism can be obtained, as shown in Figure 2-4.

#### 2.2.3.2 The Artificial Cell

Biological organisms provide a powerful example of systems exhibiting complex behavior; it seems this is a result of parallel interaction of many simple cells, rather of the complexity of each element. One of the goals of the Embryonics project is to show that biological inspiration allows us, without excessive difficulty, to design complex systems based on combining simple cells [63, 67, 70, 100, 101, 103, 132].

The previous subsection introduced the features of the electronic organism in Embryonics, which have consequences over the lower level of organization, represented by the cells. The genome program must be stored in each of the cells and all must feature a coordinate-dependent access mechanism. The minimal feature set for our electronic cells must therefore include [128]:

- an [X, Y] coordinate system to allow the cell to locate its position inside the organismic array and thus its function;
- a memory to store the genome;
- an interpreter to read and execute the genome; in digital computing this is equivalent to specifying a language according to which the genetic program will be encoded;
- a functional unit, for data processing purposes; this may contain a variety of logic elements, from a single register to a full arithmetic and logic unit (ALU) and beyond, depending on the application;
- a set of interconnections handled by a routing unit that will allow cells to communicate with each other.

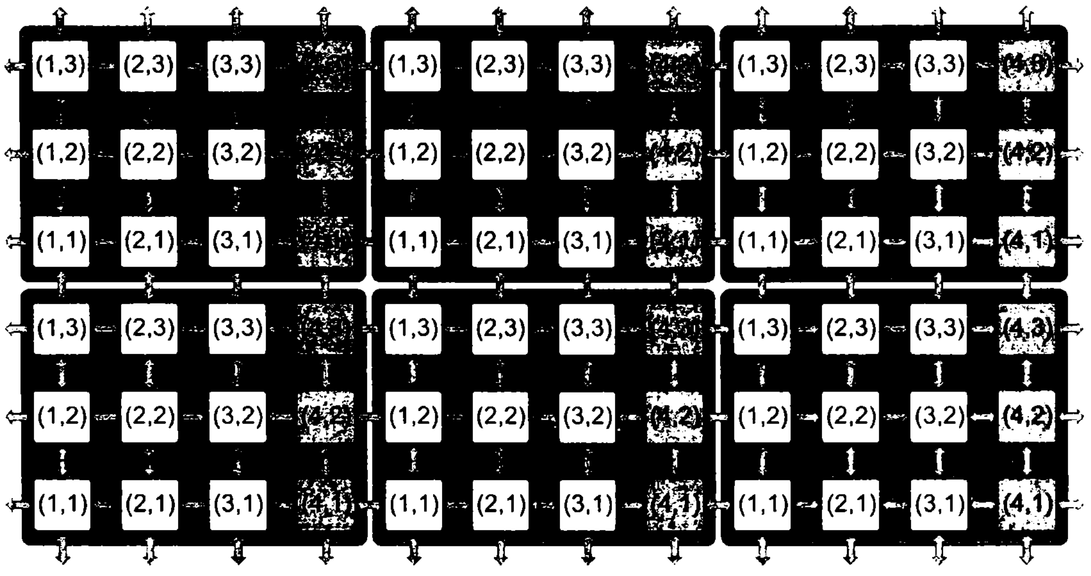
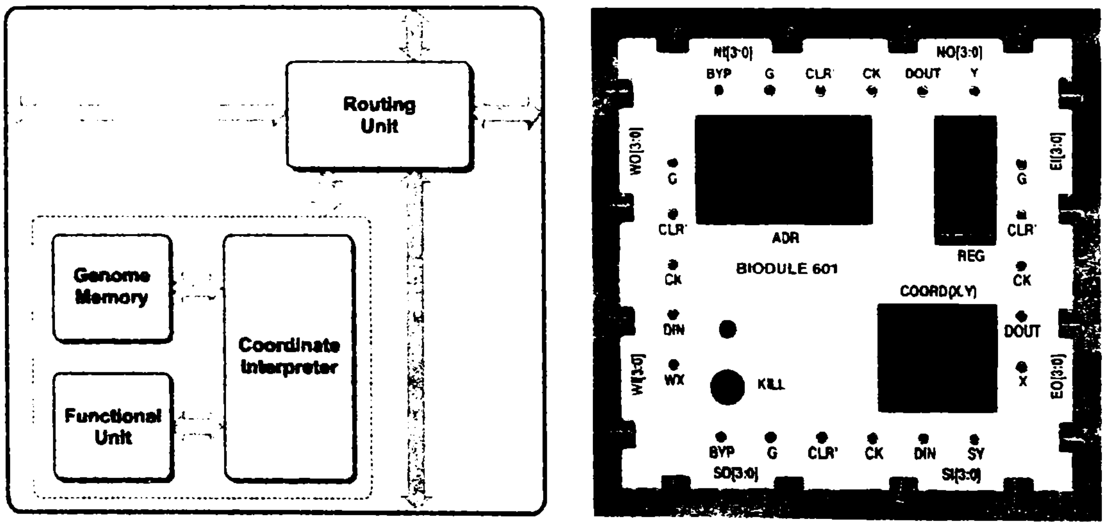


Figure 2-4: Multiple copies of the same organism through coordinate cycling [128].

Furthermore, each cell requires some means of separation from other cells, similar to the cellular membrane encountered in biology. Therefore one may find that the genetic program has to deal with a double aspect: it has to specify and configure the functional resources inside the cell, but at the same time it also has to store the software to be executed by functional units, either of its own or pertaining to other cells. Therefore, several different pieces of information need to be stored inside the genome:

- the *polymerase genome* is the part of the genetic information that establishes a boundary between cells, thus encoding the electronic correspondent of the biological cellular membrane;
- the *ribosomic genome* is the genome portion that specifies the configuration of the electronic ribosomes represented by the functional resources; assimilated in an FPGA-type environment as programmable logic blocks (which assemble cabled-logic machines), the ribosomes have the role of executing pieces of specific software, which is also encoded by the genome;
- finally, the software driving the ribosomes from a single cell, or even different cells is called the *operative genome*, and may be regarded as being actually the software that drives microprogrammed machines.

The architecture of an artificial cell that would encompass all required features is shown in Figure 2-5A. Designed to operate as an element from a two-dimensional array of any finite size, the first implementation of a cell in Embryonics was called the MicTree cell [16, 62, 66]. At the time, the cellular level (see Figure 2-2) was the bottom of the Embryonics hierarchy, the cell being realized using an FPGA mounted on a custom printed circuit board together with a set of 7-segment displays and LEDs, which was then inserted in a plastic box, called Biodule 601 [16], shown in Figure 2-5B.



A. B.

Figure 2-5: The artificial cell. A: the internal architecture and B: physical implementation of the MicTree cell. [Photo by André Badertscher]

The biodules are capable to be fitted together like a puzzle, forming a two-dimensional array in which direct neighbor-to-neighbor connections are provided [128] without the need for extra cables. The electronic organism consists therefore of a two-dimensional array of MicTree cells. The fixed architecture of the MicTree cell, while perfectly capable of demonstrating the capabilities of the Embryonics hierarchical architecture of a bio-inspired computing system, presents, however, several constraints. One such limitation is due to the fixed size (four bits) of each of the coordinate registers, the consequence being an organism limited in dimension to a maximum of 16x16 cells [55, 62, 66]; the functional unit consists of a register, also four bits wide. While this is quite sufficient for the purpose of building a fully functional demonstrator system, it is a serious limitation for the range of potential applications.

The memory of the Biodule 601 has a capacity of 1024 words, each being 8 bits wide. Applications implemented with MicTree cells include the Biowatch machine (which uses four MicTree cells), a cellular automaton that generates random numbers (which uses five MicTree cells), and a parenthesis-checking Turing machine (which uses ten MicTree cells) [67].

While the MicTree cell's architecture is simple and flexible, allowing the implementation of a range of applications (given that a sufficient number of such cells is available), it generates a few issues that are worth considering:

- the MicTree cell is actually a microprogrammed machine with a fixed functional unit and with programmable connections; this comes in contradiction with the fact that several different cellular architectures can be found in nature. Moreover, with a four bit-wide register, the functional unit may be considered as too weak for many applications, thus forcing a shift in complexity towards the genetic program;
- the physically-fixed cellular dimensions make the existence of the polymerase genome obsolete;
- the memory available for genome storage has a fixed capacity that will eventually limit the implementation capabilities of such a system;
- though a cellular architecture does present some advantages over traditional systems, considering the wide range of possible applications, it would be more adequate to be able to tailor the cells to the specific patterns required.

All these issues may appear as independent if seen purely from an engineering standpoint; but since they are closely connected in the biological world, addressing them may be done in a bio-inspired fashion,. Nature has perfected a solution that provides a flexible cellular architecture: each cell is built by molecular structures, determined and synthesized with the support of complex chemical processes. While mastering such processes in electronics is not within the grasp of modern science, another level of organization that would become the bottom of the Embryonics hierarchy and placed below the cellular level can be implemented: this is the molecular level.

#### 2.2.3.3 The Artificial Molecule

As previously mentioned, the cellular level provides an intrinsic capability for self-repair through its coordinate system. While in biological organisms a cell's death is a common thing to happen (it will be replaced by another, freshly grown, one), cost effectiveness in digital engineering makes only a limited number of expendable cells affordable. For instance, if we assume that a cell consists of a few hundred molecules, the resource penalty inflicted by its "death" obviously requires minimizing the occurrence of such situations.

A possible solution is to also endow cells with a certain degree of fault tolerance by performing a transfer of the self-repairing concepts from the cellular level (described in subsection 2.2.3.1 and shown in Figure 2-3) to the molecular level. Such a decision will have important effects over the molecular level, appearing somewhat as a scaled down version of the cellular level:

- the molecules themselves will be universal, meaning that any two molecules can be configured so as to exchange or take over roles from each other;

- the concept of spares will also be present at the molecular level, allowing any cell to tolerate a number of occurring faults by internal (molecular) reconfiguration using spare molecules.

Furthermore, because cells will be composed of molecules, all requirements mentioned in the previous subsection will have to be scaled down and accommodated at the molecular level. As a direct consequence, the architecture of a molecule will contain the following:

- a simple, fine-grained, functional unit, capable of universal computation. It has to be fully reconfigurable, that is, both the functionality and the inputs/outputs are programmable;
- a routing unit managing the communications between molecules, from either direct neighbors or from separated molecules;
- a genome memory that will accommodate the configuration for both the functional unit and the routing unit;
- finally, due to the fact that cells don't have a fixed size anymore, the polymerase genome (which encodes the boundaries of the cell, similar to the cellular membrane in biology) has to be also accommodated by each molecule.

One should remark that any molecule with such an architecture will store only parts of the cellular polymerase genome (each molecule has to know its position with respect to a certain cell, whether its location is inside the cell or at one of its borders), of the ribosomic genome (the functional and the routing units are ribosomic resources for the cell), and of the operative genome (the remaining storage space from the genome memory, containing other sequences of microcode); the entire cellular genome is therefore the collection of individual molecular parts.

The implementation of an electronic molecule that meets all of the above requirements was called MuxTree and was realized by using the same principle of inserting the actual hardware inside a plastic box (shown in Figure 2-6), allowing the creation of puzzle-like structures used previously for the MicTree cells [133].

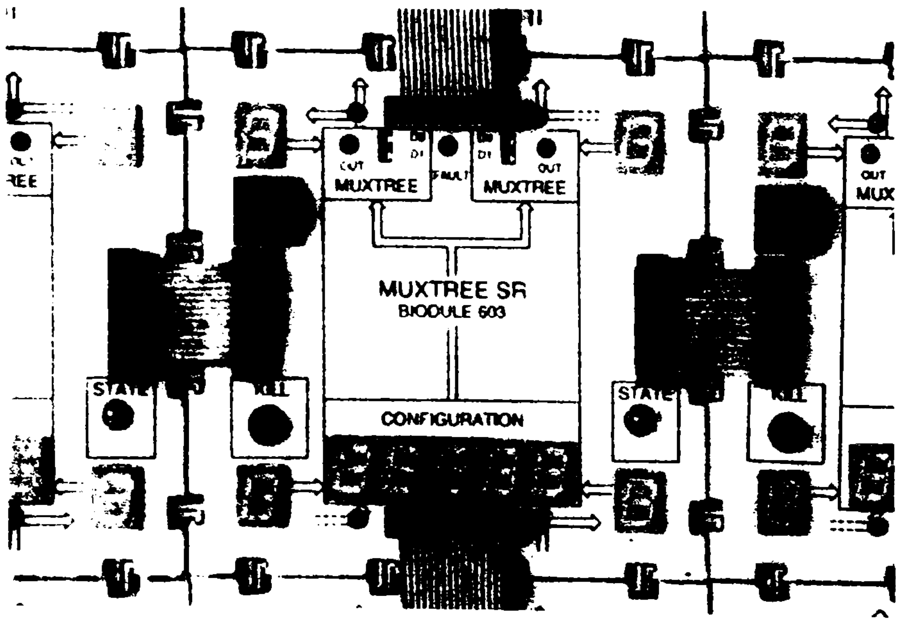


Figure 2-6: The implementation of the MuxTree molecule. [Photo by André Badertscher]

The conception and early development of the MuxTree molecule was the subject of a Ph.D. thesis [128]; it was designed to implement the basic element of a multiplexer-based, bio-inspired FPGA and realized by using a commercially available chip of the Xilinx 4000 series [67]. Like all FPGAs, MuxTree units also provide a puzzle-like assembly as a two-dimensional array of elements.

The architecture of MuxTree includes all necessary resources mentioned above, the block schematic being shown in Figure 2-7. The *functional unit* (FU) of the MuxTree molecule offers both combinational and sequential resources under the form of two-input multiplexer – hence the name – and a D-type flip-flop; any function can be implemented, regardless the complexity, by combining a sufficient number of MuxTree molecules.

The *switching block* (SB) is responsible of the programmable connectivity between molecules (see Figure 2.8). There are two categories of interconnections that can be used: those intended for transporting information between non-neighboring molecules (similar to long distance buses in commercial FPGAs, for instance FastLANE™ in the Xilinx 6000 series), which are programmable, and those that manage communication between neighboring molecules (similar to short distance buses or direct connections), which are not programmable. The long distance buses cover all four directions (both input and output) and information is routed according to the information stored by the *configuration register* (CREG), each direction being able of accepting information flow by using the multiplexers, either from any other direction or as constant, preset signals (shown as a rectangle input for each multiplexer in Figure 2-8).

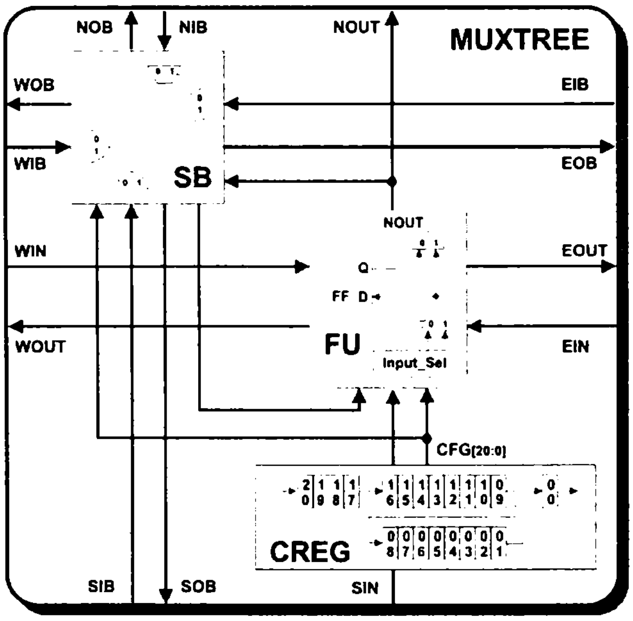


Figure 2-7: Block schematic of the MuxTree molecule.

The configuration register (CREG) is a special purpose register used to store the information required to drive the programmable functional unit (FU) and the switching block (SB). By consequence, CREG stores a binary string called *the molcode*, which accommodates the molecule's ribosomic and operative genomes. While the ribosomic genome drives the molecular ribosomic parts (which are the FU and the SB), the operative genome is indirectly determined by the molcode: it has to be stored in one bit-wide pieces of information by using the only available memory unit, the flip-flop. Implementing any micro-programmed machine will therefore face the difficult problem of microprogram storage, which will have to be stored bit by bit by MuxTree molecules. This is an essential shortcoming since it is both terribly ineffective (involving a bit-by-bit storage of a program) and resource wasting (the rest of FU and the SB being difficult to use to their full potential inside a structure meant to store pieces of microcode), the Embryonics project had to develop other solutions.

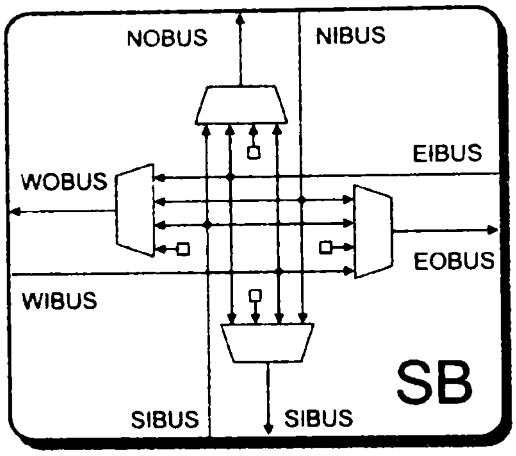


Figure 2-8: Block schematic of the Switch Block.

## 2.3. Objectives

As part of a long-term research project such as Embryonics, this thesis aims at establishing advancing steps that will preserve some of the already existing concepts, introduce new ones and/or modify existing ones, in order to bring the architecture of ontogenetic machines to a more mature and stable state. The previous research efforts materialized in a Ph.D. thesis [128] with the conclusion that the current architecture of MuxTree was to be improved with respect to its memory capabilities, both because of engineering reasons (providing an efficient storage for the genetic micro-program) and of bio-inspired reasons (the biological genome, which is quite large, is stored by a dedicated memory structure, which is the DNA). While the Embryonics project spans over at least the phylogenetic (a first hardware implementation, the FireFly machine, though technically not being part of Embryonics, allowed the exploration of on-line evolutionary processes) and ontogenetic directions (the BioWatch being a demonstrator platform built of MicTree cells and MuxTree molecules) [26, 108], the goal of this thesis is to advance the research along the ontogenetic direction (see Figure 2-9).

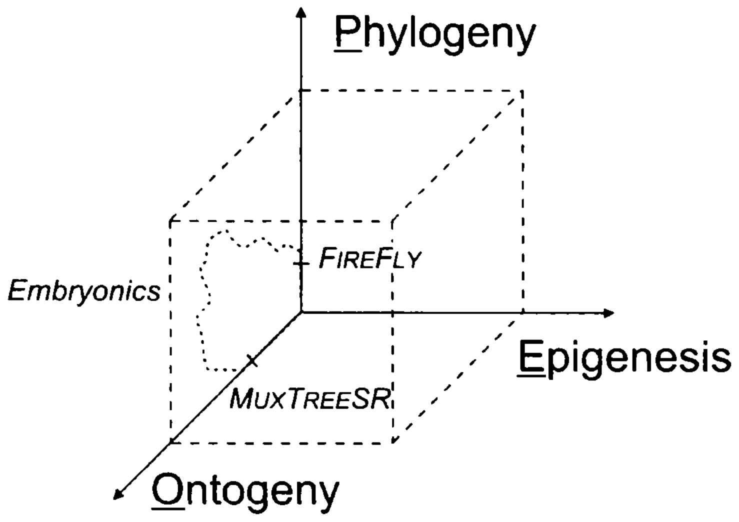


Figure 2-9: The Embryonics project in the POE landscape.

Therefore, the main objective is to extend the architecture of MuxTree by transforming the configuration register CREG into a flexible molecular resource that would also provide storage space for the operative genome. Furthermore, an effective design methodology would preserve the concepts and mechanisms already proven in the Embryonics framework but also bring new flexibility by adopting bio-inspired means for building memory structures. Last, but not least, due to its novelty character, the introduction of a new concept is usually followed by unforeseen problems regarding its integration within the platform, an aspect that also has to be taken into account and settled successfully.

Advancing the Embryonics design towards a more mature state will be done according to the requirements of today's space exploration era; this we believe to be its most valuable potential application, where extremely long term operations without the possibility of human maintenance require superior levels of fault tolerance as a top priority.

# CHAPTER 3

# A BIO-INSPIRED MEMORY ARCHITECTURE

## 3.1. Introduction

As argued in Chapter 2, MuxTree is actually a new concept in designing digital circuits [57, 58, 59]; although its conception process was inspired from the generous spring of bio-inspiration, its birth was not struggle-free, as the internal architecture would have to be capable of performing real-world tasks more reliably than current, classic machines, while keeping overall costs at a reasonable level. There is at least one issue that may be regarded as a disadvantage: although any digital hardware can be built by using MuxTree molecules only, implementing large memory areas feels quite ineffective. This happens because the only internal resource of the MuxTree molecule available as a storage unit is a *single* flip-flop (from the functional unit FU), while a wider storage capability is already present (represented by the configuration register CREG) but is unavailable to the user (see Figure 2-7). With a memory capacity of only one bit per molecule, building a memory structure even for a simple micro-programmed machine is prohibitive; furthermore, the intrinsic characteristics of a storage area will likely lead these molecules to not making a full use of their remaining (logic) resources.

Under these considerations, although a current implementation already existed [72, 131], a more refined architecture was perceived to be feasible, particularly when storage features were concerned, while preserving the already proven mechanisms unaltered. The memory issue is vital, since it does not imply only dozens of bits of genetic information; a typical dimension of an entire genetic program is of hundreds or even thousands of words. Therefore we came to look forward to exploring new possibilities for MuxTree, in order to enhance its design while keeping all of its strengths.

Making the MuxTree as versatile as possible appears as a matter of scale: at the cellular level, the MicTree cell (see Chapter 2, Figure 2-5) contains a dedicated memory for genome storage, which will have to be assembled in an efficient manner by smaller units – the MuxTree molecules – when going to the molecular level. Our approach towards enhancing the original design, called MuxTreeSR is to provide bio-inspired storage that would also be as efficient as possible [100, 101, 102, 103]; this was called RAM-MuxTreeSR [100].

An example of a memory structure in Embryonics is presented in Figure 3-1, where an electronic cell contains 4 blocks of separate memory areas, implemented with RAM-MuxTreeSR molecules. The dotted boxes denote the memory molecules grouped together as rectangular memory structures while the light color indicates the non-memory molecules, which will actually process the information stored inside the memory. The dotted rectangles delimit standalone memory areas, each featuring outputs ports (shown as *NOUT* in Figure 3-4) that provide stored genetic information every clock cycle. These memory areas are delimited from each other and from non-memory units (denoted as dotted lines) in a similar way done by the *cellular membrane* (actually by a mechanism of growing a special border, which encloses the area [47, 123, 128]; this membrane determines the dimensions of a cell); because this partitioning does not take place at the cellular level, involving molecules instead, we decided to call them *macro-molecules*.

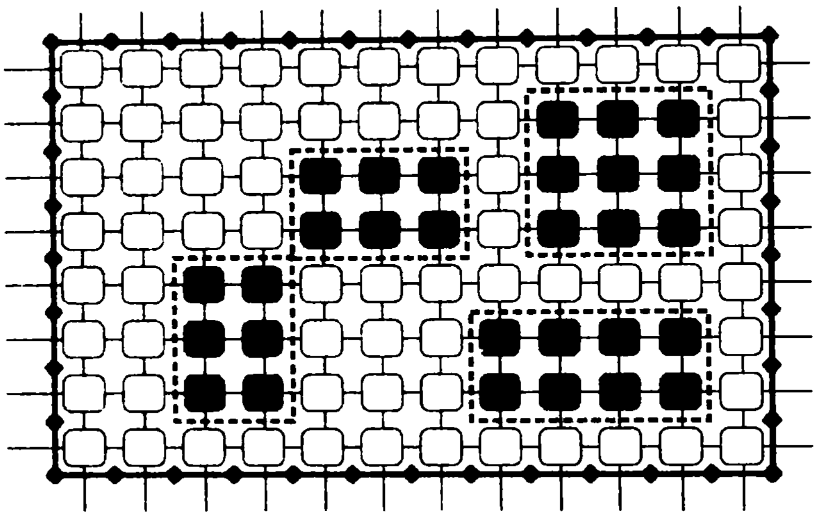


Figure 3-1: General structure of RAM-MuxTreeSR memory structures.

In order to implement such structures, there are several essential issues that have to be settled before the beginning of any design process: addressing a memory block, delimiting memory blocks from each other (in a similar way the cellular membrane does), and preserving/enhancing existent mechanisms in the new context.

## 3.2. Memory-Related Issues: Cyclic VS Addressable

Implementing a memory area might not seem to involve different issues than the case of implementing a random access type memory (RAM) or even a read-only memory (ROM). However, rather than building a random access memory out of MuxTree molecules, the purpose is to have the genome memory implemented in a bio-inspired manner that would be both suitable to the MuxTree architecture and efficient from a computing perspective.

Since the prototype MuxTree element was not specifically designed for data storage from the beginning, it did not have special features covering this issue; instead, the focus was on achieving the best connectivity and robustness. The only possibility for data storage was offered by the flip-flop in the functional unit, a solution that unfortunately required a great number of MuxTree elements for implementing even a small memory area. This is the main reason why making the best possible use of the configuration register was particularly appealing. However, the feasibility of such approach, together with the immediate consequences over the Embryonics framework required careful consideration.

The architecture of a conventional RAM memory is well known. However, the goal was not necessarily implementing its exact structure with MuxTree elements, although it is technically possible. Considering the storage capability of the MuxTree being present under the form of its only flip-flop, it would seem quite a waste to implement a RAM memory; this task would require a great number of MuxTree elements used as a storage capacity of one single bit only, while rendering the rest of the internal logic largely unusable.

On one hand, this approach would provide us a great flexibility. Since the memory will be used to store the genetic program, a RAM-type implementation – i.e. an addressable type of memory – would allow jump-like instructions to be present in the genetic program. However, the amount of additional logic needed inside the MuxTree for efficiently implementing a RAM-like memory proved far too large to be taken into consideration.

On the other hand, another option presents itself under the form of a cyclic type of memory [100, 102], a non-standard memory structure that allows continuous and sequential access to its data. This means that, although functional within the Embryonics framework, this type of memory will not be addressable – i.e. it will not allow the presence of jump-like instructions in the genetic program – the advantage being the amount of logic necessary to implement the memory in this way would be significantly reduced. The memory will continuously shift its data and the access to a given word at a specific address will not be possible; however, this is not a requirement since the genetic program will be executed continuously, similar to the process that takes place in every living being. The minimum amount of logic necessary to carry out the implementation on this path is worth in terms of sacrificing the power of a RAM memory, a synthetic comparison between the two types of memory being shown in Figure 3-2.

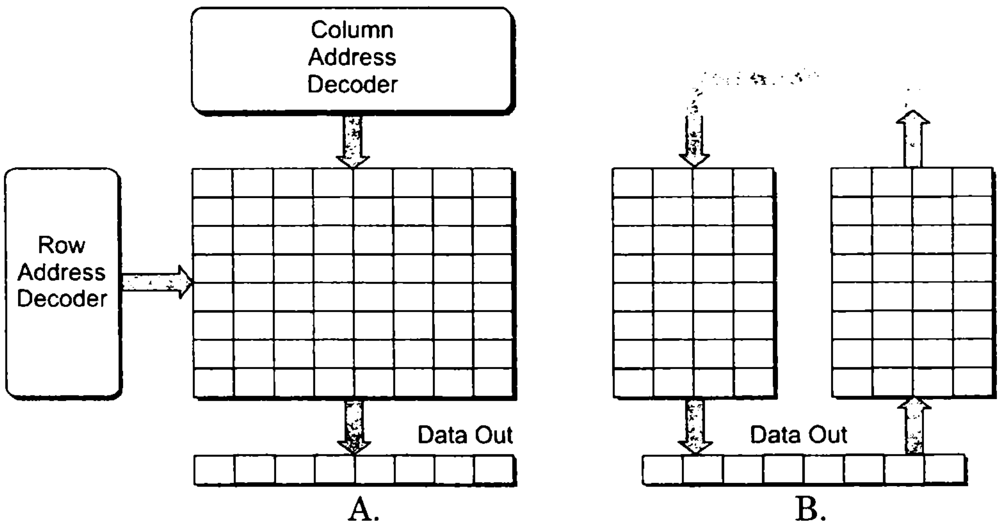


Figure 3-2: Addressable memory (A) VS cyclic memory (B).

There are several ways one can design a cyclic-type memory. Rather than storing words into one large memory, our approach was to *distribute* a word worth of genetic information into several, smaller memory areas.

The memory we want to implement does not have the same constraints as typical RAM memories, such as random memory cell addressing, so it will not need the complicated addressing mechanism, the address decoders and all the required additional logic. Its purpose is to store the genetic program, which is not supposed to change during the normal operation of the system, with the imposed constraint being the memory should be able to shift its data, also allowing an output point for the data stream, i.e. the data must be accessible through a particular port. Since the MuxTree features ensure that information propagate initially throughout the configuration process and then later, throughout the busses driven by the switching block (see Figure 2-8), it would be useful to also re-use as many existing connection patterns as possible.

## 3.3 Memory Organization

### 3.3.1 The Global Memory

The memory designed for the Embryonics project has neither the features of a conventional random access memory, nor the actual structure of a RAM; instead, it is actually more similar to a read-only memory (ROM), providing access ports to the stored data through a cyclic mechanism [100, 102]. For instance, storing a memory word that is 4-bit wide, the memory array will be an abstract entity, consisting out of 4 basic memory areas. This is where the memory area actually differs in structure when compared with conventional memories. From now on, we will refer to the overall memory structure as to the *global memory* and we will use for the actual storing region the term of *basic memory* (i.e. macro-molecule) area, since this is the fundamental storage area in our organism. These terms are necessary since the global memory might consist of several basic memory areas that are actually distributed inside the cell. An example of the global memory inside the electronic organism is shown in Figure 3-3 and consists of memory molecules (dark colored) grouped in four macro-molecules or basic memory areas; each dotted rectangle delimits a macro-molecule, meaning each word of the genetic program is 4-bit wide.

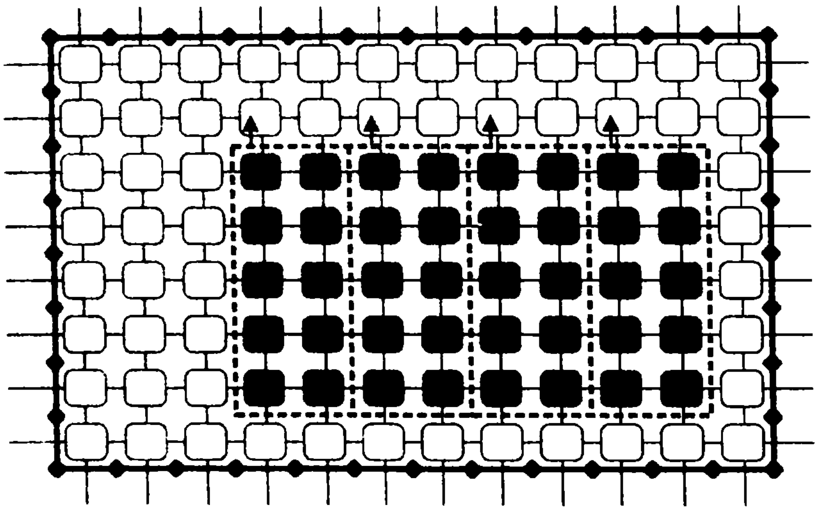


Figure 3-3: A global memory of 4 macro-molecules inside a cell.

Because the basic memory area continuously shifts its stored data, the output data being available at the topmost elements of each basic memory area, from 4 similar data outputs coming from the 4 basic memory areas we will get a 4-bit wide information. This is another characteristic that differentiates the functionality of the global memory area from conventional memories. For the reasons presented above, the organization of the information inside a macro-molecule follows a vertical direction rather than the more usual horizontal one. The memory areas shown in figure 3-3 have the same number of molecules because they are supposed, in this example, to store words of the same genetic program. Otherwise, the number of molecules may differ.

The way the memory is organized in the Embryonics project resembles the biological way: instead of having a memory storing all the needed information, our cells may contain several macro-molecules; the molecules, in turn, concurrently do the same activity: they allow data access by continuously shifting the information. This presents a strongly resemblance with the biological parallelism that is present in any given cell.

A rather poetic sight, using a top-down point of view for our electronic organism, would be that our memory architecture behaves like the DNA: the information is coded – inside DNA by chemical components, inside our memory by binary code – and the DNA has the structure of a chain while our memory shows the same similarity through the manner memory molecules connect to each other. The DNA (our memory) provides the genetic information – the genome (or, in our case, the genetic program) – and thus ensures the living process (the functionality) of the cell, be it biological or electronic [101]. Part of this process is shown in Figure 3-4, where the information flow from one MuxTree memory molecule to another follows a circle-like path, starting with the bottom-left molecule, going upside to the top and then repeating the process until the right-most column. This is where the information is routed so as to reach again to the bottom-left molecule.

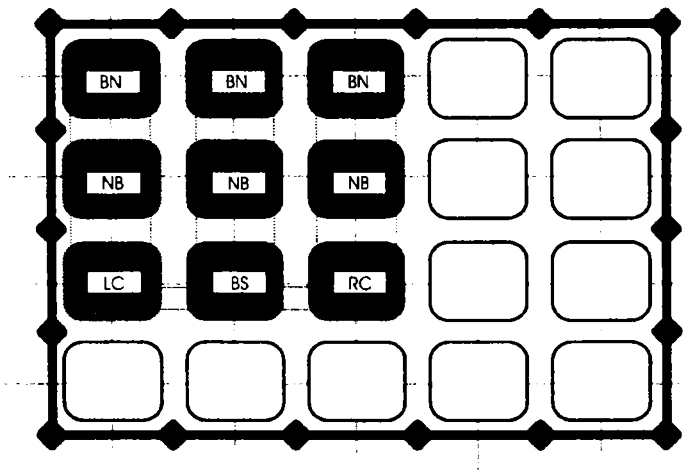


Figure 3-4: Information path within the basic memory area.

## 3.3.2 The Basic Memory (Macro-Molecule)

From a certain point of view, a macro-molecule is actually very similar in structure to the cell itself; it is a rectangular area inside a cell, which is also a rectangular area. Because of the similarity between them, the ground for re-using the connections is already established and such is the case of a normal, non-memory region; all what is needed is a mechanism similar, in a way, to the cellular automaton building the cellular membrane.

The information stored in this memory has to be continuously shifted. This ensures that at every clock cycle the memory array assembles with its data outputs one complete memory word. The words stored inside the memory are therefore output at the data-out connections and can be re-routed by using the normal, non-memory molecules.

## 3.4 Memory Architecture

### 3.4.1 Introduction

Following a long established tradition of implementing the designs as actual hardware prototypes rather than limiting ourselves only to software simulations, we will go into more details concerning the prototype development and the conceptual issues involved by the addition of memory structures. First, the memory architecture will be introduced, together with the necessary modifications to achieve such structures. Extending the mechanisms for self-repair and self-replication so that they work with memory structures will then be presented.

The original MuxTreeSR design was enhanced to offer data storage capabilities. Whereas the original design used the configuration register for storing the configuration bits (which are to be loaded at the beginning and then remain fixed for the entire operation time), the new design is further advanced by allowing parts of the configuration register to behave like memory units under certain conditions. The previous design's impressive features such as self-repair and self-replication were preserved, keeping the functionality while enhancing the MuxTreeSR element with new, necessary features. Details of the register's functions will be given but legacy features from the previous design (such as the cellular automaton used for replication, the logic required for re-routing the array in the presence of faulty elements or some parts inside the core of MuxTreeSR element such as the functional part or the switch block) will not be discussed as they were the subject of another thesis [128]; instead, we will focus on modifications made to the configuration register, connected with modifications operated to the functional part and the re-routing logic, to achieve the memory capabilities.

All parts from the original design that were suited to be preserved were kept as close as possible to their initial implementation. While conserving the initial functionality of the design warrants the performances obtained initially, it also helps at keeping the costs affordable for a practical implementation.

### 3.4.2 Short Overview

An efficient way of implementing digital machines is provided by *logidules* (for logic modules), which are plastic cubes containing standard logic circuits (gates, flip-flops, etc) [4]. The logidules are extremely versatile, they can be connected together (not unlike a puzzle) and automatically provide the circuits' power supply as well as the minimal connections between modules.

It is therefore not surprising that we exploited the same approach when designing the prototype, by taking advantage of modules called Biodule 603 [16]. To implement our designs we used a Xilinx XC4013PQ240-4 FPGA with a locally stored configuration. This allows us the opportunity of upgrading our design at a later date with no need of physically modifying the architecture of the Biodules.

Like all logidules, the Biodules 603 can be joined together to form a two-dimensional array with the connections implemented either through the automatic contacts on the perimeter of the box, or through four 16-bit-wide connectors.

As already mentioned, the RAM-MuxTreeSR molecule has several modes of operation [55, 67, 133]. The operating mode is set when the configuration patterns are loaded into the FPGA structure. The elements register, previously called *the configuration register* because it stored the configuration for the corresponding element, was slightly expanded, the elements' function and connections being now determined by a 21-bit configuration as follows:

1. Logic mode (non-memory)
   - i. Bit 0 (the head of the register) is always set to the logical value "1". This indicates that the current element was loaded with the configuration pattern;
   - ii. Bits 1 to 8 contain the control variables for the four multiplexers in the switch block SB;
   - iii. Bits 9 to 11 select the left input of multiplexer M0;
   - iv. Bit 12 is unused;
   - v. Bits 13 to 15 select the right input of multiplexer M0;
   - vi. Bit 16 is unused;
   - vii. Bit 17 (M) is the control variable for multiplexer M1;
   - viii. Bit 18 (S) is the control variable for multiplexer M2;
   - ix. Bit 19 (I) contains the default value for the flip-flop F;
   - x. Bit 20 (Mem) indicates the operating mode ("0" in non-memory mode);
2. Memory mode
   - i. Bit 0 (the head of the register) is always set to the logical value "1". This indicates that the current element was loaded with the configuration pattern;
   - ii. Bits 1 to 8 either contain the control variables for the four multiplexers in the switch block SB (when operating in 8-bit memory mode) or are used as storage bits for memory data (when operating in 16-bit memory mode); more details are given in Section 3-5;
   - iii. Bits 9 to 16 are used as storage bits in both memory operating modes;
   - iv. Bits 17 to 19 code the role of the corresponding element as indicated in Table 3-1 below;
   - v. Bit 20 (Mem) indicates the operating mode ("1" in both memory operating modes).

The configuration register CREG was modified to allow data flowing when in any of the several operating modes [100]. Its block schematic is presented in Figure 3-7 and it is split into 4 parts, each of them functioning in a different manner depending upon the initial configuration.

### 3.4.3 The Memory Structure

The general guidelines for a new memory structure being set, the new design of the RAM-MuxTreeSR allows the implementation under the form of rectangular structures. These rectangular areas have no dimensional limitations (they can be as large as the physical implementation allows), but there is however a constraint when we consider the smallest memory area possible: it must have at least two molecules (suggested by Figure 3-13C). The two molecules (as described by Table 3-1) are coded as "Bottom of a Single Column" (BC) and "Border to the North with Data Output" (BN). These are required in order to ensure the circular data flow for the memory. Figure 3-4 also presents the basic signals used for routing the stored bit-stream inside a 3x3 memory structure together. The configuration of the MuxTree elements is indicated by acronyms that will be explained later by Table 3-2.

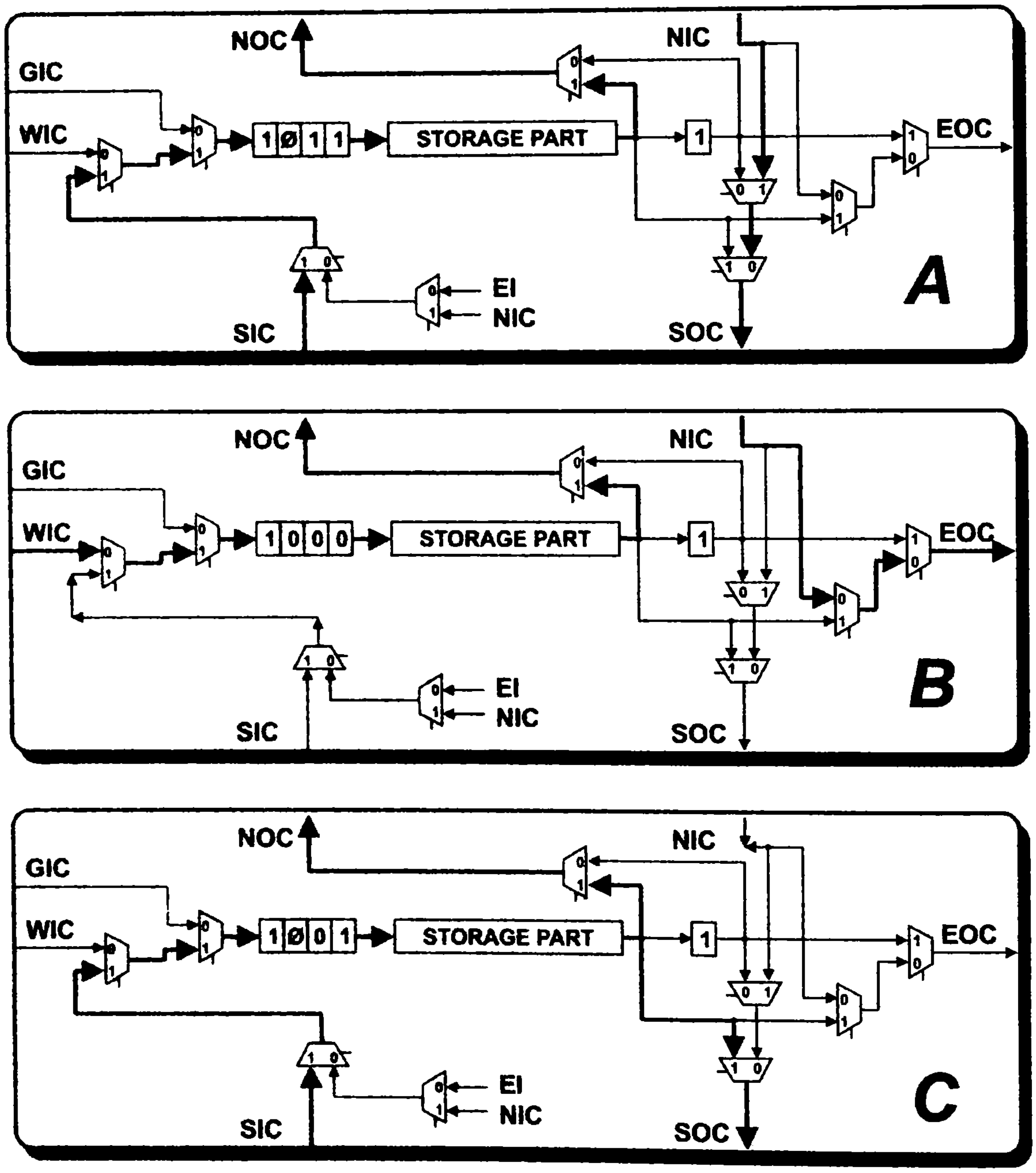


Figure 3-5: Configuration register as storage resource (Ø means "don't care"): A. No Border; B. Border to the south; C. Border to the north with data output.

The existing connections, which were originally intended for directing the initial configuration bit stream, are being re-used for the most of the structure, the exception being the bottom row where the so-called "short connections" were used to enclose the circle [100, 128]. Therefore, to route the information from the bottom-right element to the bottom-left one, the bit-stream flows through the bottom row following the path established by east input – west output local connections.

Since we need to specify which molecule has to be configured as bottom left and right corners (different internal connection patterns being involved, as shown in Figure 3-6), the minimal horizontal dimension of a macro-molecule is 2 (with the exception of the memory column), the presence of molecules configured as "border south" not being necessary. Molecules configured as "no border" do not have a special behavior; they just let the information entering through the south be shifted to the north connections. The top molecules re-route moving information to the southern output connection.

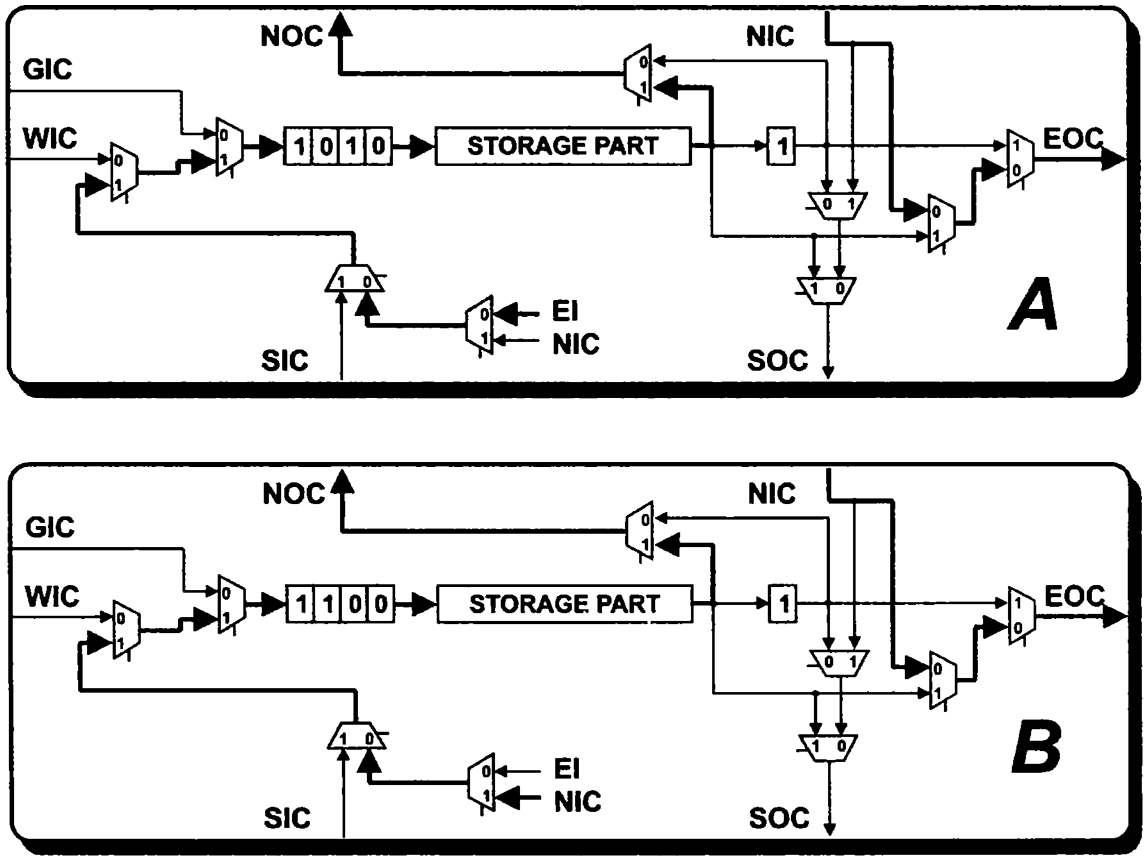


Figure 3-6: Corners of a memory structure: A. Left corner; B. Right corner.

## 3.4.4 Data Routing

The configuration register, together with the additional logic that controls the data routing process, performs differently according to different configuration patterns. Figure 3-5 provides a closer look at the configurations necessary in order to set up a memory area. They are shown together with the routing logic so the data path can easily be followed. In general, we distinguish three cases:

1. The molecule has no border behavior (A). In this case, information that enters through the south input connection is shifted through the register and exits through the north output connection. The secondary data path is present here, leading information coming from the north connection to the south output connection.

2. The molecule is at the southern border (B). In this case, information enters through the west input connection, it is shifted through the register and exits through the north output connection. The secondary data path is present here, leading information coming from the north connection to the east output

connection. The cases when the molecule is a corner in our memory structure are discussed below.

3. The molecule is at the northern border (C). In this case, information that enters through the south input connection is shifted through the register and then exits through the south output connection. In this case the molecule also provides a data output port, data being available to the north output connection.

Figure 3-6A presents the information flow inside the bottom left corner molecule. In this case, information enters through the east input connection, it is shifted through the register, and exits through the north output connection. The secondary data path is also present here, and it is identical to that in the case of border to the south. When the molecule is a right corner, data is taken over from the north input connection rather from the east input connection, as shown in figure 3-6B.

The whole memory structure shown in Figure 3-4 should now be almost complete; it can be seen why the bottom row, corresponding to the southern border, has to have different configurations for the left corner, the right corner and for the regular southern border molecules. The memory configuration bits, as they are defined now, only cover 6 possibilities out of possible 8, so there are two spare positions that could be later exploited. For instance, the possibility of having memory columns, i.e. memory areas consisting of only one single column, could be achieved with a minimum of additional logic (this modification only requires a few more logic gates) [100].

### 3.4.5 Operating Modes for the Configuration Register (CREG)

The previous design used the entire register to store the configuration [128, 131]. The amount of logic used for routing the data during configuration was smaller, as shown in Figure 3-7A. The modifications that were made in this project enabled the more flexible use of the configuration register [100]. With the price of adding one bit in the register and a small amount of additional logic, we are now able to use the register in two operating modes for a total of three different possibilities:

- non-memory register
- memory register, 16-bit-wide
- memory register, 8-bit-storage + 8-bit-configuration for the switch block

The new configuration register is shown in Figure 3-7B. It is different from the previous implementation not only by its dimension (previously 20 bits, now 21 bits) or by the logic involved in the connectivity, but by the manner it is working and by the manner it treats the configuration pattern. The multiplexers that were added allow flexibility in establishing the necessary data paths. However, when operating as a non-memory element, the register has the very same behavior as in the case of the previous design. Everything is preserved with this respect, all the connections and the bits are identical with the corresponding ones from the original design. Exception in this case is the extra bit added, bit 20, which is set to logical value "0".

When operating in the memory mode, the extra multiplexers are used for building two distinct ways of functioning. Bits 16 to 9 are always used exclusively for storing purposes. Bits 8 to 1 are used depending upon which memory mode the element is into. For the purpose of selecting the length of the storage area when operating as a memory, the value stored in the flip-flop at configuration time is used. This means that for a logical value of "1" stored in the flip-flop, the molecule will behave like 16-bit-wide memory molecule and for a logical value of "0" it will operate as an 8-bit-wide memory molecule with the capability of driving the switching block. This way, if the possibility of re-routing the switch block driven connections is desired, it can be achieved by using this operating mode.

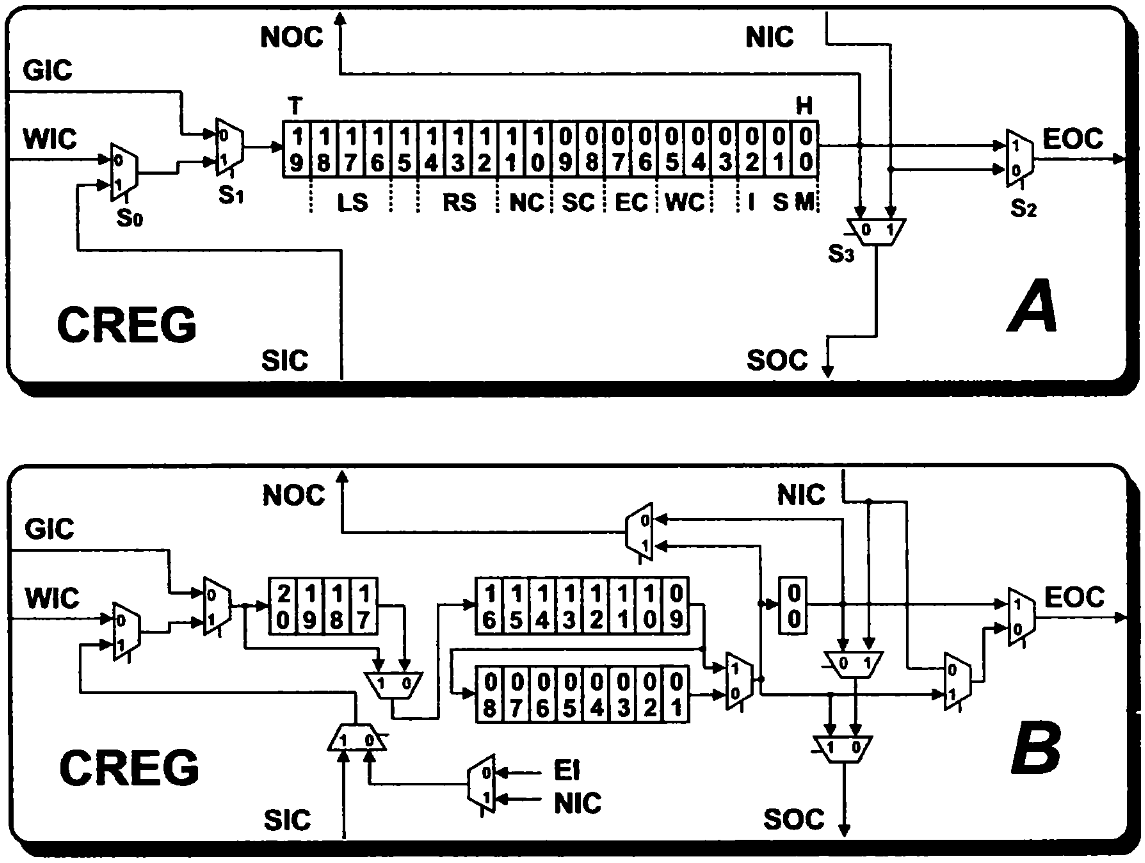


Figure 3-7: Configuration register details: A. Initial design; B. Modified design.

If the molecule is operating in 16-bit-wide mode, than bits 8 to 1 are also used as storage bits, together with bits 16 to 9, thus providing a 16-bit-wide storage field, from bit 16 to bit 1. If the element is in the 8-bit-wide operating mode, bits 8 to 1 are used as configuration bits for the switching block inside the MuxTree element, the multiplexers allowing the shifting process only for the bits that are actually used for the memory. This means that the bits used as configuration bits (bits 20 to 17 and, depending on the corresponding memory mode, bits 8 to 1) are not shifted when the storage bits are. This prevents corrupting the configuration pattern in each MuxTree molecule.

## 3.5 The Molecular Code

Each MuxTree molecule is loaded with a binary configuration, which forms the molecular code (MOLCODE) [100]. This code allows the molecule to operate in either one of the following two modes: the *logic mode* and the *memory mode*. At configuration time, the molecular code (MOLCODE) is loaded into the molecule. The MOLCODE $MC_{21:0}$ is stored into a group of two components, one being the flip-flop FF (Q) and the other being the configuration register

($CREG_{20:0}$), as shown in Figures 3-8 and 3-11. The operating mode is selected by the value of bit $M=CREG_{20}$:

- M=0: logic mode (all combinational logic resources are available for user);
- M=1: memory mode (no combinational logic resources are available, except the switch block in the short memory mode).

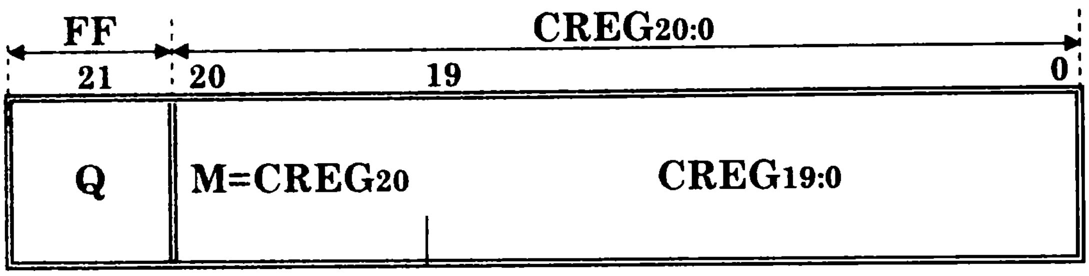


Figure 3-8: The molecular code (MOLCODE) $MC_{21:0}$.

### 3.5.1 The Logic Mode (M=0)

#### 3.5.1.1 General Description

The logic mode is defined by M=CREG20=0. In this mode, all combinational logic resources of the MuxTree molecule are available for the user. The molecular code corresponding to the logic mode is shown in Figure 3-9.

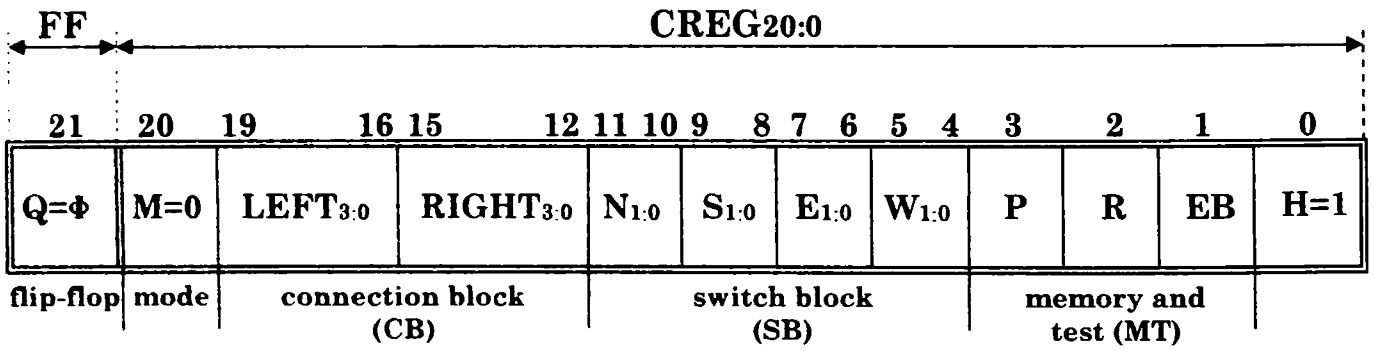


Figure 3-9: The MOLCODE ($MC_{21:0}$) in logic mode.

Let us detail the meaning of each of the MOLCODE bits, from the right to the left [67]:

- Bit H ($CREG_0$) is always set to the logical value "1". This indicates that the current molecule was loaded with the MOLCODE.
- Memory and test bits (MT or $CREG_{3:1}$) are special configuration bits (Figure 3-11):
  - bit EB ($CREG_1$) is the control variable for multiplexer $M_1$;
  - bit R ($CREG_2$) is the control variable for multiplexer $M_2$; it toggles between sequential (R=1) and combinational (R=0) mode;
  - bit P ($CREG_3$) contains the value for the asynchronous preset of the flip-flop FF; at the rising edge of INIT, the flip-flop will load the value of bit P.
- Bits $N_{1:0}$, $S_{1:0}$, $E_{1:0}$ and $W_{1:0}$ (SB or $CREG_{11:4}$) drive the switch block *SB*, as shown in Figure 3-10. They contain the control variables for all four possible directions: west, east, south and north. The *SB* provides all the combinations of interconnections between the inputs and the outputs from the four

directions, i.e. how different signals that enter a molecule can be routed or deviated along the desired path.

- Connection block bits (CB or $CREG_{19:12}$) drive the multiplexer connection block, as shown in Figure 3-11:
  - bits $LEFT_{3:0}$ ($CREG_{19:16}$) select the input of multiplexer $M_3$;
  - bits $RIGHT_{3:0}$ ($CREG_{15:12}$) select the input of multiplexer $M_4$.

  At the moment, bits $LEFT_3$ and $RIGHT_3$ are not used (they are reserved for further developments).
- Bit M=0 ($CREG_{20}$) indicates the logic mode.

  Bit Q (flip-flop FF) is not used when in logic mode. The Q bit is redundant when operating in the logic mode; after charging the molecule with the MOLCODE, bit P will override the influence of bit Q and the flip-flop will be set according to the value of P bit.

### 3.5.1.2 An Example

Let us consider a simple example of artificial organism, a single cell (Figure 3-12) realizing a modulo-4 up-down counter [67] defined by the following sequences:

- for $M=0$: $Q_1Q_0 = 00 \to 01 \to 10 \to 11 \to 00 \to \ldots$ (counting up);
- for $M=1$: $Q_1Q_0 = 00 \to 11 \to 10 \to 01 \to 00 \to \ldots$ (counting down).

It can be verified that the two ordered binary decision diagrams Q1+ and Q0+ of Figure 3-12A and B (where each test element is represented by a diamond with a single input, a “true” output, and a “complemented” output identified by a small circle) represent a possible realization of the counter [52, 67]. The leaf elements, represented as squares, define the output values of the given functions ($Q_1^+$ and $Q_0^+$ in the example) computed with the following equations:

$$Q_1^+ = M\left(Q_1 \cdot Q_0 + Q_1' \cdot Q_0'\right) + M'\left(Q_1 \cdot Q_0' + Q_1' \cdot Q_0\right)$$
$$Q_0^+ = Q_0'$$

The direct implementation of the ordered binary decision diagrams on silicon follows the layout from Fig. 3-12C. Each test element is implemented with one MuxTree® molecule, keeping the same interconnection diagram and assigning the values of the leaf elements to the appropriate multiplexer inputs. The two state functions $Q_1$ and $Q_0$ are the outputs of the D flip-flops of the top row of the MuxTree® molecules.

The cell implementing the counter has therefore 3 rows and 2 columns of MuxTree molecules [100]. From the multiplexer diagram of Figure 3-12B and from the description of the MuxTree molecule in logic mode (Figures 3-10 and 3-11) we can compute the control bits of each molecular code, finally generating the MOLCODEs of Figure 3-12C, each MOLCODE being a word of six hexadecimal digits (00QM, $CREG_{19:16}$, $CREG_{15:12}$, $CREG_{11:8}$, $CREG_{7:4}$ and $CREG_{3:0}$). The string of MOLCODEs is defined as the *ribosomic genome* RG [67].

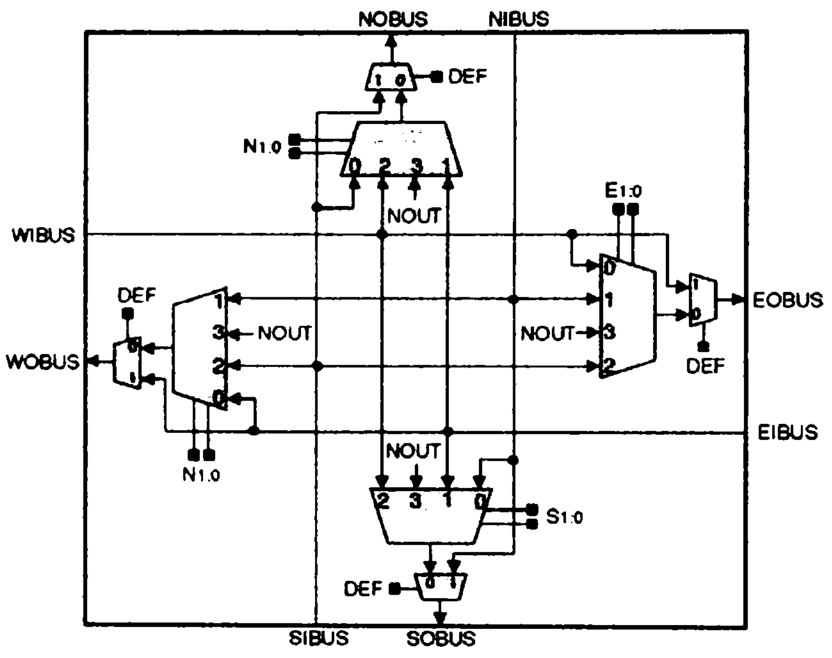


Figure 3-10: The switch block's (SB) internal architecture
$(DEF = M \cdot Q = CREG_{20} \cdot Q)$.

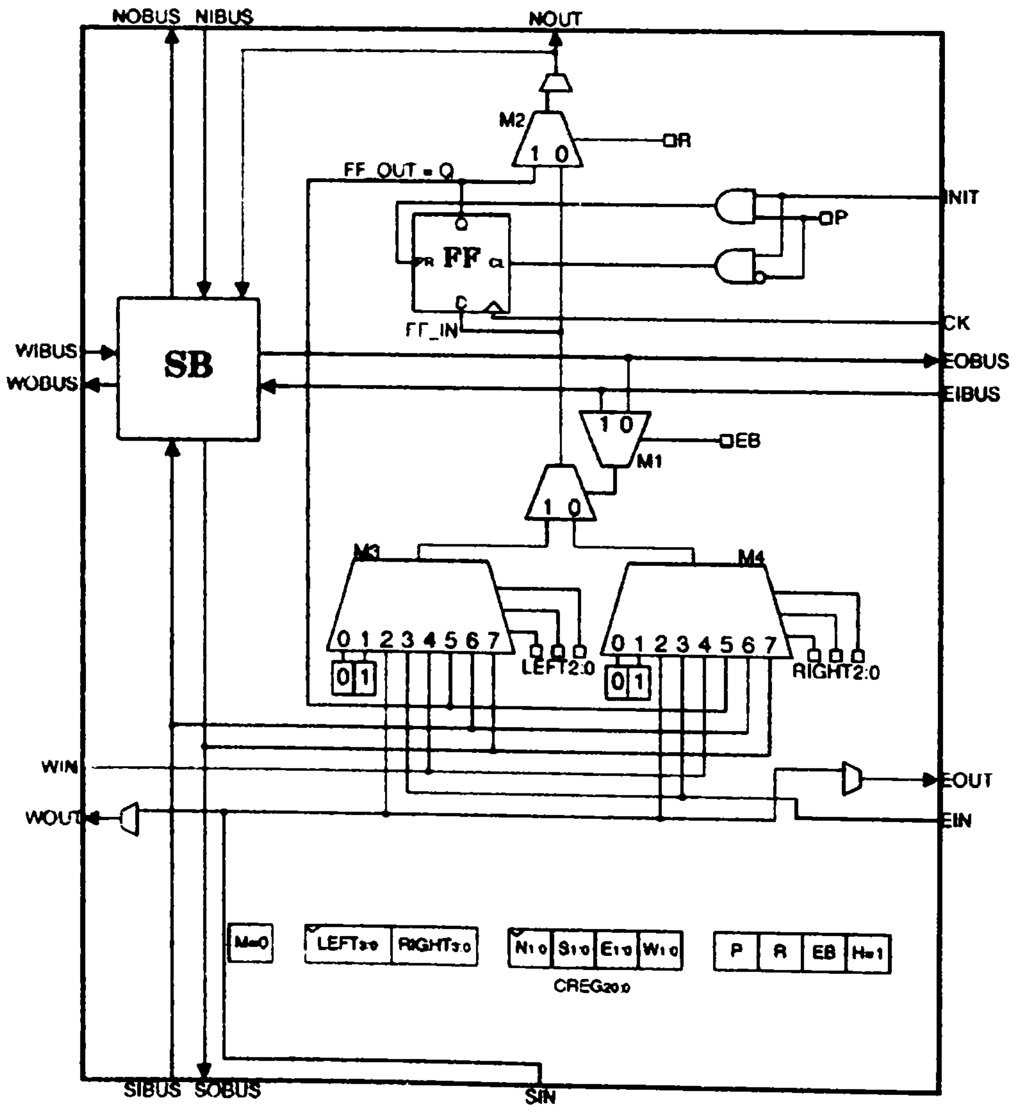


Figure 3-11: Overall structure of a MuxTree molecule in logic mode.

## 3.5.2 The Memory Mode (M=1)

### 3.5.2.1 General Description

The memory mode is defined by M=CREG20=1. In this mode, the operating internal resources of the MuxTree molecule are the configuration register(CREG), the flip-flop (FF) and, possibly, the switch block (SB). Part of the configuration register is used as a storage memory [100]. Depending on the value of $Q = MC_{21}$, we have two memory sub-modes:

- $Q = 0$: *short memory* with switch block, the storage size being 8 bits;
- $Q = 1$: *long memory*, the storage size being 16 bits.

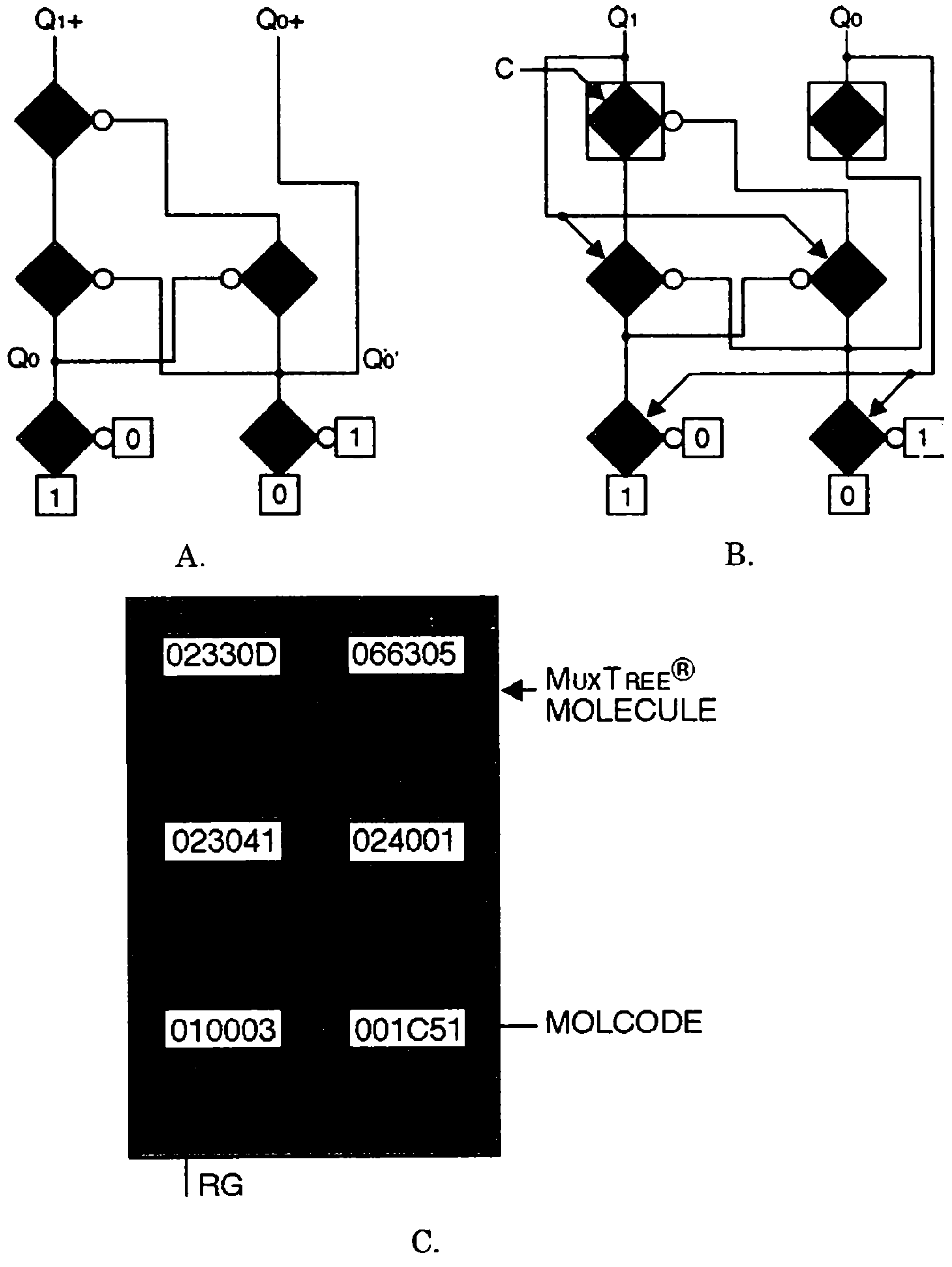


Figure 3-12: Modulo-4 up-down counter. (A) Ordered binary decision diagrams for Q1+ and Q0+. (B) Multiplexer diagram using MuxTree® molecules. (C) 6 MuxTree® molecule cell; RG: ribosomic genome.

A MuxTree memory molecule is presented as a block schematic in Figure 3-13A. It uses input and output connections (IO connections) in all four possible directions. The IO connections are internally routed at configuration time and they build the memory structure [128]. For this, each molecule has to be loaded with a specific molecular code which will strictly determine its behavior, and which is related to its position inside the structure. This is achieved by using bits $MEM_{2:0}$ ($CREG_{3:1}$), which are special configuration bits, defining the role of the molecule inside a memory area. They are common to both memory sub-modes and are shown in Table 3-1. The shape of a memory structure is that of a common rectangle (i.e. its horizontal and vertical dimensions are not fixed, they are to be chosen by the user) and is formed by MuxTree molecules operating in memory mode. The stored data are continuously shifted (one bit per *CK* period) and the user has access to it at every molecule situated in the upper row of the structure (output *NOUT*).

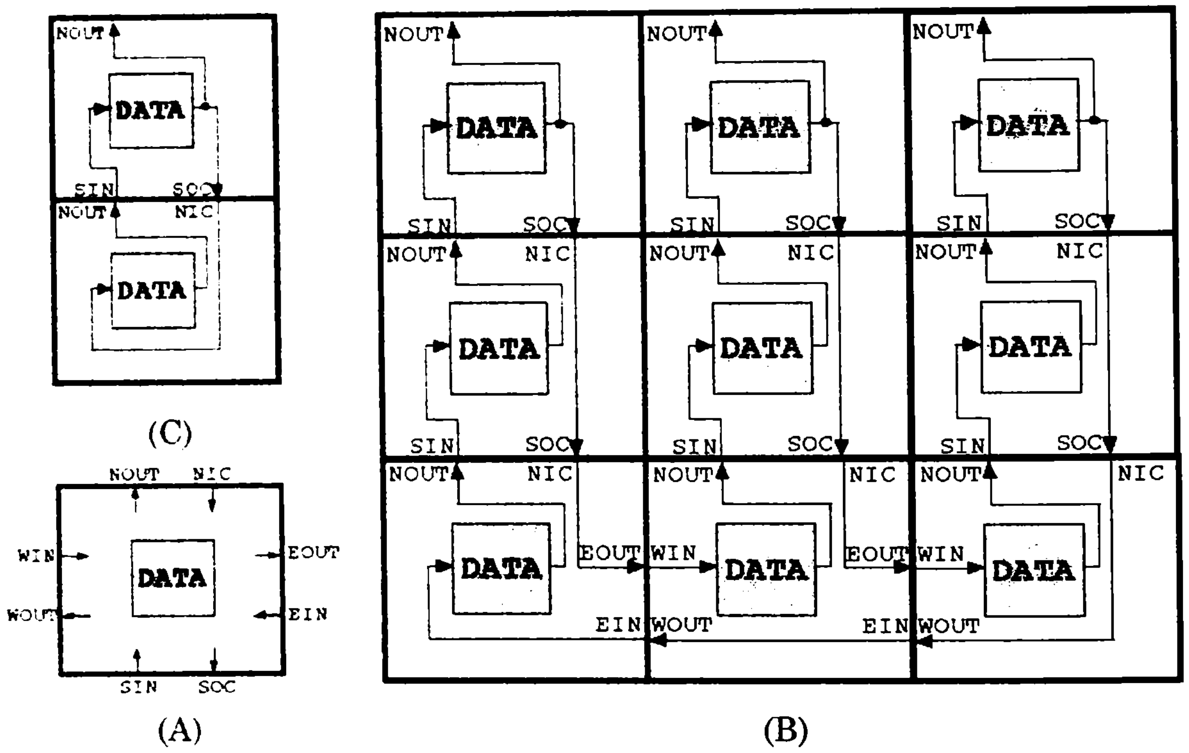


Figure 3-13: The memory mode: (A) Block schematic of a memory-configured molecule; (B) A macroscopic view of a 3x3 shift-memory area; (C) A 2x1 shift-memory column.

| $MEM_2$ | $MEM_1$ | $MEM_0$ | MOLECULE'S POSITION | |
|---|---|---|---|---|
| 0 | 0 | 0 | Border to the south | BS |
| 0 | 0 | 1 | Lower right corner | RC |
| 0 | 1 | 0 | Lower left corner | LC |
| 0 | 1 | 1 | Bottom of a single column | BC |
| 1 | 0 | $\Phi$ | Border to the north with data output | BN |
| 1 | 1 | $\Phi$ | No border | NB |

Table 3-1: The molecule's possible positions and their respective configuration bits.

Figure 3-13B shows the routing of the IO connections in a 3x3 shift-memory area. The stored data are shifted as follows: from the bottom left molecule, data serially enter the molecules situated upwards in the same column. Once the top molecule is reached, data are routed to the bottom molecule from the next column to the right. Data are then shifted upwards, toward the top molecule, then go to the bottom molecule of the next column and thus the process repeats itself. When data reach the top molecule from the rightmost column, they are routed to the bottom molecule and from there data are crossing, from the right to the left, every bottom molecule. When reaching the bottom leftmost molecule, one complete shifting is completed.

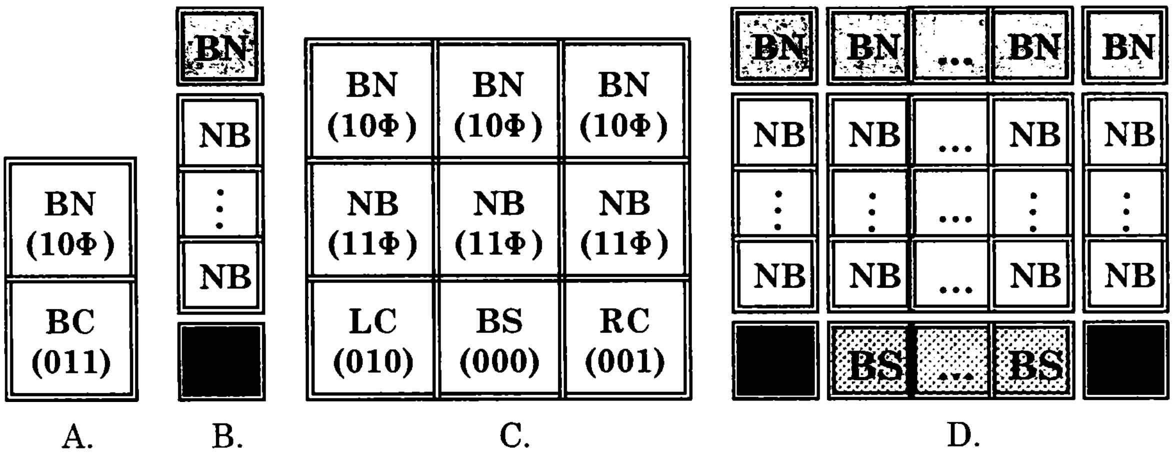


Figure 3-14: Memory structures and their molecular configurations MEM2:0: (A) The minimal shift-memory structure (2x1); (B) The general structure of a single column shift-memory; (C) A 3x3 shift-memory structure; (D) The general structure of a shift-memory.

We implemented the MuxTree molecule allowing a large choice for the user when specifying the dimensions of a memory area. The minimal structure is a single column (a 2x1 rectangle shown in Fig. 3-14A), but there are no upper limits defined for the dimensions of the memory rectangle – they are set by simply configuring the memory molecules with the appropriate MOLCODEs. The general structure of a single column memory is presented in Fig. 3-14B and the general memory rectangle is shown in Fig. 3-14D. The memory configuration bits (MEM2:0) for the structure presented in Fig. 3-13B, respectively Fig. 3-13C, are shown in Fig. 3-14C, respectively Fig. 3-14A. A memory structure cannot be implemented without using the LC, RC, BS and BN-type molecules or BC and BN-type molecules, according to the type of structure (single column or rectangle). Therefore these molecules appear with a darker color in Fig. 3-14.

The memory we implement with MuxTree® molecules is not an addressable memory, i.e. ROM-like. It is a continuously shifting memory, somewhat similar to a huge shifting register. This is a cost-effective solution that allows a sufficient functionality while keeping the logic involved to an acceptable level of complexity.

The memory structure has data output ports in all molecules marked as BN. This means the user has access to the stored data only in BN-type molecules. The bit-stream coming out through a data output port starts with the least significant bit stored inside the corresponding molecule (CREG12 in short memory mode or $CREG_4$ in long memory mode) and is shifted from the left (most significant bit) to the right (least significant bit) every *CK* clock cycle.

### 3.5.2.2 The Short Memory Mode (Q=0)

One of the two memory sub-modes is the short memory with switch block. In this mode, each molecule can store 8 bits of genome data while keeping the functionality of the switch block. The MOLCODE bits are shown in Figure 3-15:

- Bit H ($CREG_0$) is always set to the logical value "1" and indicates that the current molecule was loaded with the MOLCODE;
- Molecule's position bits ($MEM_{2:0}=CREG_{3:1}$) were presented in Subsection 3.5.2.1 and shown in Table 3-1;
- Bits $N_{1:0}$, $S_{1:0}$, $E_{1:0}$ and $W_{1:0}$ (SB or $CREG_{11:4}$) have the same attributes as in logic mode (Subsection 3.5.1);
- Bits $DATA_{7:0}$ ($CREG_{19:12}$) are used for storing genetic data;
- Bit M=1 ($CREG_{20}$) indicates the memory mode;
- Bit Q=0 (flip-flop FF) indicates the memory sub-mode.

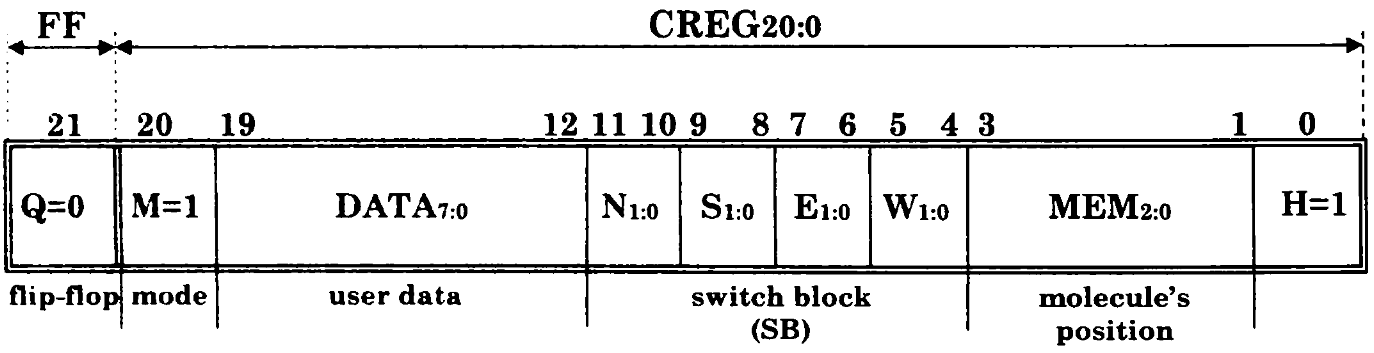


Figure 3-15: The short memory MOLCODE ($MC_{21:0}$).

The possibility of storing 8-bit-wide words of genome data, while keeping the routing features of the switch block, provides a good functionality for the molecule. The functional parts of the molecule in this case are the switch block *SB*, the configuration register CREG and the flip-flop FF, all shown in Figure 3-16. The parts that are not available for the user are drawn with lighter gray.
The data path is suggested in Figure 3-16. The configuration register includes the storage part of the molecule ($CREG_{19:12}=DATA_{7:0}$). Bits are shifted each *CK* clock cycle; the user data from the previous molecule enter through *SIN*, *EIN* or *WIN* connections, are then shifted through $CREG_{19:12}$ and then exit the molecule through *NOUT* and/or *SOC* connections, as well as all buses via SB in order to enter the next molecule. From a macroscopic point of view, the data flow is shown in Figure 3-13B, depending on the molecule's position, i.e. the value of $MEM_{2:0}$.

### 3.5.2.3 The Long Memory Mode (Q=1)

The other memory sub-mode is the long memory mode [100]. In this mode, each molecule can store 16 bits of user data with the price of losing the functionality of the switch block. The MOLCODE bits are shown in Figure 3-17:

- Bit H ($CREG_0$) is always set to the logical value "1"; this indicates that the current molecule was loaded with the MOLCODE;

- Molecule's position bits ($MEM_{2:0}$=$CREG_{3:1}$); they were presented in Subsection 3.5.1. and shown in Table 3-1;
- Bits $DATA_{15:0}$ ($CREG_{19:4}$) are used for storing the operative genome's data;
- Bit M=1 ($CREG_{20}$) indicates the memory mode;
- Bit Q=1 (flip-flop FF) indicates the memory sub-mode.

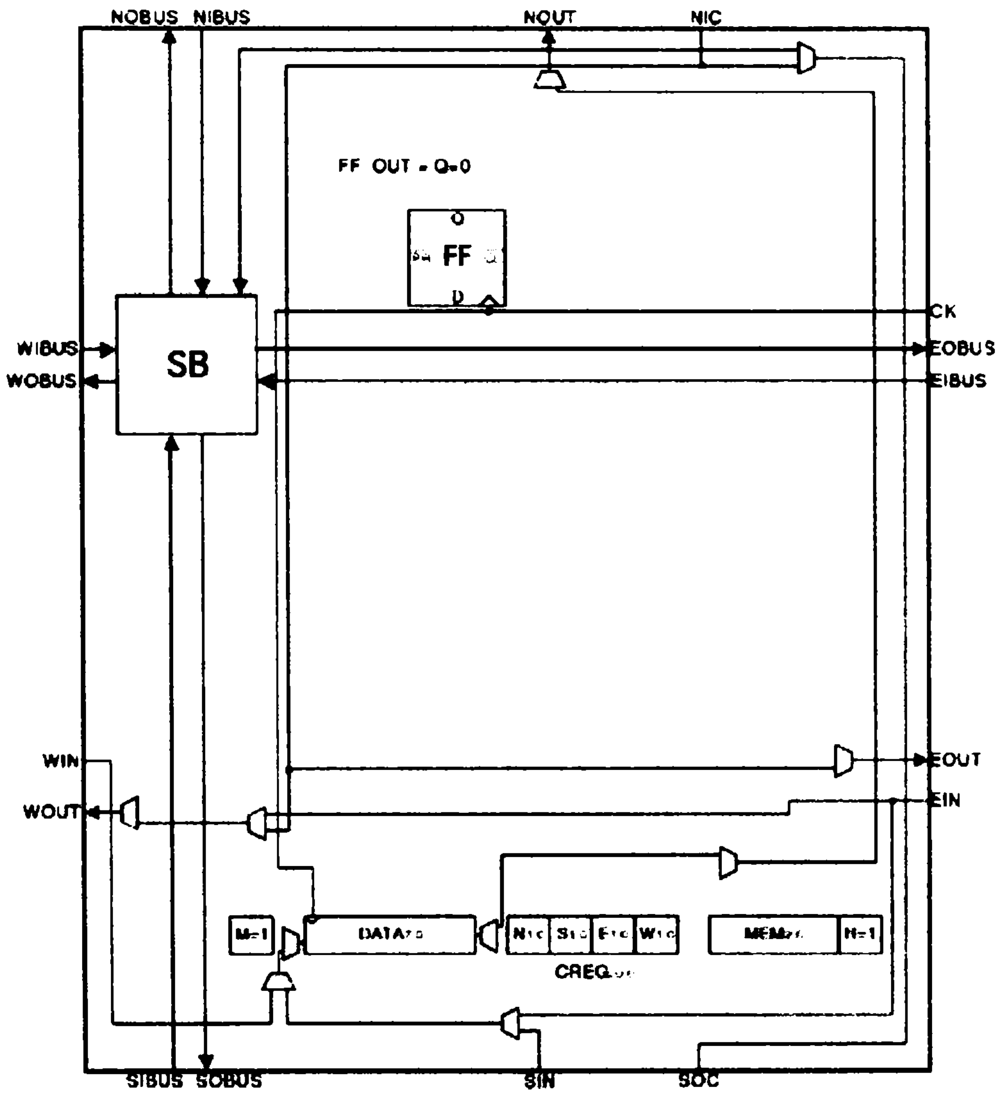


Figure 3-16: Overall structure of a MuxTree molecule in short memory mode.

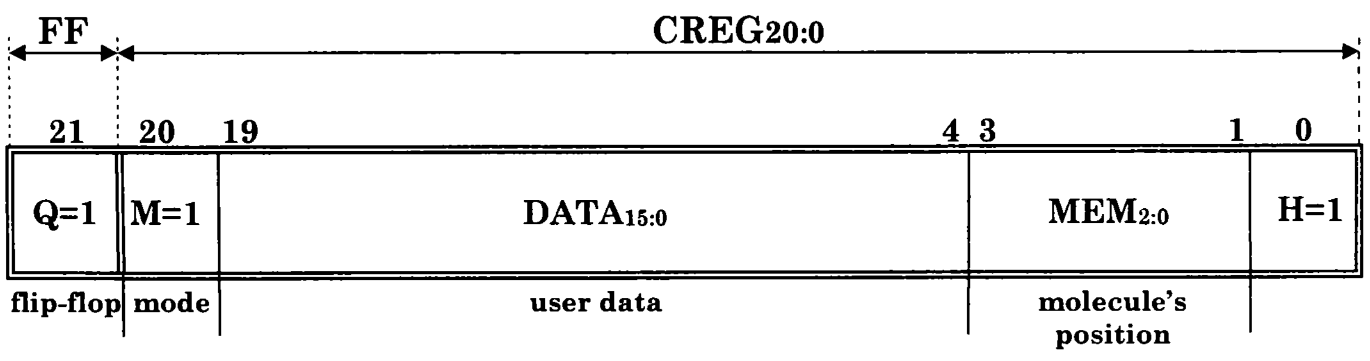


Figure 3-17: The long memory MOLCODE ($MC_{21:0}$).

The possibility of storing 16-bit-wide words of genome data provides double storage capacity when compared to the short memory mode. There is, however, a disadvantage: because bits $CREG_{11:4}$ that normally drive the switch block are now used for storage, the switch block is disabled. In order not to lose all the bus routing facilities, we designed the SB to also operate in long memory mode, but instead of actually routing the buses it simply implements a pass-through: *SIBUS* is directly connected to *NOBUS*, *NIBUS* is directly connected to *SOBUS*, *EIBUS* is directly connected to *WOBUS* and *WIBUS* is directly connected to *EOBUS*.

This means greater storage capacity at the expense of flexibility. The functional parts of the molecule in this case are the configuration register CREG and the flip-flop FF, all shown in Figure 3-18. The parts that are not available for the user are drawn with a light gray.

The configuration register includes the storage part of the molecule ($CREG_{19:4}=DATA_{15:0}$). Bits are shifted each *CK* clock cycle; the genetic data from the previous molecule enter through *SIN*, *EIN* or *WIN* connections, are then shifted through $CREG_{19:4}$ and then exit the molecule through *NOUT* and/or *SOC* connections in order to enter the next molecule. From a macroscopic point of view, the data flow is shown in Figure 3-13B.

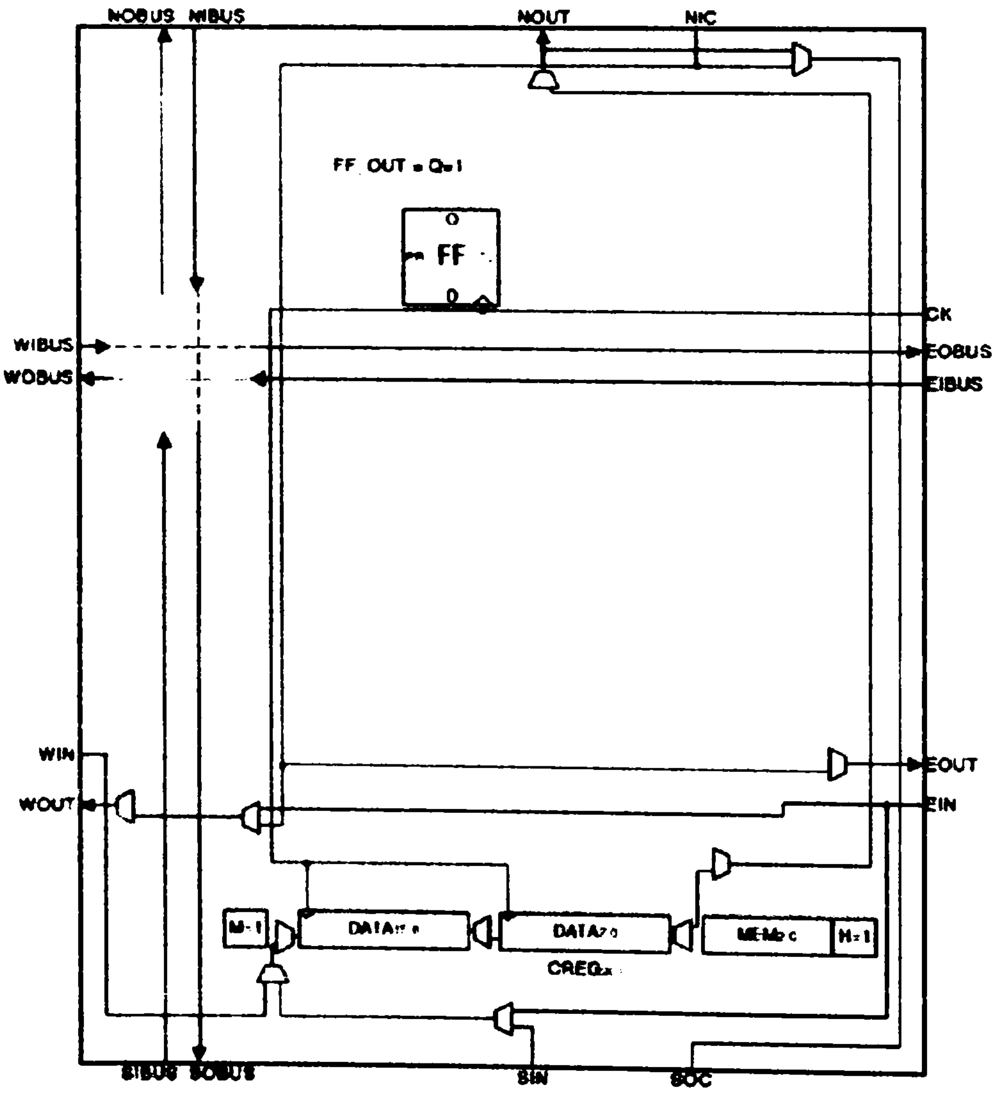


Figure 3-18: Overall structure of a MuxTree molecule in long memory mode.

### 3.5.2.4 An Example

A modulo-6 synchronous counter can be implemented with the microprogram shown in Table 3-2. The microprogram was written in the PICOPASCAL language [64].

| Address | DATA | DATA$_{2:0}$ | PICOPASCAL |
|---|---|---|---|
| 00 | 5 | 101 | while H |
| 01 | 6 | 110 | while H' |
| 02 | 2 | 010 | if [$Q_2$] |
| 03 | 2 | 010 | if [$Q_1$] |
| 04 | 0 | 000 | do 0 |
| 05 | 0 | 000 | do 0 |
| 06 | 0 | 000 | do 0 |
| 07 | 3 | 011 | else |
| 08 | 2 | 010 | if [$Q_0$] |
| 09 | 0 | 000 | do 0 |
| 0A | 0 | 000 | do 0 |
| 0B | 0 | 000 | do 0 |
| 0C | 3 | 011 | else |
| 0D | 1 | 001 | do 1 |
| 0E | 0 | 000 | do 0 |
| 0F | 1 | 001 | do 1 |
| 10 | 4 | 100 | endif |
| 11 | 4 | 100 | endif |
| 12 | 3 | 011 | else |
| 13 | 2 | 010 | if [$Q_1$] |
| 14 | 2 | 010 | if [$Q_0$] |
| 15 | 0 | 000 | do 0 |
| 16 | 0 | 000 | do 0 |
| 17 | 1 | 001 | do 1 |
| 18 | 3 | 011 | else |
| 19 | 1 | 001 | do 1 |
| 1A | 1 | 001 | do 1 |
| 1B | 0 | 000 | do 0 |
| 1C | 4 | 100 | endif |
| 1D | 3 | 011 | else |
| 1E | 2 | 010 | if [$Q_0$] |
| 1F | 0 | 000 | do 0 |
| 20 | 1 | 001 | do 1 |
| 21 | 0 | 000 | do 0 |
| 22 | 3 | 011 | else |
| 23 | 1 | 001 | do 1 |
| 24 | 0 | 000 | do 0 |
| 25 | 0 | 000 | do 0 |
| 26 | 4 | 100 | endif |
| 27 | 4 | 100 | endif |
| 28 | 4 | 100 | endif |
| 29 | 7 | 111 | end |
| 2A | 7 | 111 | end |
| 2B | 7 | 111 | end |

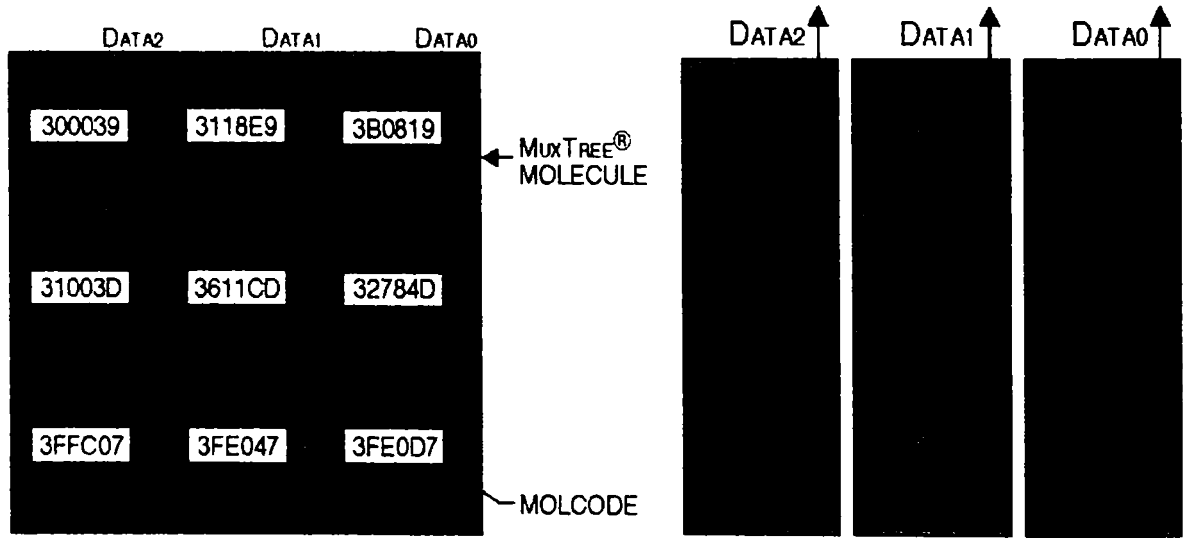
| | | | |
|---|---|---|---|
| **2C** | 7 | 111 | end |
| **2D** | 7 | 111 | End |
| **2E** | 7 | 111 | End |
| **2F** | 7 | 111 | End |

Table 3-2: A microprogrammed modulo-6 counter.

The microprogram shown in Table 3-2 consists of 42 memory words (addresses from 00 to 29 in hexadecimal), each word being 3 bits long (DATA in decimal, DATA2:0 in binary). For storing this program we use the following memory structure (Fig. 3-19):

- 3 single column shift-memories, each column consisting of 3 molecules operating in long memory mode (Fig. 3-19A); each column computes one of DATA bits (DATA2, DATA1 and DATA0);
- Each shift-memory column stores 48 bits of data, as we employ 3 molecules per column, each molecule storing 16 bits of data; because the microprogram needs only 42 words, the last memory entries, from address 2A to address 2F, repeat the last instruction (DATA=7="end").

During the configuration phase, the string of MOLCODEs, which is defined as the ribosomic genome RG, is entered according to Fig. 4-10C. During normal operation, due to MEM2:0 values, the actual connections are those of Fig. 3-19B; the memory data of Fig. 3-19B constitute the program of a binary decision machine and are finally defined as the *operative genome* OG [67].



(A) (B)

Figure 3-19: A microprogrammed modulo-6 counter. (A) The memory structure using 3 macro-molecules, each with a single column configuration. (B) The molecular codes.

## 3.6 The HOLD Mechanism

Our approach of designing the additional memory mode led us to build a continuously shifting memory (Section 3.5 and Figure 3-19B). It is now possible to have a genetic program stored inside the memory and be executed for an indefinite period of time, one instruction per clock cycle. The reasons for which we chose to implement such a type of cyclic memory, as well as the constraints this architecture imposes, were mentioned in Section 3.2. A possible important limitation that was discussed was the lack of branch-type instructions, an aspect that made us consider implementing a mechanism that, while certainly not solving this issue, would however bring some control over program execution by halting it when needed. Such a feature, together with the possibility of storing parts of the same genetic program inside different macro-molecules allows altering the initial synchronization, thus providing an form of control transfer, simulating indirectly the effects of a jump-type instruction.

In order to stop the shifting process, some *memory hold* mechanism is required. Essentially, this mechanism allows the memory to shift as long as the *HOLD* signal of the memory area is low (logic "0"). As soon as this signal becomes high (logic "1") the memory shifting is prevented. The *HOLD* signal is connected to the south input signal *SIN* from the bottom left molecule (the bottom left corner) of the memory area (the *SIN* signal, in this particular molecule, is not used after configuration time in any of the two memory modes: Fig. 3-13) [100]. The *HOLD* signal propagates itself to all memory molecules of the area, along the path shown in Figure 3-20.

In the case where *K* several distinct memory areas are simultaneously used, it is possible to define *K* different *HOLD* signals $HOLD_1$, $HOLD_2$,...,$HOLD_k$. Independently of the number of *HOLD* signals, it is imperative that the correct routing of these signals be assured or the value *HOLD*=0 be set if there is no need to interrupt the normal shifting process.

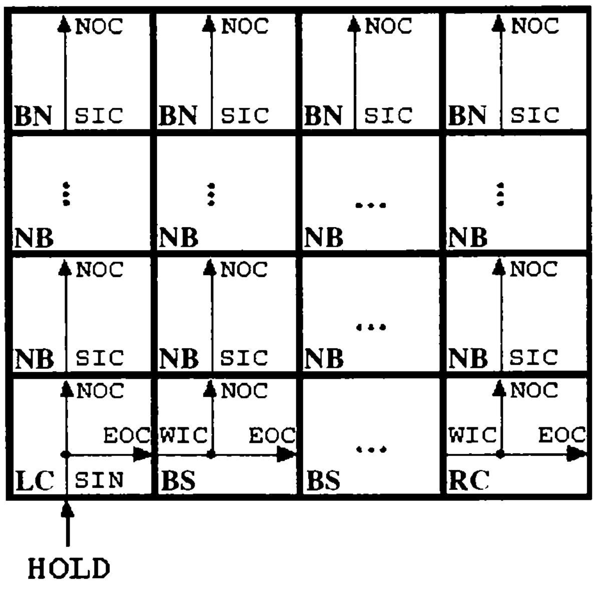


Figure 3-20: Propagation of the *HOLD* signal inside a shift-memory area.

## 3.7 Multiple-Level Fault-Tolerance

Biological entities are continuously put under environmental stress. Wounds and illnesses resulting from such stress often cause incapacitating physical modifications. Fortunately, living beings are capable of successfully fighting the great majority of such wounds and illnesses, showing a remarkable robustness through a process that we call *healing* [102].

To endow artificial organisms in Embryonics with similar features, a two-level mechanism for self-repair is provided, stretching over the first two levels of organization (see Figure 2-2) as reconfiguration processes, first at the molecular level, and then at the cellular level [63]. This hierarchical approach of self-repair allows for an effective way of tolerating faults: it has the capacity of reconfiguring according to different severity levels (addressing the least severe first, represented by faulty molecules, and then the most severe, represented by faulty cells), while it inflicts minimal resource losses through reconfiguration (replacing a "dead" molecule is certainly less expensive than replacing an entire, albeit "killed", cell).

The population level is inherently redundant, as the presence of several organisms implies that the loss of an individual is not vital to the system.

### 3.7.1 Self-Repair at the Molecular Level

The most original features of the entire MuxTree design are the self-testing [1, 2, 126], self-repairing [55, 56, 66, 102, 128, 131, 133] and self-replication mechanisms [54, 55, 120, 130]. Although the literature contains many solutions on how to design self-testing [31, 43, 45, 77] and even self-replicating structures [8, 47] the particular approach within the Embryonics project could not be an off-the-shelf one due to the uniformity and universality of the molecules and cells. Therefore, the chosen approach was of a two-level self-repair process, first at the molecular level (less expensive), and then at the cellular level (more expensive). Furthermore, the main goal was to improve the existing design without actually causing any losses when original capabilities were concerned; because these were the particular subject of a previous thesis' work, we will only provide details when they are of use to a better understanding for the development of the project in general and of RAM-MuxTreeSR in particular.

Implementing a self-repair procedure for a memory means additional difficulties, after all, a memory is dynamic, it is not subject to stuck-at faults only, it may also experience intermittent and transient faults, which are far more difficult to detect. In fact the self-repair method used for the MuxTreeSR design was entirely preserved after the addition of the memory operating modes. We will go into more details to shown how this mechanism accommodates the memory structures from RAM-MuxTreeSR.

The self-testing process takes place at initialization time when a test pattern is sent inside all configuration registers [128]. The MuxTree molecule is subject to stuck-at faults, which may also be externally injected by means of a switch operated by the user. If the register is faulty, the situation is detected by the self-testing mechanism and the molecule is replaced by a spare one, all the connections being re-routed so as to avoid the faulty molecule, which enters the state DEAD. If there are no spare molecules left to replace the faulty one, than the cell itself can no longer function properly (it can no longer repair its faults) and it will be disabled (or *KILL*ed).

Figure 3-21 presents the principles of the repairing process in the case of the A molecule being faulty (the bottom left molecule). The configuration of the entire row that includes A is shifted one position to the right, taking advantage of the spare molecule. Once the re-routing process is finished and the configuration shift is done, the spare molecule becomes active (it enters the ACTIVE state) and the faulty one enters the state DEAD.

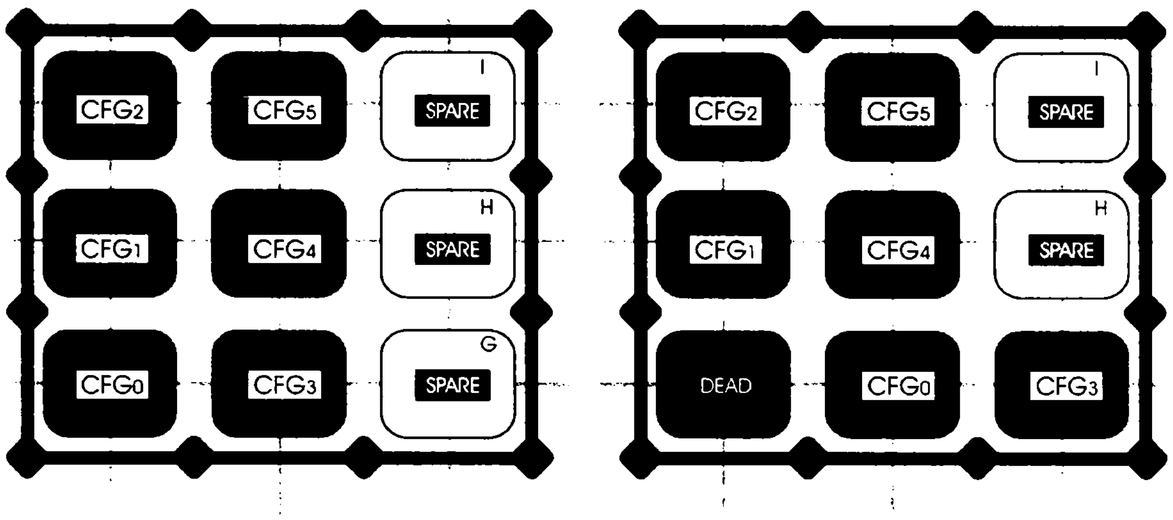


Figure 3-21: The repairing process in the case of a faulty A molecule.

Figure 3-22 presents the repairing process for the D molecule being faulty (bottom, middle). The configuration of the entire row that includes D is shifted one position to the right – starting from the D molecule, the previous one(s) remaining unaffected by the repair process – in order to take advantage of the spare molecule G. When the re-routing process is finished and the configuration was shifted, the spare molecule becomes "D" and the "original" D one enters the state DEAD. The repair process is by no means different when taking the other rows into account. Re-routing all the connections between the MuxTree elements so as to preserve the functionality of the whole structure is difficult to illustrate; we therefore present the essential connections for a 3X3 memory structure, before and after the repairing process in Figure 3-23.

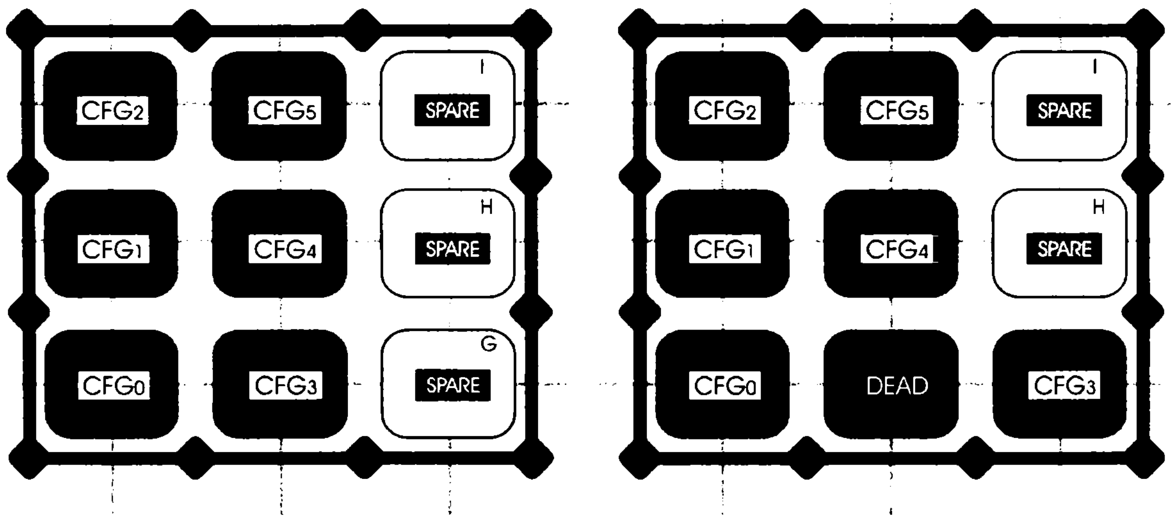


Figure 3-22: The repairing process in the case of a faulty D molecule.

Once the faulty molecule is located (E in Figure 3-23), it will be replaced by a spare one, and the data flow will entirely avoid it. Therefore molecule E "dies" and is replaced by the corresponding spare molecule H. The repairing process is similar in all the other cases.

As can be seen in Figure 3-23, the repairing process means re-routing all the connections in order to avoid the faulty molecule. This is done by activating a group of multiplexers placed at the border of each molecule and driven by the state of the molecule. The repairing process follows a precise scheme and after the state of the molecule changes to DEAD, the multiplexers will re-direct all connections going through the now faulty molecule. Since these multiplexers are integrated in the molecules, they are not shown in our figures (more details of the repairing process are given in [128]).

Unfortunately, there will always be situations when the repairs that are required outnumber the available spare molecules, no matter how frequent the spare columns are. Such an event does not have implications at the molecular level anymore, but results in a whole cell that is potentially faulty. In this situation a *KILL* signal will be generated, which will propagate throughout the cell, marking all molecules as being "dead". Figure 3-24 presents a possible scenario in a cell consisting of a 3x3 molecular array, where molecules E and H are both detected as being faulty. As the self-repair process is overcome, the entire cell will "die" and self-repairing measures will be taken at the superior organizational, cellular, level. An important remark is that the *KILL* signal is quite expensive in terms of available resources: whenever a too large number of repairs are necessary between two consecutive spare columns, the entire cell will be deactivated. Subsection 3.7.3 will present the mechanism implemented in the RAM-MuxTreeSR as online countermeasure to such situations.

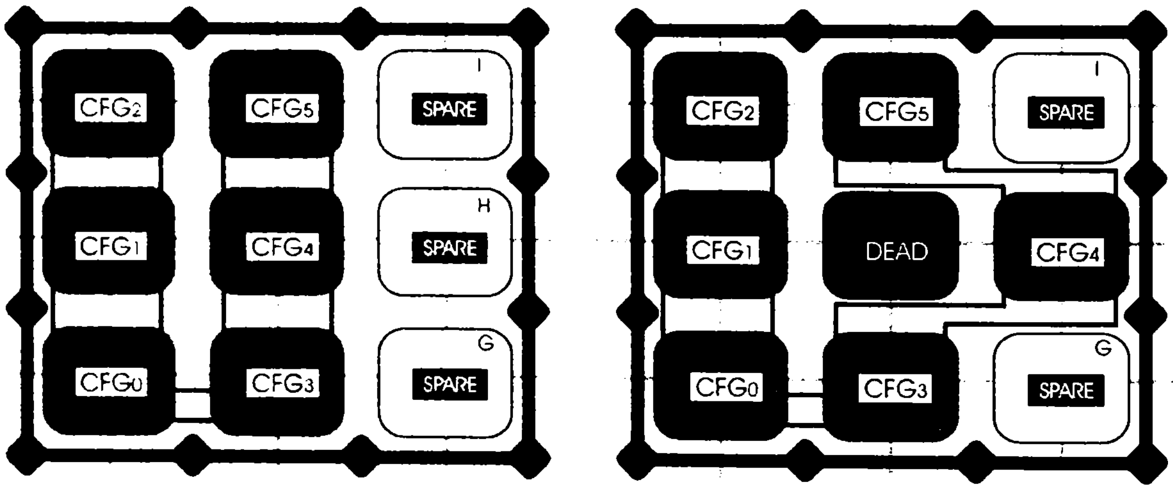


Figure 3-23: Details of the repairing process for molecule E.

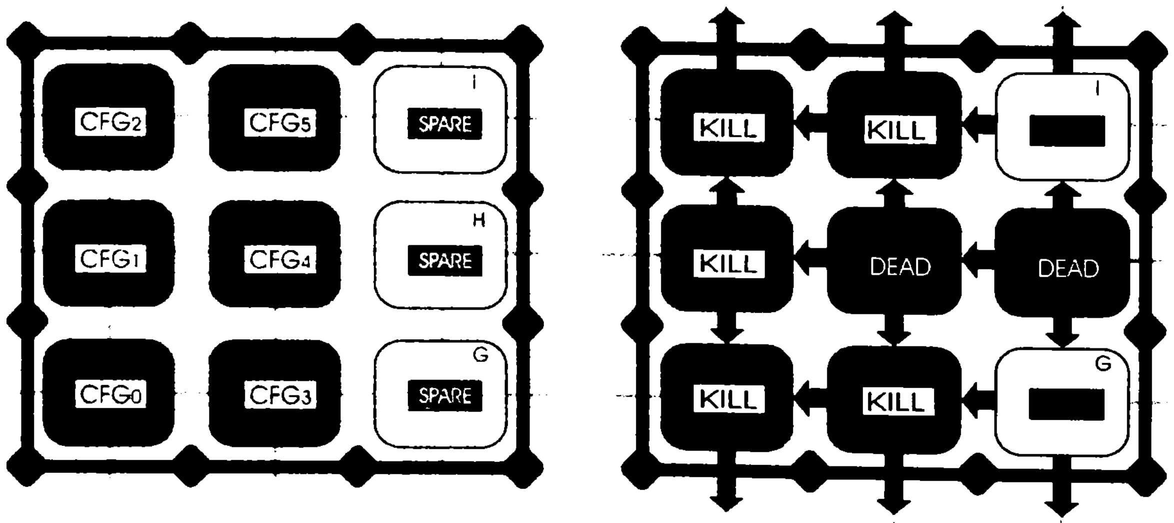


Figure 3-24: Death of an entire cell by activation of the *KILL* signal.

Thus far we presented how the self-repairing mechanism can be preserved from the MuxTreeSR design to actually accommodate the new memory operating modes introduced by the RAM-MuxTreeSR. We have shown that the reconfiguration processes can operate no matter what operating mode a molecule is operating in. However, the self-repair has to be triggered by an event associated with fault detection strategies; again, these strategies were successfully transferred into the new design and tested successfully. Unfortunately, fault detection strategies that were proven to work well under MuxTreeSR, covering a range of faults that could potentially affect either the FU or the CREG, can only protect the RAM-MuxTreeSR when operating in logic mode. They were originally intended to protect a molecule that was not supposed to change its internal configuration data in a dynamically fashion, as happens when operating in any of the memory sub-modes. A memory structure (and being cyclic makes no difference) changes its data and it is therefore very difficult to associate it with off-line testing strategies; therefore a different strategy must be put in place in order to properly extend the robustness over the macro-molecular structures. Adding fault-tolerance over the macro-molecules of the new RAM-MuxTreeSR design seems to be a natural development that will be the subject of the next chapter.

### 3.7.2 Self-Repair at the Cellular Level

The redundant storage of the entire genome in every cell is obviously expensive in terms of additional memory. However, it has the important advantage of making the cell *universal*, that is, potentially capable of executing any one of the functions required by the organism. This property, coupled with the coordinate-based gene selection, is a huge advantage for implementing a multiple level self-repair process. Since our cells (and molecules) are universal, with their expressed gene being determined by a coordinate mechanism, the system can survive the "death" of any one cell simply by re-computing the cells' coordinates within the array, provided of course that "spare" cells (i.e., cells which are not necessary for the organism, but are held in reserve during normal operation) are available [67, 128].

This particular surviving measure has to be taken once a cell cannot repair itself anymore at the molecular level, therefore triggering the *KILL* signal (see Figure 3-24). At the cellular level this actually means that the entire organism has to reconfigure itself, by disabling the whole column containing the “killed” cell and shifting its operational cells to the left. Of course, this is only possible at this level if there are spare cells available for reconfiguration. Figure 3-25 (which is quite similar to Figure 2-3) presents the repairing process for an organism consisting of an array of 3x2 cells. If we consider that the (2,1) cell (expressing gene C) has no more resources left for required self-repair, then at the molecular level the *KILL* signal will be triggered, which will effectively disable all of its molecules. At the cellular level this means that an entire column will be disabled (column 2 in our case), transforming into what in nature is called a “scar”; “healthy” cells are shifted to the right, this being the direction of self-repair. This particular implementation differs from that presented previously in Figure 2-3 for the purpose of reducing re-routing overhead.

This is how the molecular and the cellular levels of a system in Embryonics cooperate to assure a high degree of robustness: whenever the molecular level cannot handle the self-repair by itself anymore, it activates the self-repair at the cellular level so that the organism can continue to survive.

### 3.7.3 The *UNKILL* Mechanism

In digital electronic systems, the majority of hardware faults occurring in the silicon substrate are in fact *transient*, that is, they disappear after a short span of time. This observation is an important issue when designing self-repairing hardware: the parts of the circuit that have been "killed" because of the detection of a fault could potentially come back to "life" after a brief delay, therefore creating incentives for avoiding the stiff penalty induced by killing an

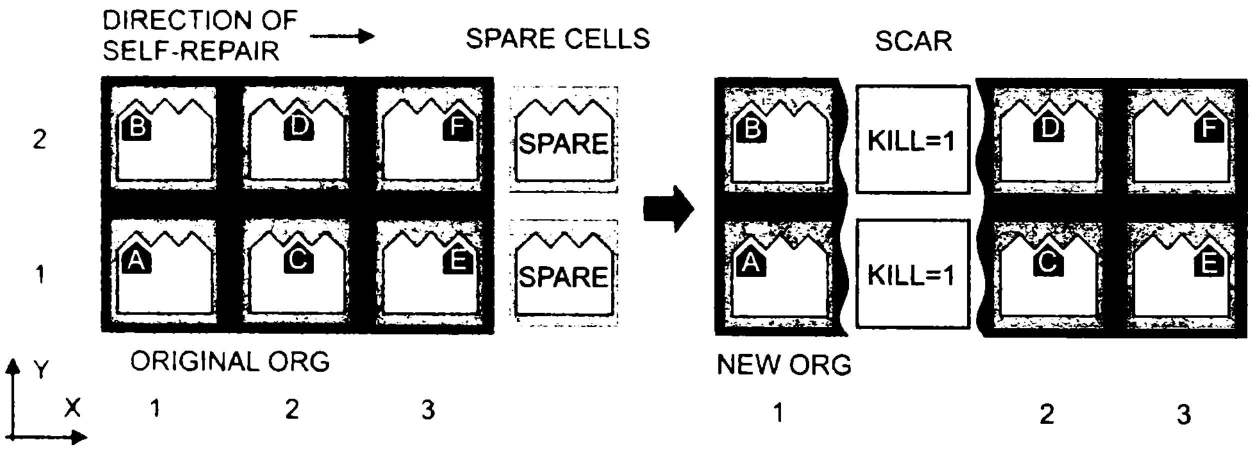


Figure 3-25: Self-repair at the cellular level [67, 102].

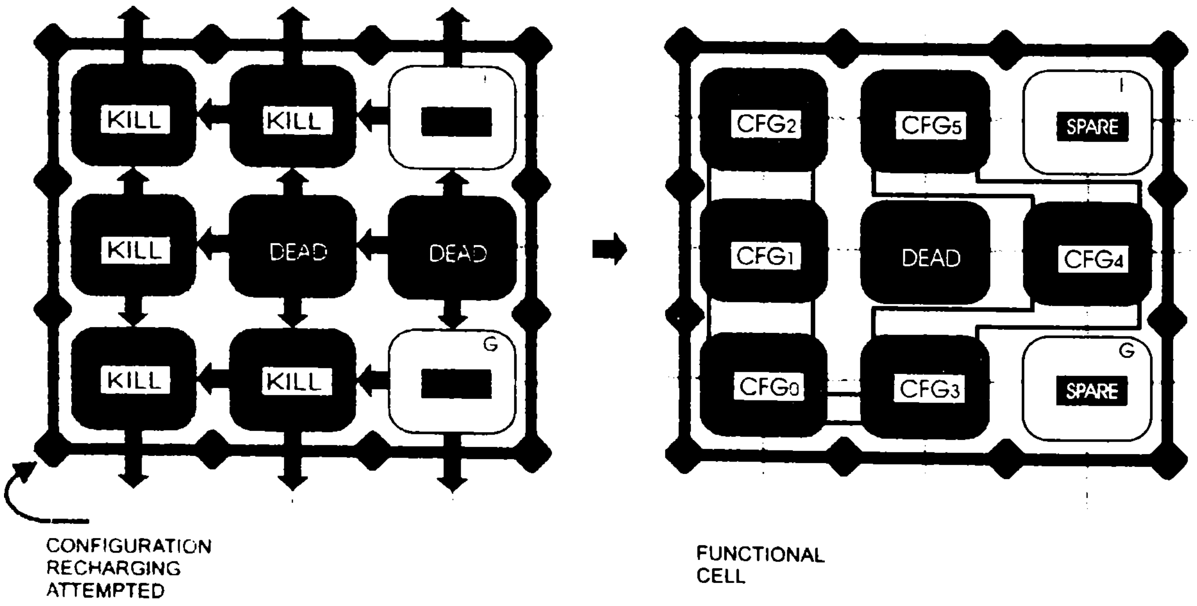


Figure 3-26: The result of the UNKILL process: a disabled cell is recovered from transient damage.

entire cell due to probably transient errors. However, detecting the disappearance of a fault and handling the "unkilling" usually requires a relatively complex circuit, thus preventing such an implementation at the molecular level. On the other hand, such a feature can be implemented at the cellular level with minimal resources [100, 102].

It is possible that a cell was "killed" due to unsuccessful repair attempts at the molecular level and at least some of the responsible faults were transient. If further reconfiguration of the cell is performed, then it is likely those faults

BUPT

would simply disappear (i.e. not present at the self-testing stage of the configuration, phase 2, see Figure 3-30) after reloading the molecules with their MOLCODEs (phase 3, see Figure 3-30). Therefore, by simply trying to configure the faulty cell with its MOLCODEs once again, gives the possibility of avoiding the loss of one column of cells; the previously disabled cell will function normally again, forcing therefore another process of coordinate recalculation, which will produce an opposite shifting process than in the case of the *KILL* signal activation [100].

Let us consider such a scenario over the same cell (2,1), which was disabled due to being non-repairable (see Figure 3-25). What happened to this cell was presented in Figure 3-24; there were two faulty molecules detected in the same row, therefore overcoming the self-repair mechanism. If at least one faulty molecule was affected transiently, then re-charging the MOLCODEs into the cell will result in a fully functional cell. In Figure 3-26 we considered that out of two faulty molecules, molecule E has permanent damage and molecule H had transient faults.

The *UNKILL* process effects extend at the cellular level also, where it forces a coordinate re-computation. As a result, the disabled column of cells, which was due to the killing of the (2,1) cell, can be used again. The process is shown in Figure 3-27.

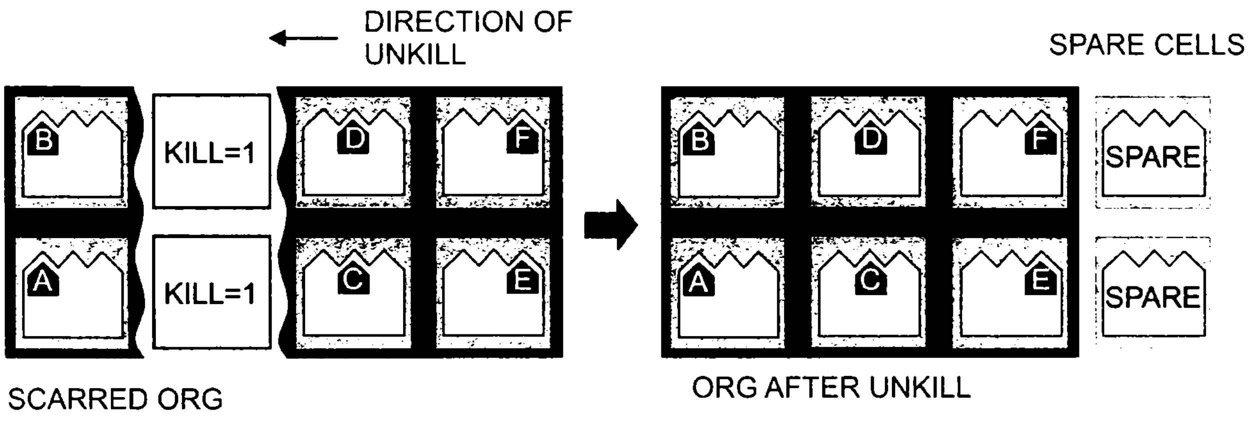


Figure 3-27: The UNKILL process at the cellular level.

## 3.8 Testing the RAM-MuxTreeSR Prototype

### 3.8.1. The Space Divider

The information contained by the MOLCODE defines the logic function, the connections, and/or the memory data of each molecule. To obtain a functional cell, i.e. a complete assembly of MuxTree molecules, two additional pieces of information are required: the physical position of each molecule within a cell has to be defined, as well as the presence and position of the spare columns required by the self-repair mechanism.

#### 3.8.1.1. General Description

The mechanism consists of introducing a regular network of automata (state machines) called *space divider* in the FPGA, which is a variety of cellular automata [8, 47, 54, 123, 129, 130, 133, 154]. Each vertical or horizontal band of the example of Figure 3-28 is an instance of this automaton. Using the space divider, one can divide the entire space of the FPGA into cells of identical size and to specify the position of the spare columns. Figure 3-28 shows an FPGA divided into cells of height 3 and width 3, with one out of every three columns being a spare. The information needed for the molecular configuration is called the *polymerase genome PG*. It can be inferred from Figure 3-28 and consists of a cycle of the following states:

$$PG = C,\ V,\ V,\ H,\ S,\ C,\ldots$$

where C represents a corner, V a vertical band, H an horizontal band, and S an horizontal band associated with a spare column. More generally, let us use the notation {X}·[k] to represent the state (or the sequence of states) X repeated k times. Then, an organism consisting of an array of m cells horizontally and n cells vertically, each cell of height h and width w, will be defined by the following polymerase genome:

$$PG = \{C,\ \{V\}\cdot[h-1],\ \{H\}\cdot[w-1]\}\cdot[m+n],\ C$$

where the presence of spare columns will be indicated by replacing one or more occurrences of H by S [67, 128].

We will detail the components of the space divider, which are binary coded using 3 bits ($COMP_{2:0}$) and presented in Table 3-3:

- vertical band (V): the cellular membrane grows vertically, from bottom to top;
- horizontal band (H): the cellular membrane grows horizontally, from left to right;
- corner (C): the growing process of the cellular membrane splits in two directions, horizontally (to the right) and vertically (upwards);
- spare (S): a special case of horizontal band, defining a column of spare molecules.

| $COMP_{2:0}$ | SPACE DIVIDER'S COMPONENT | |
|---|---|---|
| 100 | Horizontal band | H |
| 101 | Vertical band | V |
| 110 | Spare horizontal band | S |
| 111 | Corner | C |

Table 3-3: The space divider's different components.

### 3.8.1.2. An Example

We will consider the case of two identical unicellular organisms (Fig. 3-29), where the specifications of each organism are those of the modulo-4 up-down counter [100]. In this particular case, m signifies the number of organisms along the horizontal coordinate, and n signifies the number of organisms along the vertical coordinate (i.e. m=2 and n=1). The ribosomic genome *RG* is described in Subsection 3.5.1.2 (Fig. 3-12C). In the actual implementation, we add a spare column at the right of each organism.

The mechanism of inferring the polymerase genome was described previously in Subsection 3.8.1.1. In our case we have two organisms (m=2, n=1), each being constructed with a single cell (w=3, h=3). The polymerase genome *PG* has the following form:

$$PG=\{C,\ \{V\}\cdot[2],\ H,S\}\cdot[3],\ C$$

which is exactly the example shown in Figure 3-28. After coding each component of the polymerase genome *PG* as shown in Table 3-3, the binary coded *PG* results as *PG₁*:

$$PG_1=\{111,\ \{101\}\cdot[2],\ 100,\ 110\}\cdot[3],\ 111$$

The polymerase genome *PG* and the ribosomic genome *RG* configure the array of MuxTree® molecules at configuration time. For this reason, *PG* and *RG* are binary coded inside the EPROM memory of a state machine (the *configuration sender*) whose only role is to send them toward the array. Due to implementation reasons, the polymerase genome *PG* has to be expressed in hexadecimal code. The entire process of properly coding *PG* is described as follows. First, *PG₁* must be reversed, the result being *PG₂*:

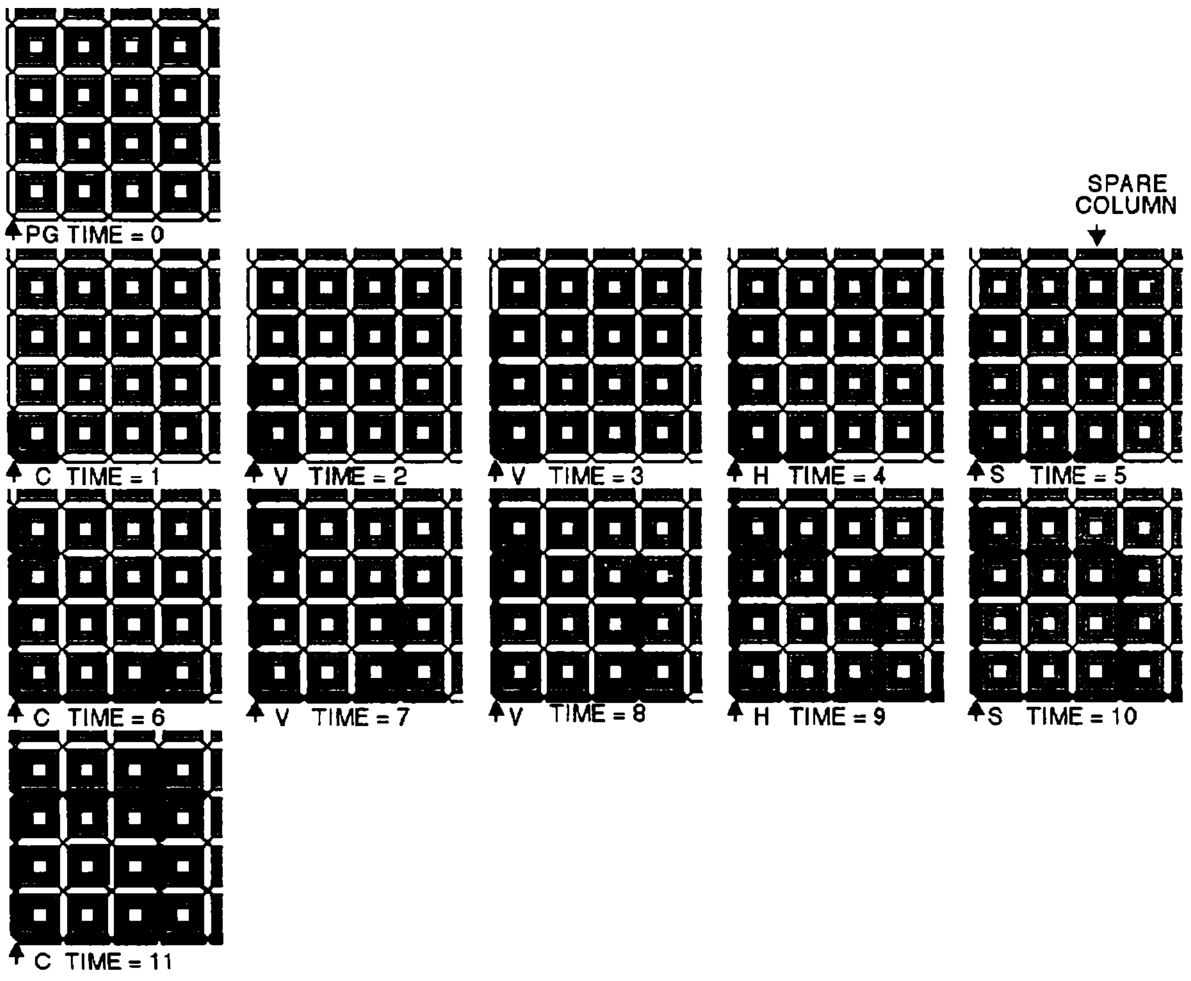


Figure 3-28: Example of a space divider (height=3, width=3, 1 spare column out of 3); PG: polymerase genome: C, V, V, H, S, C, ...

$$PG_2=111,\ \{011,\ 001,\ 101\cdot[2],\ 111\}\cdot[3]$$

To transform *PG₂* into hexadecimal EPROM-ready code, due to implementation reasons, we have to add a zero to the rightmost position, which is also the least significant bit. Unrolling $PG_2$ and adding the necessary zero to the least significant position gives us $PG_3$:

$PG_3$ = *111 011 001 101 101 111 011 001 101 101 111 011 001 101 101 111 0*

Since the direction from the least significant bit to the most significant bit is from right to left, the hexadecimal coded and EPROM-ready polymerase genome ($PG_4$) becomes:

$PG_4$ = *0001D9B7 B36F66DE*

The EPROM stores words worth 8 hexadecimal characters of data, so we need to split $PG_4$ in words of this length. The least significant word is stored at the lower address inside the EPROM. In our case, the EPROM containing $PG_4$ is presented in Table 3-4.

| EPROM ADDRESS | PG DATA |
|---|---|
| 0 | B36F66DE |
| 1 | 0001D9B7 |

Table 3-4: The EPROM-ready polymerase genome PG.

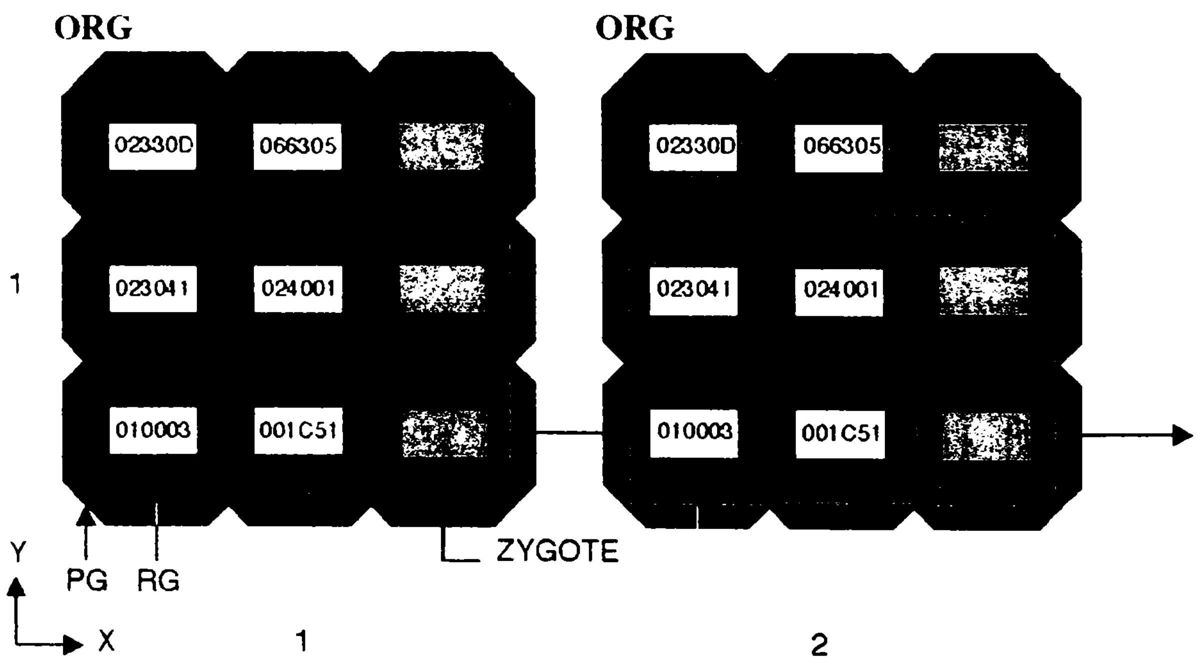


Figure 3-29: Two identical, unicellular organisms, each implementing the modulo-4 up-down counter.

The way of calculating *RG* was presented in Subsection 3.5.1.2. The complete EPROM for our example in Figure 3-29, containing both the polymerase genome *PG* and the ribosomic genome *RG*, is shown in Table 3-5.

## 3.8.2. Prototype Configuration

Testing the RAM-MuxTreeSR prototype followed two directions: first, to ensure that the functionality of the preserved mechanisms [128] was not affected by the transfer and second, that the new macro-molecular structures are well integrated within the design. The prototype consisted of 18 molecules, making up two unicellular organisms, each using a single column of spare molecules (a similar set-up is shown in Figure 3-29). The complete binary configuration is given by Table 3-5.

Let us review an organism's life cycle, starting from power-up (shown in Figure 3-30):

1. Space dividing phase: the polymerase genome *PG* is passed through the area of molecules and delimits the cells (Figure 3-28 and Table 3-5). At the moment, the space divider is not under test. The next phase is 2.
2. CREG registers testing phase: the CREG register of each molecule is tested using a *test pattern* (30000080 in Table 3-5) [128]. If no faulty registers are detected, the next phase is 3. If yes, the next phase is 5a.
3. Configuration phase: each molecule receives its ribosomic (*RG*) and/or operative (*OG*) genomes, the order the molecules are loaded with data being shown in Figure 3-29 and Table 3-5. The next phase is 4.
4. Normal operating phase: cells are functioning normally. A fault-detection mechanism will signal the errors that might occur inside each molecule. When errors are detected, the next phase is 5b, otherwise 4.
5. One or several faulty molecules are detected. If there are enough spare molecules, the next phase is 6 (a or b). If not, the next phase is 7 (a or b).
6. The self-repair process takes place at the molecular level as shown in Fig. 4-28:

   6a. The next phase is 3, i.e. the configuration phase.
   6b. The next phase is 4, i.e. the operating phase.
7. The self-repair process takes place at the higher, cellular level. There are two cases:
   i) a faulty molecule is detected in a row and there is no spare molecule available;
   ii) a faulty molecule is detected and there is only one spare molecule left in the row, but the spare molecule is itself faulty.

   In both cases the self-repair mechanism will be outrun and the cell will die, generating the *KILL* signal. This will lead to the reconfiguration at the higher, cellular level, as previously described.

   7b. The next phase is 3, i.e. the configuration phase.
   7c. The next phase is 4, i.e. the operating phase.

Testing the prototype allowed us to verify the following:

- the implementation of the new, macro-molecular structures was successful, therefore enabling a much needed, flexible storage capability. Each of the memory sub-modes, as well as the logic modes behaved as expected;
- the *HOLD* mechanism was tested to be fully functional when multiple, independent macro-molecules were employed;
- the original mechanisms of self-testing and self-repairing were preserved with their functionality unaffected when operating in logic mode. When in memory mode they only offer a partial functionality, due to the lack of a fault-detection strategy for this mode;
- the *KILL* mechanism was preserved and extended with its reverse-corresponding mechanism, called *UNKILL*; the *UNKILL* mechanism was successfully tested;

| EPROM ADDRESS | DATA | MEANING |
|---|---|---|
| 0 | B36F66DE | Polymerase genome PG (Table 4) |
| 1 | 0001D9B7 | |
| 2 | 00000000 | Nil pattern |
| 3 | 30000080 | CREG register test pattern |
| 4 | 00000000 | Nil pattern |
| 5 | 01000300 | Ribosomic RG and/or operative OG genomes interleaved with nil patterns (Fig. 5c) |
| 6 | 00000000 | |
| 7 | 02304100 | |
| 8 | 00000000 | |
| 9 | 02330D00 | |
| A | 00000000 | |
| B | 001C5100 | |
| C | 00000000 | |
| D | 02400100 | |
| E | 00000000 | |
| F | 06630500 | |

Table 3-5: EPROM for configuring the two identical unicellular organisms.

However, there are some issues that remain to be settled at this point:

- the original self-testing mechanism preserved from the MuxTreeSR design, cannot be just simply extended to obtain a self-testable macro-molecular structure;
- an original fault-detection strategy has to be developed in order to expand the robustness of the Embryonics concepts over the macro-molecules, as this may be both an essential strength and weakness of such a system

The motivations for these issues, as well as the solutions adopted for the implementation will be presented in the next chapter.

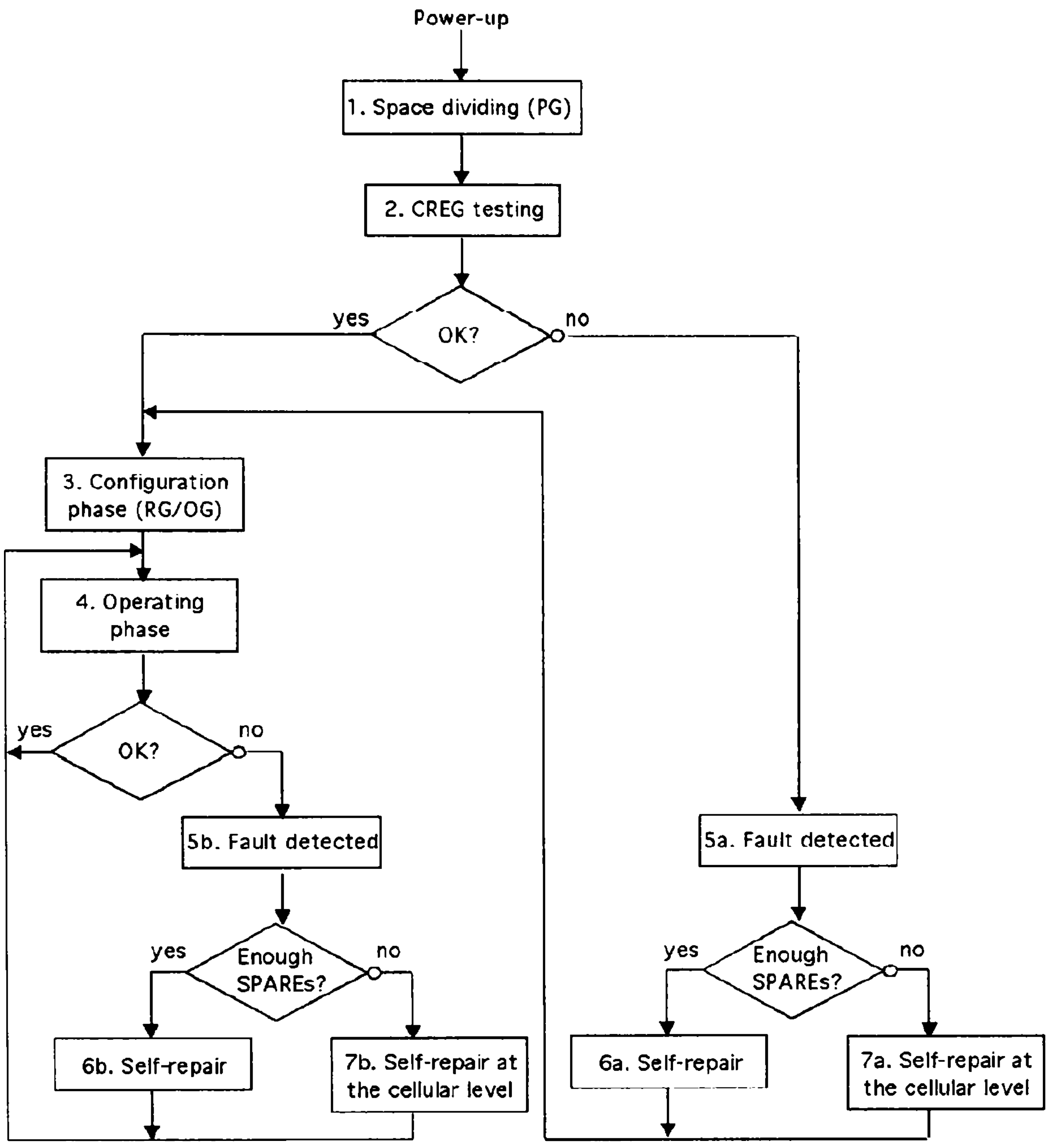


Figure 3-30: Block schematic of an organism's life-cycle.

# CHAPTER 4

# FAULT-TOLERANT MACRO-MOLECULES

## 4.1. Introduction

### 4.1.1 Bio-Inspired Storage

As pointed out previously, a macro-molecule in Embryonics is a memory structure, essentially a block of molecules operating in the same mode (i.e. the *memory mode*) [100, 102]; they are separated from the surrounding molecules operating in the logic mode or from other memory areas, by the so-called *memory membrane*, an abstract formation coded inside the *ribosomic genome* under the form of MOLCODE bits 21-20 and 3-1 [100]. The macro-molecule therefore builds a large storage zone by effectively chaining together smaller, identical memory units found in each molecule under the form of CREG. Thus, Embryonics allows flexible memory configurations, not unlike modern computer systems, which use memories built from many VLSI RAM chips. Considering the expected time of failure for a single memory chip, whenever many such chips are assembled to form a single, larger memory, the corresponding period of wait until at least one chip fails becomes significantly smaller [5]. In fact, in real terms this period can be as low as a few hours, thus making any memory protection become mandatory. For this reason, most computer memories make use of SEC-DED (single error correcting, double error detecting) codes [27, 37, 104], which greatly enhance the overall reliability.

The Embryonics project focuses on building a bio-inspired computing system that would exhibit (among others) superior, biological-like fault tolerance. Because building blocks in Embryonics are mainly a matter of hardware configuration, each and every basic brick is already endowed with capabilities of hardware fault tolerance. However, in achieving a higher complexity and flexibility, the hardware itself, which is reconfigurable, has to be driven by a form of software, with fault tolerance techniques comprised not only at the *hardware level* (functional unit) but also at *the software level* (memory unit). Due to the fact that the introduction of memory modes in Embryonics is part of its natural development, no memory protection mechanisms have yet been implemented, other then a somewhat limited approach of doubled data storage [101]. A reliability analysis of Embryonics structures has been made [84, 85], but it only concerns structures operating in logic mode. The newly added operating mode for the molecules (the memory mode) affects the functionality in Embryonics by extending it and therefore an updated reliability analysis is much needed.

The existing two levels of self-repair were, however, preserved unaffected by design progress, but they are essentially protecting the *functionality* of the whole (that is, the ribosomic genome); they cannot protect the integrity of the *genome* that governs over the system's functionality (i.e. the operative genome and, of course, the macro-molecule). Furthermore, while designing bio-inspired computing machines is quite ambitious, the key aspect remains: because of current technological limitations one cannot go above *inspiration*. This is to say that fundamental, digital design techniques can only be adapted, albeit in an original and biological-like manner, to build innovative systems, they cannot be transferred directly from biology into digital systems.

### 4.1.2 Fault, Error, Failure

Computers are fine exponents of the present days' technological wave. Though solidly set on the road of evolution, computers experience difficulties that arise from this road becoming manifold: some applications require speed above anything else, while others require the highest possible reliability. Unfortunately, computers themselves can fully fulfill none of the requests, fueling the need for different and better-suited designs, and therefore justifying a quest for new inspiration in both their hardware and software designs.

Since their beginning, computers were protagonists of the quest for performance. Benefits of computing power and technological advances enabled the coming of the space exploration era, which in turn encouraged a shifting in priorities for achieving performance from brute computing force (which appears to have reached somewhat sufficient levels today, for most applications) towards the advent of dependable computers. While computers are generally designed from the very beginning as remarkably reliable machines, they are bound to experience various kinds of errors in their lifetime due to often neglected factors such as atmospherics, electrical noise, component malfunctions or even design or programming flaws [104], as illustrated in Figure 4-1.

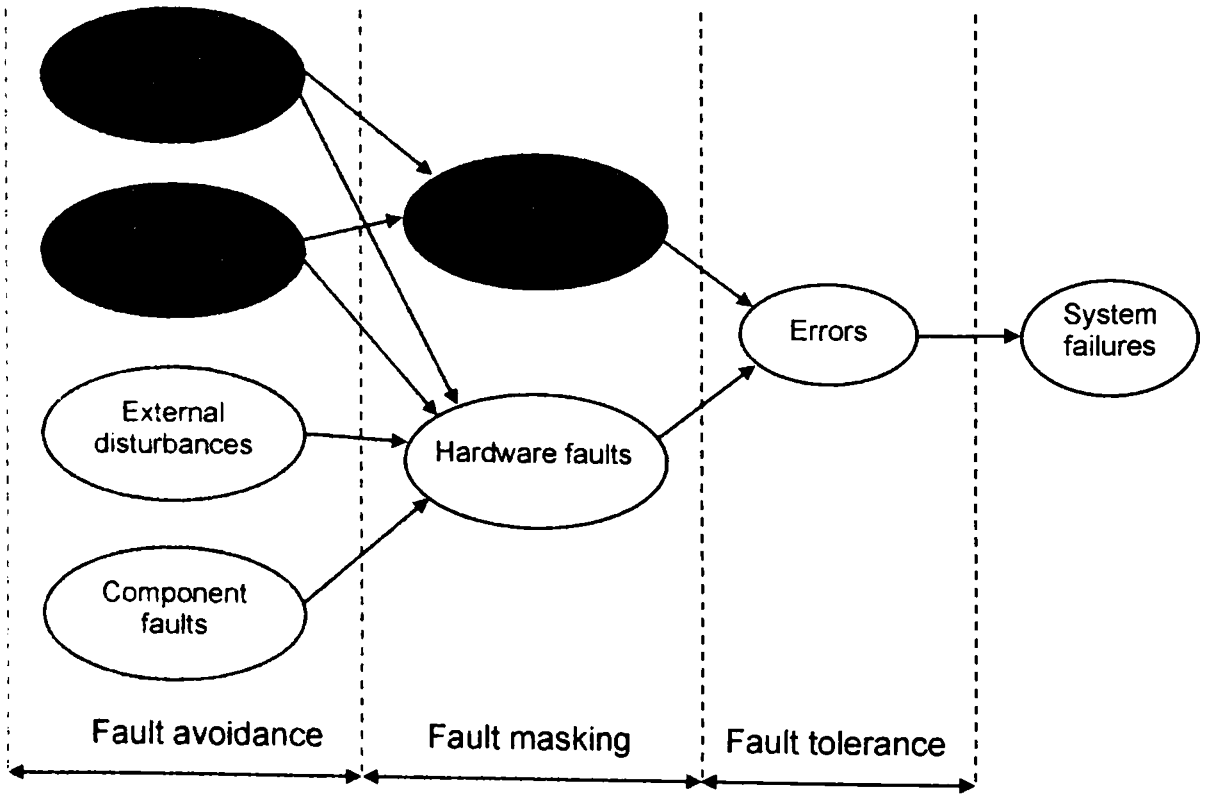


Figure 4-1: Fault-error-failure causal model [39].

There are several factors leading to inadequate operation of a computing system; generally, these can be avoided (*fault avoiding*) through rigorous quality assurance during each stage [39]:

- Specification mistakes happen in the early stages of the design process due to vague formulations or even misconceptions related to the system objectives.
- Implementation mistakes. After the specification process reaches maturity it is followed by the implementation, that is, the transformations from hardware and software specifications into operating equipment and algorithms.
- External disturbances are due to environmental or artificial factors such as radiations, electromagnetic interferences, operator mistakes.
- Component faults arise at random during normal operating time due to age, wear and tear.

Faults can be classified over broad criteria but their nature (hardware or software) and time behavior (permanent, transient and intermittent) are essential. Software faults can usually be recovered from by extensive use of error detecting and correcting codes [104] while hardware faults usually require system reconfiguration and/or human intervention. The techniques used to achieve error tolerance help a system operate normally even in the presence of errors.

In Figure 4-1 factors that are not taken into account by Embryonics are shown in grey. This is simply a matter of particularities of the project, which is, by all means, not a piece of software; therefore the dashed conditions simply do not apply. Concerning hardware mistakes that can be made, those that might appear during specification and implementation stages are avoided during the initial testing process, thus making an operating embryonic structure vulnerable to either component faults or external disturbances [45]. While component faults could be ruled out because of the extensive initial test sequence employed by the Embryonics project, followed by the two-level reconfiguration process, external disturbances can potentially upset the normal behavior of the system for no apparent reason.

With respect to their behavior in time, faults fall into the following categories [27]:

- permanent faults: their effect is due to a fault affecting the normal operating of the device constantly and over an indefinite period of time, i.e. permanently. There exists a broad literature dedicated to coverage and modeling of permanent faults [27, 39]. They are also considered under the terms of *solid* or *hard fails*.
- non-permanent faults: they occur randomly and affect the system's functional behavior over indefinite, but finite, periods of time. Because the moments of occurrence show variable frequency and duration, these faults are very difficult to be modeled. There are two kinds of non-permanent faults: transient and intermittent.

Transient faults originate as an effect of environmental conditions such as atmospheric parameters (temperature, humidity, pressure), supply characteristics (fluctuating power, noise, crosstalk), pollution, and cosmic rays and particles. Because the system is affected even when no identification of permanent faults is possible, there is no model available for such faults, often referred to as *soft fails* or *soft errors*. Intermittent faults are caused by various factors, other than environmental: component parameter variations, timing, loose connections, etc; they typically affect the system over very short time intervals, thus making any debugging process more difficult. By using accelerated testing under stress conditions it is possible to make non-permanent faults become permanent, which can then be modeled [45].

### 4.1.3 Inspiration Toward Achieving Dependability

Dependability can be defined as "*the ability of a system to avoid service failures that are more frequent or more severe than is acceptable*" [3]. It is therefore a synthetic term that involves a list of attributes including reliability, fault tolerance, availability, and others. In real world, a dependable system would have to operate normally over long periods of time before experiencing any fail (reliability, availability) and to recover quickly from errors (fault tolerance). The term "*acceptable*" has an essential meaning within the dependability's definition, setting the upper limits of the damage that can be supported by the system while still remaining functional. It is therefore worth elaborating upon ways to compute the threshold beyond which the damaging effects become no longer acceptable. All of the above being considered, dependable systems are crucial for applications that prohibit or limit human interventions, such as long-term exposure to aggressive (even hostile) environments. The best examples are long-term operating machines as required to manage deep-underwater/nuclear activities and for space exploration.

The quest of building digital systems that offer superior dependability can draw benefits from two distinct sources [97]. The first one, already mentioned, is the oldest and most complex computing system, which has been around since the dawn of times: Nature. Its living elements continuously demonstrate a variety of solutions for achieving robustness in an error-prone, macro-scale environment. There are numerous similarities and differences between artificial, digital computing systems and living beings; although such a thorough analysis is beyond the scope of this thesis, a synthetic view is highlighted by Table 4-1 [97]. However, it is likely that the field of digital computing could exploit some of the mechanisms implemented by Nature and adapt them to the electronic environment, a representative example being the Embryonics project.

Another source of inspiration may be constituted by novel computing paradigms, whose research already considered dependability-raising techniques. The emerging field of quantum computing relies on successful calculus in an error-prone, micro-scale environment. Since errors are (as of yet) intrinsic to quantum systems [149], a number of techniques were established in order to overcome their damaging effects and deliver consistent results. However, though a variety of methodologies for estimating dependability parameters have been proposed, they usually remain localized within their original field and rarely reach other architectures. Insights to the fields of quantum [148, 149] and bio-inspired computing will be provided next in order to argue upon the benefits that could be drawn by importing a methodology of estimating the computing accuracy threshold from quantum computing to the Embryonics project, together with additional means of further increasing dependability levels in Embryonics systems.

| Artificial systems | Natural systems |
|---|---|
| No unified strategies concerning redundant resources | Unified strategies via hierarchical redundancy |
| Capabilities empirically tested (through functional stress and accelerated aging) in short periods of time | Capabilities tested in long periods of time |
| Disputable maturity status, notions not yet standardized [3] | Maturity endorsed by the variety of species, proof for successful biological processes |
| Short design periods | Long design periods, not useful to attain by human processes |

Table 4-1. Comparison between types of dependable systems

### 4.1.4 Self-Repair in RAM-MuxTreeSR

The previous sections (4.1.1 and 4.1.2) may now be put together in order to provide a more detailed perspective about the self-repair capabilities of our new design. Of course, by preserving the previous self-repair strategies and mechanisms, the robustness degree of the molecules operating in what is now called the logic mode remains unchanged. There are several strategies that concurrently work in order to achieve fault-tolerance [128]:

- The configuration register CREG is tested at system start-up for detecting any stuck-at type faults that may exist (Figure 3-30, phase 2). Once the system is up and running, there is no mechanism that would detect a change in the binary pattern stored by CREG.
- The functional unit FU is tested through resource replication and majority voting. Correct data is ensured by the voting mechanism, while the resource replication strategy allows the detection of possible errors.
- The switch block SB is not tested by itself.
- Once an error is detected, the reconfiguration mechanism is triggered hierarchically, first at the molecular level and then at the cellular level;

All of the above considered, one can say that any system in Embryonics is well protected against permanent faults (or stuck-at type), any error detected in either the CREG or the FU triggering the self-repairing strategies (except for the situation when the error first affects the CREG).

However, transient and intermittent faults were not considered initially, as too expensive to fix, since the CREG was not supposed to modify its value during operation; from this perspective, the FU is better adapted to endure these kinds of faults. Moreover, intermittent faults are most difficult to repair and therefore a strategy that would include them would also involve an important (and possibly quite complex) hardware overhead. If any strategy involving the recovery from intermittent faults is to be implemented, one needs to establish first the likelihood of such a scenario, which will be argued in the next section.

In order to assess the importance of the genome memory, it is worth remembering that the genetic program is not just plain data that can be read and written randomly; instead it is intended as a large program, ideally written once and then read and executed for indefinite periods of time. The existence of the macro-molecules solves the storage issue required by the genome memory;

however, for a clear understanding of the interaction between the preserved self-repairing mechanisms and the new macro-molecular structures, we feel that more insight into the matter is necessary.

When an error is detected at the molecular level, the reconfiguration mechanism is triggered, the faulty molecule is marked as being "dead" and a spare molecule takes its place; but one should not forget that, originally, all was designed from a functional point of view.

Let us suppose the same reconfiguration process would be triggered should an error be detected inside a molecule operating in any of the memory modes. This would mean the "death" of the respective molecule and the transfer of its role toward a spare one, which will then become active. But because of transferring genetic material from the faulty molecule to a spare one, the final result is that the faulty molecule "dies" and the spare one starts behaving *the same* as the faulty molecule before being "killed". Instead of recovering from the fault, the final outcome is a very similar situation to the initial one, but this time with the spare resources reduced by a molecule.

These characteristics point out an architectural vulnerability in the current RAM-MuxTreeSR design: a potential error in the genetic program can neither be corrected, nor even be detected. Furthermore, the areas where highly dependable systems may be successfully used in (and Embryonics is no exception), such as space exploration, involve potential exploiters of this particular vulnerability; this could have most serious effects since the genome either drives actual hardware (polymerase and ribosomic genome) or contains instructions on how additional hardware will be driven (operative genome).

Considering all of the above arguments it can be concluded that:

- currently, Embryonics has no memory protection mechanisms implemented;
- such implementations are both desirable and feasible; with the human expansion into space having a most upsetting potential with respect to electronic devices, they should run on top of other (typically radiation shielding) techniques.

## 4.2. Single Event Upsets: An Analysis

A convenient and effective way of fault modeling is to consider a software error as simply affecting a binary value; then a transient fault is nothing else but a bit that changes value for no apparent reason due to entering a variety of possible transient regimes triggered by external disturbances. In fact, bits of information may be regarded as small amounts of electric charge, moving inside electronic devices. By quantifying the value of the charges one can differentiate between them and refer to as the two symbols of information, bits 0 and 1. Because we represent information by means of electrical charges this makes electronic devices vulnerable to whatever conditions might interfere and modify these charges, the result being erroneous data. A remarkable effort is made during the design process to decrease the sensitivity electronic/semiconductor devices may show towards such upsetting phenomena.

During their life cycle, electronic devices constantly suffer a number of influences that manifest predominantly over transient regimes. A special case is the category of digital electronics, which deals with transient phenomena due to changing states and logical values. Such phenomena introduce a variety of errors unified in the literature under the name of *transient faults*, *soft errors* or *single event upsets* (SEUs). The rate electronic devices are affected with is known under the term of *soft error rate* or simply *SER* and is measured in fails per unit time [38]. No matter the name they are referred under, these errors affect the normal execution process and are due to electromagnetic noise and/or external radiations rather than design or manufacturing flaws [113, 114].

A key issue concerning bits of information and their associated electric charges is listing the conditions that might lead to such disturbances. Some may be eliminated by carefully applied technology, but some necessitate design changes or may even prove impossible to prevent. Many experts consider that soft fails are mostly caused by electronic noise [127, 163]; it seems that potential candidates generating such noise are energetic nuclear particles, which are responsible of inducing ionization electron–hole pairs, and fall into three categories:

- $\alpha$–particles are born as part of radioactive decay processes;
- radioactive isotopes, widely used for a range of purposes, might contaminate semiconductor materials leading to soft errors; evidence regarding this was given in a famous paper by May and Woods of Intel [75] and confirmation came by irradiating electronic chips [28, 29, 156];
- cosmic rays, containing a broad range of subatomic particles.

The fact that radiation can affect electronic circuits has been long known, memory and logic upset behaviors being observed in satellite electronics. In 1978 Ziegler of IBM realized that if $\alpha$–particles affect semiconductor devices causing soft fails, then there must be a fair probability that other particles coming from the outer space under the form of various cosmic rays might cause the same thing. Causes were attributed to energetic heavy ions in the solar wind [166]. Since the early 1980s has been known and studied the influence and capacity of cosmic radiation and of a broad range of particles to influence semiconductor circuits, leading to transient faults or soft errors.

Typical modern digital devices incorporate techniques for recovering from these errors, such as error detecting and correcting codes, various parity-check schemes and others; however, these mechanisms focus on the devices prone to influences from such errors, i.e. memory elements. As pointed out [113, 114] there are several key reasons for protecting memories:

- techniques involved are well understood and relatively easy to implement, thus being quite inexpensive;
- memory area in each computing system is proportionally significant;
- research conducted so far shows that combinational logic is much less susceptible to soft errors than sequential logic [22, 51].

In order to model the frequency of soft errors it was necessary to further investigate how exactly they were born and what dependencies may exist between soft errors and various particles and beams that have either a cosmic origin (heavy energetic ions) or are generated on earth (other particles). They contribute to the overall SER independently and are therefore additive. The soft error rate (SER) does not appear as constant; as technological advancement leads to shrinkage of electronic devices, they become more vulnerable to soft errors caused by lower energy radiation.

### 4.2.1 Radioactive Isotopes

Radioactive contamination inside or in the proximity of semiconductor production facilities can cause an unusually high SER. It is worth mentioning what is historically known as the "Hera problem". At the beginning of 1987 LSI memories produced by IBM were hit by anomalous behavior, which led to erroneous data. Intriguingly, only batches produced in the United States seemed to be affected while those produced in Europe showed no sign of similar failures.

There were reasons to believe that the packaging was responsible for all the problems but soon this was ruled out, as exchanging packages did not affect the incidence of reported problems, thus indicating there was a problem in the chips themselves. After intense efforts the conclusion was the discovery of traces of radioactivity; however, it was difficult, if not impossible, to establish where those radioactive traces came from, but the type of the chemical trouble-producing element was certain: polonium ($Po^{210}$). This particular radioactive isotope is a product of the uranium decay chain, but strangely, the other expected particles were totally missing. After months of searching for the contamination source, it was finally identified under the form of one bottle of nitric acid, used in the process of manufacturing the chips that showed signs of contamination. This was the successful end of the "Hera problem" [163].

Intel was also hit by problems with their 2107 series of 16Kb DRAM memories [13], the cause being radioactive contamination. The reason was later found to be the factory being built downstream in close vicinity of an old uranium mine.

### 4.2.2 Cosmic Ray Influence

Any ionizing particle that hits an electronic device induces strong perturbations in the electromagnetic field of component elements. Since electronic devices rely on the properties of p-n junctions, the disturbances created translate into electron–hole pairs, that is, a random signal. If the local electromagnetic fields are sufficiently strong then the pairs cannot recombine; instead, they will find a way out to the nearest appropriate device contact and the corresponding charge collection could provoke a soft error. Disturbances are also created when cosmic ray particles are involved in collisions with semiconductor nuclei, the result being a wide range of secondary nuclear fragments generated in an avalanche-type phenomena: nucleons, pions, light ions such as deuteron ($^2H$), triton ($^3H$) and helium ($^3He$ and $\alpha$–particles), and heavy residual nuclei such as oxygen, carbon, and magnesium.

As particles that compose these rays come from outer space and enter the earth's atmosphere, there is a chance of hitting other particles, resulting in two types of collisions: elastic and inelastic. As a result of these collisions an entire flux of high-energy particles is born, reaching the earth's surface. Several factors affect the distributions of the final flux [114]:

- **Altitude:** Due to the filtering effect of the atmosphere, the lower the altitude, the smaller rate of particles. For instance, the flux at an altitude of 3100m (Leadville, CO) is approximately 13 times greater than at sea level.
- **Geomagnetic region (GMR):** Earth generates a magnetic field that also has a shielding effect. Therefore cosmic rays show the smallest penetration around the equator and the largest at the poles. The measures of this effect range from 1.0 GV near the poles to 17 GV at the equator.

- **Solar cycle:** The 11-year solar cycle strongly affects the particle flux, Earth being under a constant bombardment of particles coming from the sun and associated phenomena. Intriguingly, the sun's activity has an opposite effect than common belief might consider: as the activity of the sun increases a lower rate of particles can be witnessed, even if solar flares can increase the particle rate. In fact, periods of active sun see up to a 30% lower rate of particles compared to periods of quiet sun [114]. The explanation lies in the fact that the magnetic field around the earth strengthens during periods of active sun, increasing its shielding effect and thus reducing the penetration of cosmic rays. The combined filtering greatly affects the energetic particle flux, particles with highest energies being least represented in the total flux (neutron graph given in Figure 4-2).

During the 1980s a series of experiments were made by IBM as an attempt to measure the particle flux from cosmic rays [162]. For a particular energy, this can be expressed as the number of particles having that energy, hitting the planet per unit surface per unit time. Obviously, not any kind of radiation/particle would provoke a soft error, and they also have an uneven distribution. In particular, higher energy particles are more susceptible to upset semiconductor devices but also less likely to occur while lower energy particles occur more frequently but do less or no damage.

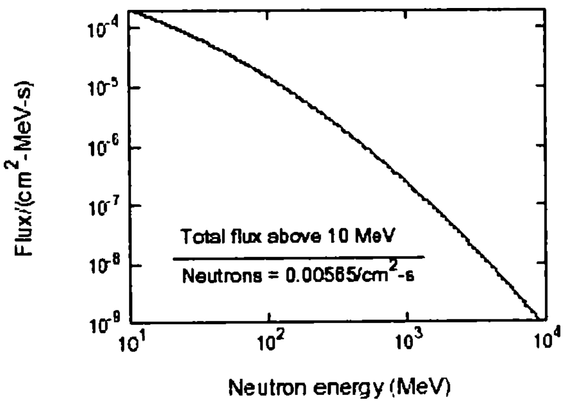


Figure 4-2: Neutron flux plotted against energy [162].

The term "cosmic rays" offers a generous accommodation, since it has no precise scientific definition; the general line is that cosmic rays come from outer space and correspond to a broad range of energetic particles. By now there is a separation between at least three categories [162]:

1. **Primary cosmic rays** are particles lurking in the universe and solar system that eventually may hit our planet. It is common belief that they are a product of intense reactions taking place in the universe, including those generated by the sun under the form of the solar wind, and are composed of three categories of particles. Protons account for as much as 92%, the rest being $\alpha$–particles (6%) and some heavy atomic nuclei [48, 121, 152]. The dynamics of the ray composition follow that of the galaxy, meaning that the particle movement is affected by the background magnetic field and by the spiral spinning.
   Our sun is also a major source of primary cosmic rays, releasing particles with energies much lower that those coming from the galaxy. It also has a cyclic activity, with an 11-year period, that drastically influences the overall

quantity of generated particles. If during periods of quiet sun the particles from the solar wind that hit able the atmosphere are virtually not capable of penetrating it, their numbers increase by a million times during periods of active sun, the sun-generated flux becoming greater than that corresponding to the intra/inter-galactic flux. The measured flux of primary cosmic rays is about $10^5/m^2 \cdot s$.

2. **Secondary cosmic rays** are particles born by collisions when primary cosmic rays enter the earth's atmosphere. These are also known under the term of "cascade particles" (Figure 4-3A).
3. **Terrestrial cosmic rays** are particles that actually reach the sea level. Of the particles that reach the surface only an estimated maximum of 1% originate from primary cosmic rays, the rest being cascade-generated particles, usually from the 3rd to 7th generations. Complex phenomena appearing during periods of active sun lead to an increase of up to 200% in intensity of cosmic rays reaching the surface of the planet. In fact, during high activity periods our sun generates the solar wind, carrying much more particles (up to $10^6$ times more). The solar wind also creates a very strong magnetic field that acts as an effective shielding against intra-galactic cosmic rays. Therefore, while the sun generates many times more particles, the magnetic shield created simultaneously might actually reduce the final flux at sea level by about 30%. Considering that the solar cycle has a periodicity of 11 years, the extremes in sea-level particle flux lag by 1-2 years.

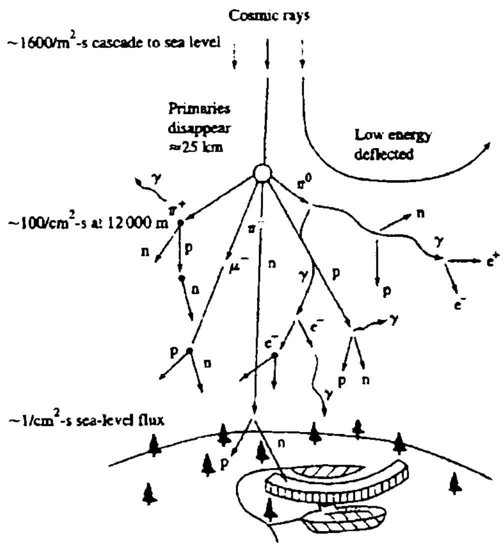
A large number of parameters of the atmosphere change with altitude and therefore an analysis of the presence and quantity of cosmic rays in the atmosphere cannot be done without considering it. The atmosphere's density is $1033g/cm^2$ at sea level, altitude being measured in physics the same as barometric pressure is. The density of the atmosphere causes particles with strong nuclear interaction, called *hadrons*, to suffer collisions before they hit the surface. The number and intensity of these collisions literally create a cascade of particle showers (Figure 4-3A), preventing the original primary particles from reaching the surface; instead, the particles that result from the collisions propagate the cascading effect until later generations finally reach the sea level. The composition of the resulting flux is hadrons (nucleons and pions), leptons (muons and electrons) and photons and is measured as about $360/m^2 \cdot s$.

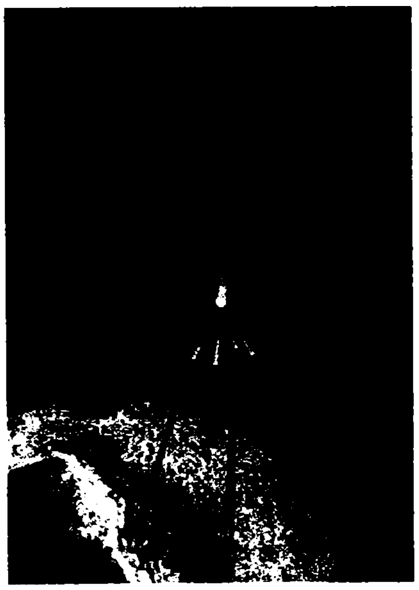
At the time a particle enters the atmosphere it behaves differently depending on its physical properties. Hadrons lose their energy very quickly to atmospheric nuclei; charged particles lose energy to atmospheric electrons; the least deflected are the heavier particles while the most deflected are the lighter ones. Most of the particles decay spontaneously or reach thermal energies making atmospheric absorption also play a key role. The lifetime of a certain particle is also an important factor, pions and muons being unstable particles, with a lifetime of a few nanoseconds and about a microsecond, respectively. Measurements indicate that the peak of cascading density occurs at an altitude of about 15 km, an altitude usually referred to as the Pfotzer point [162]. This is also the altitude many commercial aircraft use, and this is where *the fail rate of electronic devices is about 100 times worse than at terrestrial altitudes.*

When protons hits the earth's upper atmosphere (Figure 4-3B), most of the resulting particle fragments decay or are absorbed by it. However, muons have a half-life time of about 1.4 microseconds, which is just long enough that some reach the earth's surface. Table 4-2 gives a calculation of what kind of particles the total flux is composed of. Muons, which are considered as heavy electrons, appear to dominate the medium/high energy particle spectrum. Proton flux is largely affected by interactions with atmospheric electrons and pions have a too short life to reach the surface level. Finally, the particle flux reaching sea level with energies below 100MeV is very sensitive to local environments such as buildings [162]. The energetic levels that seem to be of most concern for SER are between 200 and 3000MeV.

| Particle type | Total flux $cm^2$ / year |
|---|---|
| Muons | 65466 |
| Neutrons | 44812 |
| Pions | 48.4 |
| Protons | 360 |

Table 4-2: Theoretical calculation for sea-level particle flux at New York City [162].


Cosmic rays
~1600/m$^2$-s cascade to sea level
Primaries disappear ≈25 km
Low energy deflected
~100/cm$^2$-s at 12 000 m
~1/cm$^2$-s sea-level flux
γ
π⁺
π⁰
π⁻
μ⁻
n
p
e⁺
e⁻



A. B.

Figure 4-3: A. Cosmic rays cascade phenomena [163] and B. showers [172].

### 4.2.3 Modeling Cosmic Ray Influence

A significant effort towards particle measurements has been done until now, but it is quite difficult to count and measure the physical properties for all particles that may be potentially involved in creating soft errors; a vast effort is however ongoing in order to model phenomena involved. CREME96 (from Cosmic Ray Effects on Micro Electronics Code) and NUSPA (NUclear SPAllation) are some of the code collections used to simulate and predict the effect that cosmic ray particles might induce over semiconductor materials.

NUSPA's initial focus on sea-level interactions (where protons and neutrons play the most significant role) was extended to model interactions at high altitudes, including pion interactions. The energy exchange between particles at the atomic level is determined by the probability of colliding (elastic or inelastic scattering) with a target nucleus for each incoming particle. The incoming particle only affects the target nucleus with a small amount of energy (in an elastic scattering) while its trajectory is deflected to a small angle; the nucleus will recoil but will not become excited, i.e. will not change its original energy state. During inelastic scatterings large amounts of energy are exchanged, resulting secondary particles and an excited intermediate nucleus that will transform into a stable and lighter nucleus with additional particle emissions. These show a greater ionization potential than the initial colliding cosmic ray particle and are quite effective in producing soft errors [127].

Elastic and inelastic scatterings do not show the same potential of harming semiconductors; in fact, elastic scattering induced soft errors are considered to be negligible for the moment, until the technological advancement will decrease the critical charge to the order of 50fC. An example of inelastic scattering is discussed in Section 4.2.6.

### 4.2.4 Introduction to Particle Physics

For the clarity of SER phenomena, a deeper investigation on how particles presented in Table 4-2 interact is necessary. All particles known to man are made of quarks and leptons (which are believed to be point-like particles [127]). This is why they are considered as elementary particles, i.e. the basic building blocks of matter. The electrons, *muons*, tau particles, and their associated neutrinos are all situated in this category.

Hadrons are particles that interact by the nuclear strong interaction, composed of quarks, either as quark-antiquark pairs (mesons) or as three quarks (baryons). This classification specifically excludes leptons, which do not interact by the strong force. Baryons are massive particles built of three quarks in the standard model; they include, among other particles, the *protons* and the *neutrons*. Mesons are composed of only two quarks; they include the *pions* among other particles.

At sea-level approximate 94% of the total flux of hadrons is represented by neutrons (Table 4-4). Being uncharged particles, usually neutrons go through electrical circuits of electronic devices without any interactions. Unless combined into a nucleus, a free neutron follows a $\beta$-decay process with a half-life of about 10.3 minutes with a proton, an electron and its corresponding neutrino being released, as described by equation (1):

$$n \rightarrow p + e^- + \bar{\nu}_e \qquad (1)$$

Statistically, only a small number of neutrons (about 1 out of $4 \cdot 10^4$) eventually hit a silicon nucleus, but then the hit is very likely to produce a soft fail. Considering energies above 20MeV, at sea level, about $10^5$ neutrons/cm$^2$·year can be counted. When estimating the probability of an interaction between an energetic neutron and a silicon nucleus one must also estimate a variety of parameters such as total absorption cross section, active atoms per chip, and active thickness of the circuit. If the active thickness of the circuit is 1um then almost every silicon-neutron collision results in a soft fail [162, 165]. Neutron flux is exponentially influenced by altitude as in equation (2), where $I_I$ represents the cascade flux at altitude $A_I$, L being the attenuation factor or the absorption length.

$$I_2 = I_1 \exp\frac{1}{L}(A_1 - A_2) \qquad (2)$$

About 2% of the hadron flux is made up of protons. Being charged particles, protons must break the shielding of the earth's own magnetic field, which bends their trajectory toward the planet's surface. As a result of both collisions and magnetic shielding, the particles from newly created cascades might end up back into the space. The minimum energy required by a proton to start a cascade of particles that will eventually reach the surface of the earth is called *geomagnetic rigidity*, which depends on the latitude. The proton flux and the upsets it produces are shown in Figure 4-4 a, and b, respectively.

Pions ($\pi$-mesons) are unstable particles with a mass of 135MeV, and account for about 2% of the total hadron flux at sea-level. A pion may exist in the form of a neutral pion (its anti-particle being itself) or a positive pion (its anti-particle being the negative pion). The neutral pion lives for about $10^{-16}$ seconds while positive and negative pions have longer lifetimes of about 26ns ($26 \cdot 10^{-9}$). Since their number is so small compared to other nucleons, one should expect that their SER contribution be negligible.

The neutral pion ($\pi^0$) decays to an electron, positron, and gamma ray by the electromagnetic interaction while charged pions ($\pi^{+/-}$) behave differently, releasing muons and corresponding neutrinos, as indicated by equation (3):

$$\pi^0 \rightarrow e^- + e^+ + \gamma \qquad \pi^+ \rightarrow \mu^+ + \nu_\mu \qquad \pi^- \rightarrow \mu^- + \bar{\nu}_\mu \tag{3}$$

Charged pions can interact with matter. Low-energy positive pions are repelled by the nucleus but when an energetic positive pion loses kinetic energy to a nucleus, it releases a proton (4):

$$\pi^+ + {}_Z^A N \rightarrow p + {}^{A-1}_{\ \ Z} N \tag{4}$$

But the most important contribution to SER is brought by the interaction between matter and negative pions, called *pion capture*. When a nucleus captures a pion, its entire mass is transformed into nucleonic energy, provoking a nuclear fission (5):

$$\pi^- + {}_Z^A N \rightarrow n + {}_{Z-1}^{A-1} N \tag{5}$$

The effect of pion capture has been experimentally tested and it is estimated that *every* negative pion capture within the active volume of an electronic circuit leads to a soft fail. At sea-level measurements of pion capture into silicon is about 8.5/cm$^3$·year [162, 163]. For intermediate energies (between 100 and 250 MeV) pion contributions to SER of modern chips (16-Mb DRAMs) appears to be about 5 times greater that SER caused by protons and may have a quite significant impact at aircraft altitudes [166]. Studies of pion-induced soft errors are still in progress [127].

The muons ($\mu$-leptons) are particles with a lifetime of 2.2 microseconds. They are produced in the upper atmosphere by the decay of pions produced by cosmic rays (Figure 4-3) and follow a decay process releasing electrons (6):

$$\mu^{+/-} \rightarrow e^{+/-} + \nu + \bar{\nu} \tag{6}$$

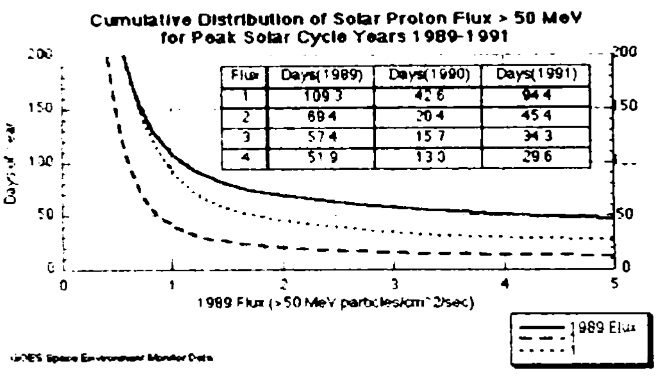
At sea-level the number of muons is about $10^3$ times larger than of pions. There are two effects that could lead to soft fails: the muon capture and the electrostatic muon scattering from nuclei, which is about the same as pion capture rate considering that the rate of inducing a charge greater than 100fC is approximate 7/cm$^3$·year.

A muon capture begins with a negative muon orbiting around a silicon nucleus. By combining with a proton, a neutron and a neutrino are produced, leaving the neutrino with most of the initial mass-energy (7):

$$\mu^- + p \rightarrow n + \nu_\mu \qquad (7)$$

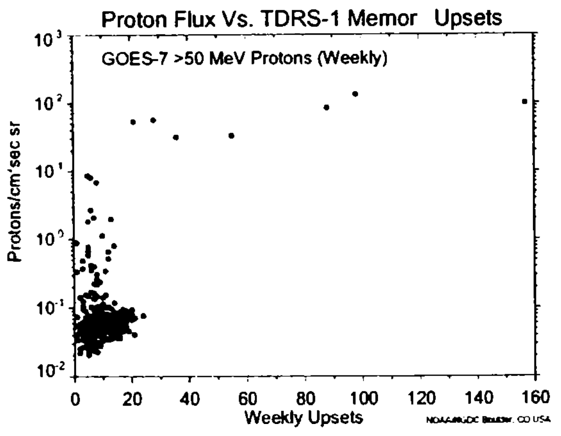
The resulted neutron can further interact with the nucleus causing a range of possible interactions [127, 162]. However, quite few muon captures can cause a charge burst capable of a SEU. The muon capture rate at sea-level is given by equation (8):

$$Muon\ capture\ rate \approx {}^{510}\!/_{cm^3} \cdot year \qquad (8)$$



| Flux | Days(1989) | Days(1990) | Days(1991) |
|---|---|---|---|
| 1 | 109 3 | 42 6 | 94 4 |
| 2 | 69 4 | 20 4 | 45 4 |
| 3 | 57 4 | 15 7 | 34 3 |
| 4 | 51 9 | 13 0 | 29 6 |

A.



B.

Figure 4-4: Distribution of high-energy solar proton flux (A) [175] and weekly upsets produced by the proton flux (B) [174].

### 4.2.5 Ion-Induced SEUs

The earth's atmosphere acts as a filter for many energetic particles that would upset electronic devices. Therefore at high altitudes the number of damaging particles increases, difficult or impossible maintenance leading to a different need of SER-aware types of electronic device design and implementation. Satellites are also sensitive to cosmic rays, the number of potentially upsetting particles at satellite altitudes increasing with the appearance of energetic heavy ions such as iron (Fe), and oxygen (O). Although initial work on energetic heavy ion induced soft fails has not been considered due to discrete satellite components which were highly resistant to radiations, it was found that 100MeV iron particles carry at least part of the responsibility of reported fails [163].

Particle distribution in space is made of 92% protons, 6% $\alpha$–particles while the remaining 2% account for heavier ions (atomic number Z $\geq$2). Early models of SEU vulnerabilities were based on estimations and extrapolations that proved to be unreliable; such was the case when it was believed that SEUs were overwhelmingly dominated by protons and later proved that both protons and heavy ions have to be considered [143].

A first attempt to investigate heavy ion induced SEUs was conducted by Croley et al. with the conclusion that roughly two-thirds of the fails were due to ions with Z $\geq$6 [12, 143]. Elements such as carbon (C), oxygen (O) and iron (Fe) were detected in at least six energy bins while occurrences of nitrogen (N), neon (Ne), magnesium (Mg), silicon (Si), sulphur (S), argon (Ar) and calcium (Ca) were detected in at least one energy bin (Figure 4-5). Measurements were conducted with typical uncertainties in the range of 10-20%. Testing semiconductor devices (Fairchild 93L422) at satellite altitudes (TDRS-1) confirmed the fact (also advanced by Croley et al. [12]) that heavy energetic ions could substantially account for SEU rates, with at least 45%, or roughly equal proportions with the protons, of the SEUs on TDRS-1 being triggered by these particles.

It is therefore considered that, depending on application, heavy ions and protons can show an equally important potential in leading to soft fails. Precise measurements of ion distributions and energies were taken for particle fluxes [142] and are now used by the CREME software for setting of what is called "99% confidence level worst case" [141].

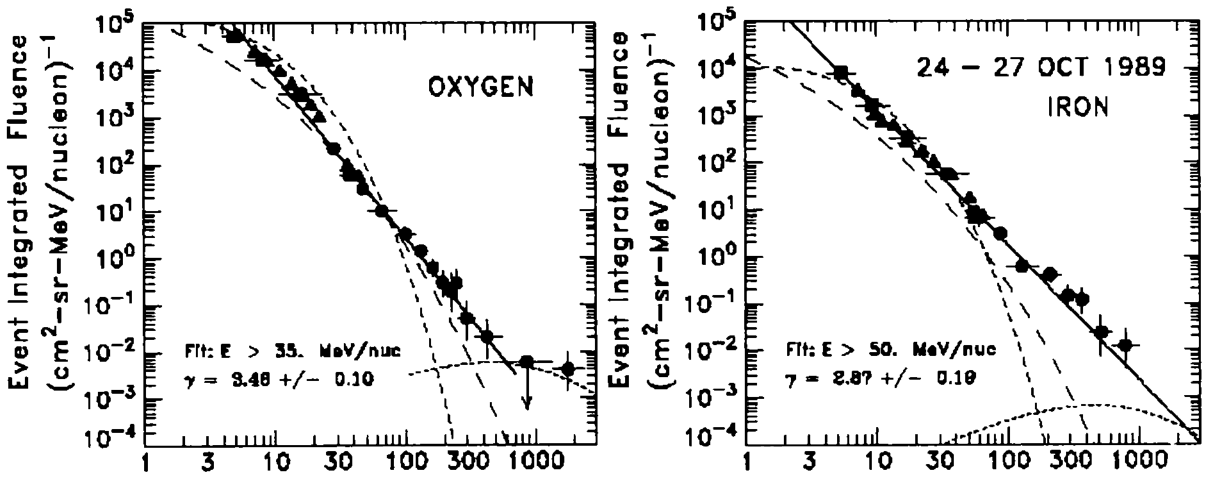


Figure 4-5: TDRS-1 event measurements (kinetic energy in MeV/nucleon expressed horizontally) [143].

### 4.2.6 Neutron-Induced SEUs

The core of each semiconductor device is based on the principles and properties of p-n junctions. When a high-energy neutron strikes a p-n junction, the energetic impact dislocates atoms producing a pathway of electron–hole pairs [113]. Some of the deposited charge will recombine by releasing a short duration current pulse. But if the area hit by the particle is a memory cell and if the charge that accumulates has a greater value than the minimum needed to flip the stored value, than a soft error will result. The minimum charge that results in a soft error is called the *critical charge* ($Q_{CRIT}$) [21]. It depends on the supply voltage values and the effective capacitance of the drain nodes and it plays an important role on estimating the SER. A method for estimating the SER due to atmospheric neutrons (with energy greater than 1MeV) in CMOS SRAM circuits [33] was empirically proven to verify equation:

$$SER \propto FAe^{-\frac{Q_{CRIT}}{Q_S}} = KFAe^{-\frac{Q_{CRIT}}{Q_S}} \qquad (9)$$

where:

- K is a constant independent of device technology with an empirically determined value of $2.2 \cdot 10^{-5}$;
- F is the neutron flux with energies greater than 1MeV, per square centimeter per second;

- A is the area of the circuit sensitive to particle strikes, in square centimeters;
- $Q_{CRIT}$ is the critical charge, in fC;
- $Q_S$ is the charge collection efficiency of the device, in fC; it is a measure of the magnitude of charge generated by a particle strike.

Equation (9) shows the dependency between SER and $Q_{CRIT}$ and $Q_S$; device scaling for memory elements affects the two parameters almost equally (smaller transistors are upset more easily by particles but their sensitive volume is also reduced) while SER per chip in combinational logic increases dramatically with a nine orders of magnitude (technology scaling from 600nm to 50 nm) [33]. Memory cells show a very dependant SER upon cell technology (stacked capacitor SC, trench with external charge TEC or internal charge TIC) as shown in Figure 4-6:

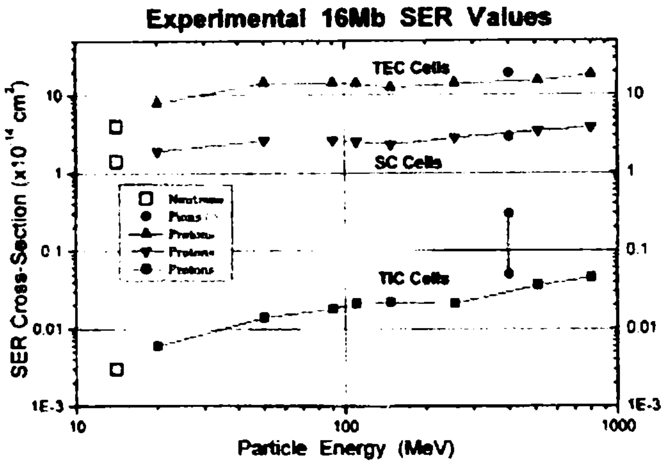


Figure 4-6: Soft-fail cross sections for neutrons, protons, and pions [166].

Being uncharged particles, neutrons do not lose energy by electronic ionization; instead, they interact with the target nucleus via two types of scattering, elastic and inelastic. During an elastic scattering, phenomena taking place is characterized by small recoil energies (the nucleon cannot excite the nucleus, hence a minor role in the production of soft fails) follows equation 10 [127].

$$\textit{nucleon} + \textit{target} \rightarrow \textit{nucleon} + \textit{target} \qquad (10)$$

Inelastic scatterings, on the other hand, involve much higher exchange energies (on the order of MeV or even larger) while the identity of the incoming particle is lost. The process takes place as described by equation 11, where X1, X2, etc can be proton, neutron, deuteron, triton and helium and, if the kinetic energy of the incoming nucleon is greater than 280MeV secondary pions may be produced. An example of inelastic scattering would be of a 200MeV energetic neutron hitting a silicon wafer (see equation 12):

$$\textit{nucleon} + \textit{target} \rightarrow X_1 + X_2 + \ldots + X_n + \textit{residual nucleus} \qquad (11)$$

$$n + {}^{28}Si \rightarrow 2p + 2n + {}^{25}Mg^{*} \qquad {}^{25}Mg^{*} \rightarrow n + 3\alpha + {}^{12}C \qquad (12)$$

The secondary particles generated by the interactions taking place are given in Table 4-3. Since elastic and inelastic scatterings involve complex nuclear reactions, providing an accurate result for the number of induced soft errors is quite difficult to obtain. However, with semiconductor technology scaling the case of devices with QCRIT<50fC will be reached in the future, a moment when elastic scattering induced soft errors may not be negligible [127].

### 4.2.7 SEUs Induced by Alpha Particles

Another source of soft errors in semiconductor devices consists in their capacity of being affected by stray *alpha* particles. The naturally occurring decaying processes of chemical elements (such as uranium and thorium) generate helium ions $^4He$ (stable helium isotopes composed of two protons and two neutrons) also known as $\alpha$–particles. They are also a common product of radioactive $\alpha$-decay due to impurities in semiconductor chip materials and packaging (such as uranium and thorium). The $\alpha$-decay process may release different energetic particles, and can be represented by equation:

$$^{A}_{Z}X_{N} \rightarrow ^{A-4}_{Z-2}Y_{N-2} + \alpha \qquad (13)$$

where X and Y are known as the mother and daughter nuclides.

| Secondary particle | Kinetic energy | Electron-hole pairs / $\mu$m | Particle range ($\mu$m) |
|---|---|---|---|
| p | 5.224 | 13.51 | 225 |
| p | 4.195 | 15.91 | 155 |
| n | 65.478 | 0 | ∞ |
| n | 22.958 | 0 | ∞ |
| n | 6.815 | 0 | ∞ |
| $\alpha$ | 12.218 | 79.91 | 90.5 |
| A | 12.025 | 80.83 | 88.1 |
| A | 7.881 | 108.84 | 43.6 |
| $^{12}C$ | 4.138 | 1253.34 | 3 |

Table 4-3: Reaction between 200MeV neutrons and silicon nuclei [127].

These particles have typical kinetic energies of the order of 2-9MeV [13, 127] and proved to be most upsetting with respect to semiconductor devices; in comparison with others, $\alpha$–particles have a very large mass and charge, and they can produce an inside sudden burst of a million electrons over a narrow, few-microns-wide path length. The burst may result in altering the quantum of a memory cell, thus producing a SEU [14, 75, 110]. The decaying elements previously mentioned as uranium and thorium can be traced as impurities present in chip and packaging materials [113, 114]. As technological progress was made, SER due to $\alpha$–particles lost momentum through careful supervision of the quality of materials used.

In semiconductor industry, the trends of shrinking technology to sub-0.25μm, decreasing supply voltage and node capacitances, make the SER due to alpha particles a potentially major reliability concern to logic processes because of the quadratically decreasing critical charge [14, 15, 114]. Package designs, such as lid coat or flip chip, strongly influence the SER induced by alpha particles, increasing more rapidly with decreasing critical charge than neutron induced SER [138, 139]. Experiments and simulations conducted [110] conclude that over the last technological processes the sensitivity of logic circuits to alpha particles has decreased, while due to device scaling and higher flux of lower energy neutrons they are more vulnerable to SEUs induced by these particles [161].

Another cause that could generate $\alpha$-particles is a class of unstable, charged subatomic particles called leptons. One particular lepton, the *$\mu$-lepton* called simply a *muon*, can emit such particles when captured by an electronic device. Varying with geographical position it was measured that $\alpha$-particles are also emitted by at most 3% of captured muons (see equation 14) [161]:

$$Muon - induced\ \alpha - particles \approx {}^{15}\!/_{cm^3} \cdot year \tag{14}$$

### 4.2.8 Proton-Induced SEUs

The cosmic rays reach their highest intensity at high altitudes, all elevated-altitude tests proving the existence of soft fails in electronic equipment [175]. In fact, military aircraft featuring high performance computing material and large memories have experienced SEUs on almost every flight and it seems that at airplane altitudes the SER is at least 100 times worse than at terrestrial altitudes [42, 83, 162]. Particle distribution also changes with altitude, as shown in Table 4-4:

| Altitude | Hadrons | | |
|---|---|---|---|
| | Neutrons | Pions | Protons |
| Sea Level | 94% | 3% | 3% |
| 10kms | 52% | 36% | 12% |

Table 4-4: Variation of particle distribution with altitude, from sea level to 10.000 meters [166].

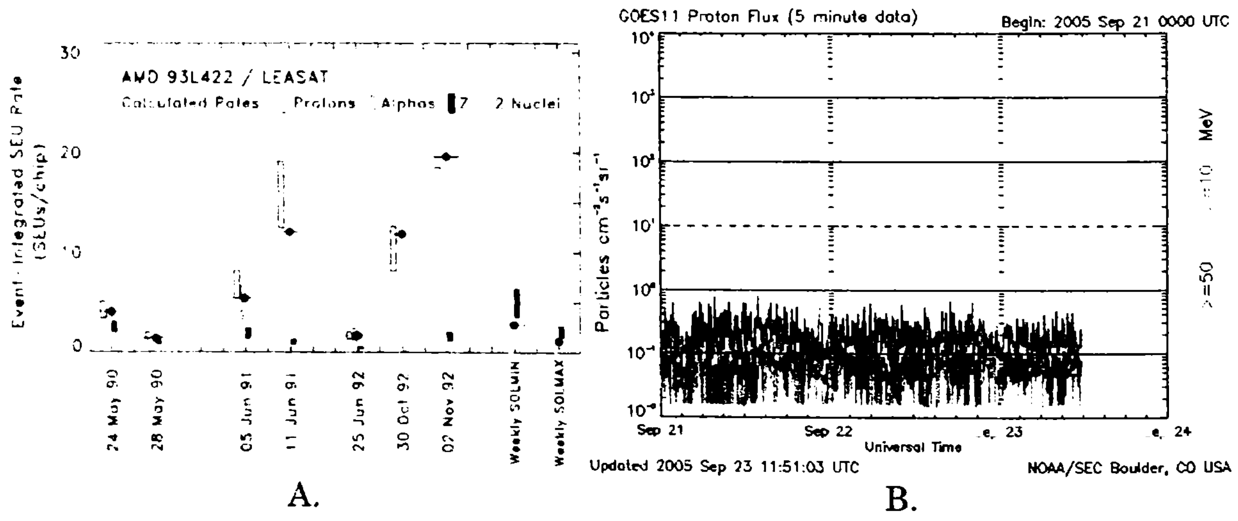


Figure 4-7: A. Observed SEU rates (SEUs per chip) compares with calculations of proton-, alpha-, and heavy ions-induced rates [143]; B. GOES-10 proton flux measurements (thresholds at10, 50, and 100MeV) [173].

Figure 4-7 shows an actual measurement of high energy proton flux by NOAA's satellites. At these energies, protons are quite similar with neutrons in their potential of affecting electronic circuits. An example of inelastic scattering involving protons and a silicon wafer is given by equation:

$$p + {}^{28}Si \rightarrow \alpha + {}^{25}Al \tag{15}$$

Calculations between proton-induced SEU rate and actual measurements performed on board of TDRS-1 satellite given in Fig. 2-9 reflect existing uncertainties concerning the cross-sections, pointing out the difficulties that arise when assessing proton influences over semiconductor materials.

### 4.2.9. Conclusions

In order to evaluate SER induced by cosmic particles a number of models were developed, such as CREME96 (Cosmic Ray Effects on Micro-Electronics) and NUSPA (Nuclear SPAllation model). Significant progress has been made over the years for a thorough understanding of how various particles interact with semiconductors developed, doubled by extensive testing [12, 42, 83, 165]. As a result, key aspects about particle influence have been determined quite precisely: their flux can be measured and the existing dependencies are mostly settled, as are relative SER contributions (Figure 4-8). We know neutrons constitute the main problem; fail rates for computer memory chips can be estimated even if there still are some problems when computing absolute neutron fluxes. Cosmic rays are filtered by a variety of natural (geo-magnetic field, atmosphere) and artificial factors (such as concrete shielding) [162]. However, more research needs to be carried on precise particle flux estimation [20].

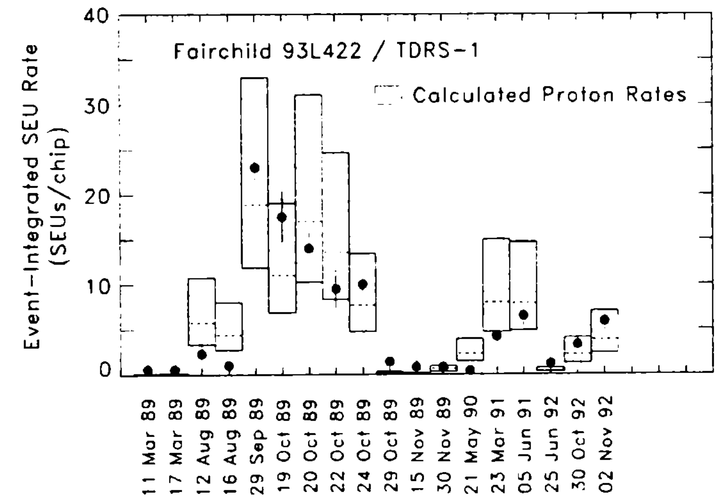


Figure 4-8: Comparison between measured SEU reported by TDRS-1 and calculated rates for proton-induced soft fails [143].

In present days, the sensitivity of electronic devices is investigated using irradiation techniques in some of the world's most advanced facilities, such as the CERN particle physics laboratory; as a result, new physical layouts emerge, more resistant to particle influences [17]. Technology testing in irradiated environments does not exclude commercial products, one of the goals being the adaptation of commercial off-the-shelf (COTS) supercomputer hardware to space applications; moreover, technology itself shows different behaviors as the most resistant to nuclear induced soft fails appear to be circuits built in CMOS or nMOS, while the most sensitive devices are built using the bipolar process. By constantly checking the overall quality of semiconductor devices it is possible to achieve SER variations between identical circuits no more than 1.5 times (in case of IBM) though commercial LSI circuits were reported to differ by as much as 200 times [163]. The sensitivity of PC microprocessors such as Intel Pentium III and AMD K7 was evaluated based on proton exposure, confirming the presence of SEUs and functional interrupts, but also the possibility of space operation by applying mitigation techniques and some functional constraints [36]. Its is however clear that COTS hardware cannot pass space operating requirements and dedicated technological processes must be employed in order to release radiation tolerant hardware (Space Environment Modular Design SEMD™, RAD-PAK® package shielding and others); SEMD™ technique alone claims to improve device resistance to SEU flip-flop errors by as much as 3000 times (from $3x10^{-7}$ without to $10^{-10}$ errors/bit·day with the technique) [18].

Much work has been done over studying how atomic or subatomic particles, many of which are created by ways of cosmic radiation, affect electronic devices: computer memories were tested and found susceptible to heavy ion and proton SEUs (see Table 4-5), with periods between events ranging from years to fractions of hours [28, 34, 83, 164, 166]; computer logic is also affected [51, 113, 114, 110, 143] (though combinational logic is less prone to soft errors than sequential logic, for a number of reasons [23, 113]) and new semiconductor designs such as FPGAs are already integrating radiation tolerant techniques [9, 10, 27]. This chapter's main task is not to bring a complete coverage of the work carried out in the SER area, but to complete the Embryonics' picture with the soft fails landscape and the need to overcome such phenomena, not by technological steps but by harmoniously embedding fault tolerance with its native capabilities.

| Chip type | Observed SER | Typical application |
|---|---|---|
| 4Kb bipolar | 1.340 | Cache memory |
| 288 Kb DRAM | 126.000 | Main memory |
| 1Mb DRAM | 3.000 | Main memory |
| 144Kb CMOS | 210 | Secondary cache |
| 9Kb bipolar | 998 | I/O channels |

Table 4-5: SER data for a variety of IBM memory chips [165].

## 4.3. A Reliability Analysis

The vast majority of soft errors are actually affecting single bits. As has been stated in [5], laboratory observations of computer memories show that "*by far the most common type of chip failure is a soft error of a single cell on a chip*". Independent memory testing revealed that there is also a small percentage of multiple bit flips. These events account for 1-7% of the total soft fails recorded, but simultaneous high bit flip events are far less frequent (only 2 cases of quadruple bit flip events witnessed with a predicted rate of about one such event in 65 years per device) [34]; other measurements show that double bit-flips account for under 5% of the total events [166]. Unfortunately, the anomalous, fluctuating behavior of particle spectra prevents an exact estimation of the soft fails type probabilities (i.e. affecting single, double, or multiple bits) and typical uncertainties allowed by current particle interaction models (usually about 20-30% or more) allow only for a range of calculations.

### 4.3.1. Datapath Model for Embryonic Memory Structures

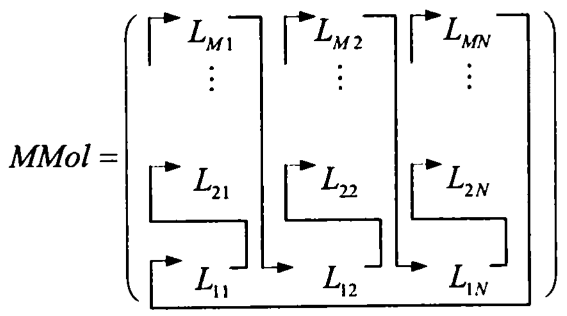
In order to perform a reliability analysis over the Embryonics memory architecture, a formal model of its functions may lead to a better understanding of fault characteristics and suggest choices in implementing fault tolerance.

Let it be considered a memory structure of size *M*x*N*, which is actually a rectangular array of molecules, each operating in the same memory mode; we call it a *macro-molecule*, since it actually has a kind of membrane coded in the operative genome that groups molecules from the same memory area together. A memory molecule has a storage capacity of *F* bits of data given by a chain of flip-flops from the configuration register *CREG*, where *F* can be either 8 or 16, depending of the memory sub-mode the molecule operates in. Given the fact that Embryonics uses a programmable number of spare molecules for self-repairing purposes (see Chapter 3, Section 3.7), we will consider our memory structure – without affecting the generality – as including *s* columns of such spares.

Such a structure would be seen mathematically as a tri-dimensional matrix, composed of *M* rows and *N* columns of physically identical storage molecules, each implemented as a chain of *F* elementary 1-bit memory cells (flip-flops) as described by equation 16:

$$MMol = \begin{pmatrix} L_{N1} & L_{N2} & \cdots & L_{NM} \\ \vdots & \vdots & & \vdots \\ L_{21} & L_{22} & \cdots & L_{2M} \\ L_{11} & L_{12} & & L_{1M} \end{pmatrix}, \text{ where each } L_{ij} = \begin{pmatrix} c_{ij}^{1} & c_{ij}^{2} & \cdots & c_{ij}^{F} \end{pmatrix} \qquad (16)$$

The memory molecules operate together by synchronously cycling the contained information from one element to another (see equation 17), the data inside any given molecule being offset by one bit with each clock period. This process allows us to define for each $L_{i,j}$ a vicinity $V(L_{i,j})$, denoted by $L_{x,y}\ L_{i,j}\ L_{z,w}$, indices *x*, *y*, *z*, and *w* being defined by equation 18.



(17)

$$V(L_{i,j}) = \begin{cases} 1 < i < M,\ 1 < j < N, & L_{i-1,j} \quad L_{i,j} \quad L_{i+1,j} \\ i = 1,\ 1 < j, & L_{M,j-1} \quad L_{1,j} \quad L_{i+1,j} \\ i = j = 1, & L_{M,N} \quad L_{1,1} \quad L_{2,1} \end{cases} \qquad (18)$$

$$\tilde{L}_{i,j} = V(L_{i,j}) \times M^{\bullet}, \qquad M^{\bullet} = \begin{pmatrix} 0_{(F-1)\times F} \\ I_F \\ 0_{(F+1)\times F} \end{pmatrix} \qquad (19)$$

Under the above considerations, we denote the data content of the molecule $L_{i,j}$ after a clock period as $\tilde{L}_{i,j}$. Using these notations the mathematical process

$\tilde{L}_{i,j}$ of circling data could be seen as a matrix operation between the vicinity of $L_{i,j}$ and a matrix $M^*$ (see equation 19).

Soft errors can be modeled by altering the contents of the vicinity matrix $V(L_{i,j})$ in the appropriate manner, a flip-type error (that is, a bit changing its state from 0 to 1 or from 1 to 0) appearing formally as an exclusive-or operation between the vicinity matrix and the error-injecting matrix for the same bitstream, which we will call $E(L_{i,j})$. The data transfer is then defined in the general case as indicated in equation 20.

$$E(L_{i,j})(k) = \begin{cases} 1, & \text{if bit } k \text{ in } L_{i,j} \text{ is erroneous} \\ 0, & \text{otherwise} \end{cases}$$
$$\tilde{L}_{i,j} = \left(V(L_{i,j}) \oplus E(L_{i,j})\right) \times M^* \qquad (20)$$

For instance, if the molecular storage array has 6 lines and 9 columns and if the molecule affected by a soft error is considered to be $L_{14}$, then the vicinity matrix contains three molecules, $L_{63}$ [$L_{14}$] $L_{24}$. Furthermore, if a molecule has a storage capacity of 8 bits and the affected bits are bit 4 of $L_{63}$ and bit 6 of $L_{14}$, then the data transfer takes place according to equation 21:

$$V(L_{1,4}) = \left(L_{6,3} \quad L_{1,4} \quad L_{2,4}\right)$$
$$E(L_{1,4}) = (00010000 \mid 00000100 \mid 00000000) \qquad (21)$$
$$\tilde{L}_{1,4} = \left(V(L_{1,4}) \oplus E(L_{1,4})\right) \times M^*$$

### 4.3.2. Preliminaries

As indicated by the datapath model introduced previously, there are no functional dependencies between data bits inside a macro-molecule. We will therefore make the assumption that, for individual flip-flops inside a molecule, failures are exponentially distributed. Similar assumptions have been found to work well for classic computer memories and even for Embryonics [5, 84, 85, 86]. Therefore the reliability of a single memory element *$R_{FF}(t)$*, that is, the probability that after t hours of normal operating that particular component has still not failed, can be considered to be equal to $e^{-\lambda t}$, where $\lambda$ is a constant found experimentally [73]. Since in our case the memory element is a flip-flop, we have:

$$R_{FF}(t) = e^{-\lambda t} \qquad (22)$$

The mean time to failure (*MTTF*) for the flip-flop is defined as follows:

$$MTTF_{FF} = \int_0^\infty R_{FF}(t)dt = \int_0^\infty e^{-\lambda t}dt = \frac{1}{\lambda} \qquad (23)$$

A molecule operating in memory mode uses a number of *F* memory elements chain together to allow serial data shifting. Assuming that the flip-flops are failing independently, the reliability of a memory molecule *$R_{MMol}(t)$* is given by the probability that after t hours of normal operating none of the memory elements have failed:

$$R_{Mol}(t) = \prod_{i=1}^{F} (R_{FF})_i (t) = (R_{FF})^F (t) = e^{-\lambda F t} \tag{24}$$

The corresponding mean time to failure for the whole memory molecule is given by equation 25:

$$MTTF_{Mol} = \int_0^\infty R_{MMol}(t)dt = \int_0^\infty e^{-\lambda F t}dt = \frac{1}{\lambda F} \tag{25}$$

As pointed previously, our considered memory structure is a rectangular array, composed of *M* interconnected rows. It is safe to say that rows are assumed to fail independently, therefore the reliability function for the macro-molecule (the entire memory structure) is given by equation 26:

$$R_{MMol}(t) = \prod_{i=1}^{M} (R_{Row})_i (t) = (R_{Row})^M (t) \tag{26}$$

Each row is composed of a number of *N* interconnected memory molecules, which are also assumed to fail independently, the reliability and mean time to failure functions for an entire memory row being given by equation 27:

$$\begin{aligned} R_{Row}(t) &= (R_{Mol})^{N-s}(t) = e^{-\lambda F (N-s) t} \\ MTTF_{Row} &= \int_0^\infty e^{-\lambda F (N-s) t}dt = \frac{1}{\lambda F (N-s)} \end{aligned} \tag{27}$$

Combining equations 26 and 27 we can now compute the reliability function for a whole memory structure as being:

$$R_{MMol}(t) = ((R_{Mol})^{N-s})^M (t) = e^{-\lambda F M (N-s) t} \tag{28}$$

Considering equations 22–28 the mean time to failure for the entire memory structure is given by the following:

$$MTTF_{MMol} = \int_0^\infty R_{MMol}(t)dt = \int_0^\infty e^{-\lambda F M (N-s) t}dt = \frac{1}{\lambda F M (N-s)} \tag{29}$$

Equation 29 points out a common sense consequence: if a component has a given reliability then a system composed of a number of such identical components will have a corresponding reliability exponentially decreasing. As the memory structures grow larger, the radiation sensitive cross-section increases also, making the memory more vulnerable to particle capture and the associated side effects. Therefore, the overall reliability and *MTTF* of the memory structure are affected. The $\lambda$ parameter, which is determined empirically and supposed to be constant (further considerations are given in Section 4.3.6), can potentially modify its value due to variations between the testing laboratory environment and changing, harsh environments such as space or high-altitude, thus offsetting the resulting reliability figures.

The Embryonics project has Nature as its primary source of inspiration. It was born to attempt a transfer of nature's proven mechanisms to electronic, digital systems. The robustness found in biological systems would translate into reliable electronic systems and is already part of Embryonics implementations of functional logic; therefore extending bio-inspiration to implement reliable memories comes as a natural next step.

### 4.3.3. Strategies for Macro-Molecular Fault-Tolerance

There are numerous coding techniques to ensure that computer memories tolerate some faults that might appear. Since the most common error affects only a single memory element, i.e. a single bit, the most used code class is that capable of SEC–DED (Single Error Correcting – Double Error Detecting codes). A memory using such a form of coding is capable of tolerating only single errors; burst or multiple errors cannot be tolerated, but some cases of multiple errors may be however detected successfully.

Embryonics could take advantage of SEC–DED codes, particularly considering that for building reliable memory structures there is an obvious need for a form of error coding. Furthermore, a coding process also allowing multiple error detection/correction is desirable, since the robustness in Embryonics is ensured by a self-repairing process distributed over two levels: molecular and cellular. The reliability analysis made previously is affected by the capacity of recovering from a single error and of detecting multiple errors. From now on, we will consider that failures inside memory molecules are exponentially distributed, they occur independently from each other and one can distinguish between the following situations:

A. Single failure; the core of a molecule is affected by a single error only. This situation can be recovered using parity-based coding schemes.

B. Double failure; the core of a molecule is affected by at least one error, but there are at most two errors on the same row of molecules. The relative distance between the errors is random. In this case the use of more sophisticated, Hamming-like codes is required.

C. Multiple failure; similar to case B but the likelihood of such an event was found to be minimal [34].

D. Terminal failure; there are too many fault occurences either in one molecule or in a whole row. This situation cannot be recovered.

E. No failures detected. Either the molecules operate normally or an undetectable combination of errors has occurred. This situation does not require or there cannot be established any measures to be taken.

In case of failure, the conditional probabilities that the failure will be of type A, B, C, or D, will be $a$, $b$, $c$, and $d$, respectively. Given all these assumptions, any reliability function concerning the entire memory structure is defined as follows:

$$R(t) = Prob\{\ \textit{an unrecoverable combination of errors has not yet occurred at time } t\ \}$$

Given the characteristic features of the Embryonics project, we may consider multiple strategies of tolerating faults, divided into two categories:

- *Fault tolerance at the molecular level.* The main advantage of this strategy might lie in the possibility of isolating the faulty molecule and make use of the transparent reconfiguration process after the "death" of the respective memory molecule. On the disadvantage side it should be noted that a considerable portion of the molecular core has to be affected for redundant coding, as well for the additional logic such a configuration would involve.
- *Fault tolerance at the macro-molecule level.* This strategy would use separate macro-molecules for redundant coding and additional logic not implemented inside, but by molecules themselves, thus reducing the space required (in terms of molecules). As a disadvantage, the use of the reconfiguration process would become quite difficult due to the lack of an

addressing mechanism: one cannot localize where a faulty molecule is located and even then, a "kill" signal would have to be sent to that molecule from outside. None of the two aspects are implemented in the current design.

In order to make an accurate assessment of the reliability capabilities and potential of memory structures in Embryonics, we will consider and compare two cases of fault tolerance strategies: single error correcting (SEC) and double error correcting (DEC).

### 4.3.4. Fault-Tolerance at the Macro-Molecular Level

Estimating the overall reliability of an entire macro-molecule can be regarded both from a classic perspective and from a different approach using approximating Poisson processes [5]. The former is based only on the empirically estimated parameter λ, while the latter relies on superimposing faults occuring in the same row onto a single device called a *protochip*. In our case, such a protochip is defined as the entire collection of memory molecules that make up a row of the corresponding macro-molecule. Given the fact that soft failures appear to be uniformly distributed and independent from each other, the Poisson assumption can be used: in each protochip, failure types form independent Poisson processes.

Our macro-molecule is a rectangular array made of $M$ rows and $N$ columns (out of which s are spares) of memory molecules, each containing $F$ flip-flops as storage units. Then the macro-molecule can be assimilated to a single protochip composed of $N$ individual chips (including s as spares), each chip being composed of $M$x$F$ storage units.

#### 4.3.4.1. SEC Strategy

If the occurring failure events were only of type A, then the reliability of one memory molecule could be written as equation 30:

$$R_{Row}(t) = \text{Prob}\{\text{no flip-flop fails}\} + \text{Prob}\{\text{single flip-flop fail}\} \tag{30}$$

If λ is the failure rate for a storage flip-flop from inside a molecule, it can now be computed that:

$$R_{Row}(t) = e^{-\lambda F(N-s)t} + F(N-s)\left(1-e^{-\lambda t}\right)e^{-\lambda(F(N-s)-1)t} \tag{31}$$

which is represented in Figure 4-9 for λ=0.02, and different values of $F$, $N$, and $s$. Then the mean time to failure for a memory row becomes:

$$MTTF_{Row} = \int_0^\infty e^{-\lambda F(N-s)t} + F(N-s)\left(1-e^{-\lambda t}\right)e^{-\lambda(F(N-s)-1)t}dt$$

$$\begin{aligned} MTTF_{Row} &= \int_0^\infty e^{-\lambda(F(N-s)-1)t}dt - \left(F(N-s)-1\right)\int_0^\infty e^{-\lambda F(N-s)t}dt \\ &= \frac{1}{\lambda}\left(\frac{1}{F(N-s)} + \frac{1}{F(N-s)-1}\right) \end{aligned} \tag{32}$$

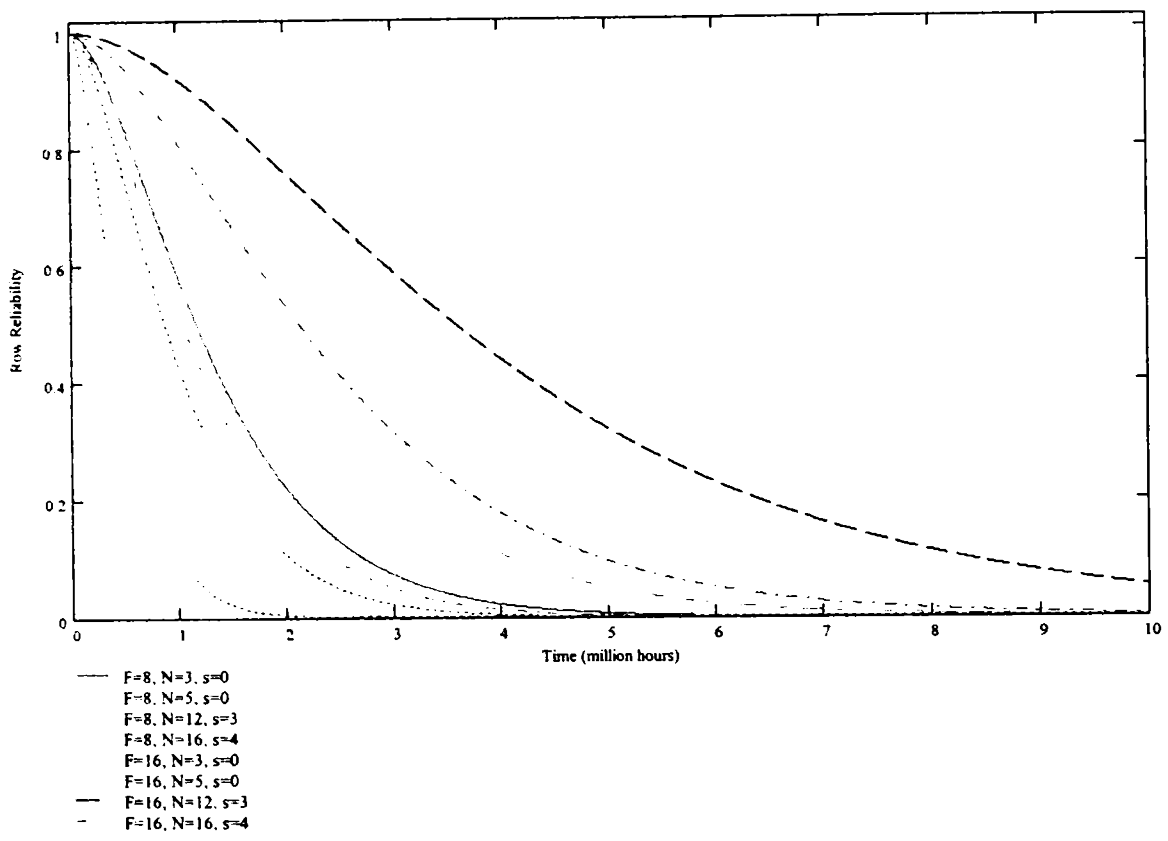


Figure 4-9: Row reliability for different macro-molecular configurations.

Considering that a macro-molecule consists of $M$ rows, the reliability function can be extended to an entire macro-molecule (equation 34) and represented in Figure 4-10:

$$R_{MMol}(t) = \left(R_{Row}\right)^M (t) \tag{33}$$

Using the notation $A = F(N-s)$ we have:

$$\begin{aligned} R_{MMol}(t) &= \left(e^{-\lambda At} + A\left(e^{-\lambda(A-1)t} - e^{-\lambda At}\right)\right)^M \\ &= e^{-\lambda AMt}\left(1 - A + Ae^{\lambda t}\right)^M \\ &= \sum_{i=0}^{M} \binom{M}{i} (1-A)^i A^{M-i} e^{-\lambda(M(A-1)+i)t} \end{aligned} \tag{34}$$

and the mean time to failure for a macro-molecule is given by equations 35–36:

$$MTTF_{MMol} = \sum_{i=0}^{M} \binom{M}{i} (1-A)^i A^{M-i} \int_0^{\infty} e^{-\lambda(M(A-1)+i)t} dt \tag{35}$$

$$MTTF_{MMol} = \frac{1}{\lambda} \sum_{i=0}^{M} \frac{1}{M(A-1)+i} \binom{M}{i} (1-A)^i A^{M-i} \tag{36}$$

Using the Poisson assumption, we now focus on a protochip component. Since each component (or chip) contains *MxF* storage units and is vulnerable only to single failures, then these form a Poisson process of intensity $a(N-s)\lambda t \big/ FM$, where $a$ is the conditional probability for a failure of type A.

Therefore, the probability that each chip experienced at most one single error at time $t$ is:

$$R_{Row}(t) = e^{-\frac{a(N-s)\lambda t}{FM}}\left(1+\frac{a(N-s)\lambda t}{FM}\right) \tag{37}$$

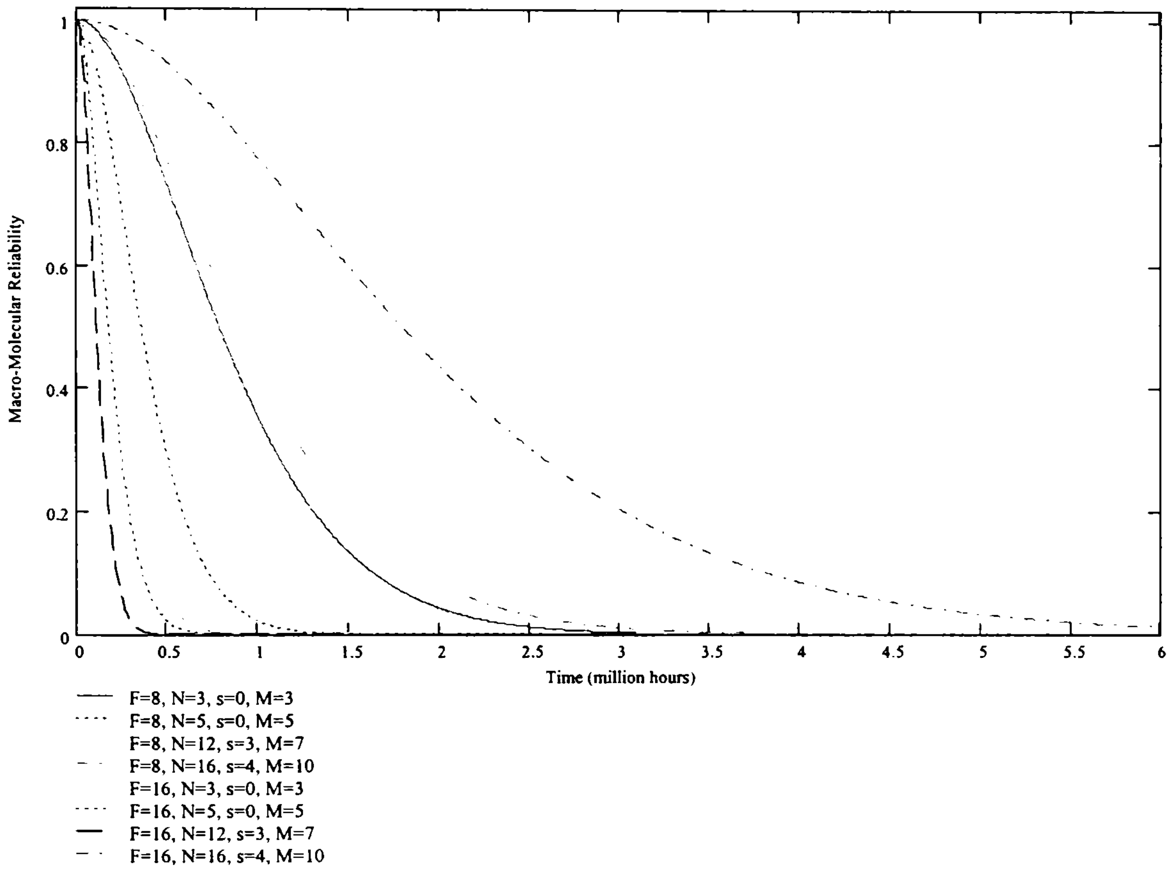


Figure 4-10: Overall reliability for different macro-molecular configurations.

Because the protochip consists of $M$ rows, the protochip reliability can then be derived as equation 38, which is represented in Figure 4-11.

$$R_{MMol}(t) = e^{-\frac{a(N-s)\lambda t}{F}}\left(1+\frac{a(N-s)\lambda t}{FM}\right)^{M} \tag{38}$$

## 4.3.4.2. DEC Strategy

We consider here failures of type A and B only; due to their reported rareness, failures of type C will not be considered. The macro-molecule layout in Embryonics could be seen as an association of rows of molecules, a stored instruction consisting of assembled bits from the same row. Therefore it appears quite similar to the definition of a protochip supporting the Poisson assumption, asserting that failure types form independent Poisson processes [5]. If the probability of a type A failure occurrence is $a$, and the probability that a given molecule has not yet failed at time $t$ is $e^{-\lambda t}$, then the reliability of one memory molecule can then be written as:

$$R_{Row}(t) = \text{Prob}\{\text{no flip-flop fails}\} + \text{Prob}\{\text{flip-flop fails}\}$$
$$\text{Prob}\{\text{flip-flop fails}\} = \text{Prob}\{\text{single fails only}\} + \text{Prob}\{\text{double fails}\} \tag{39}$$

Because we considered a macro-molecule of $M$ rows and $N$ columns of molecules with a storage capacity of $F$, also containing s columns of spares, and each molecule the probability that at time $t$ no fails or at most one single fail have been traced in a row is a Poisson process of intensity $a(N-s)\lambda/FM$. Then for a certain row we have the probabilities $P_0$ that there have been no failures traced in a row, and $P_1$ that there has been exactly one failure traced in a row (case A):

$$P_0(t) = e^{-\frac{a(N-s)\lambda t}{FM}} \tag{40}$$

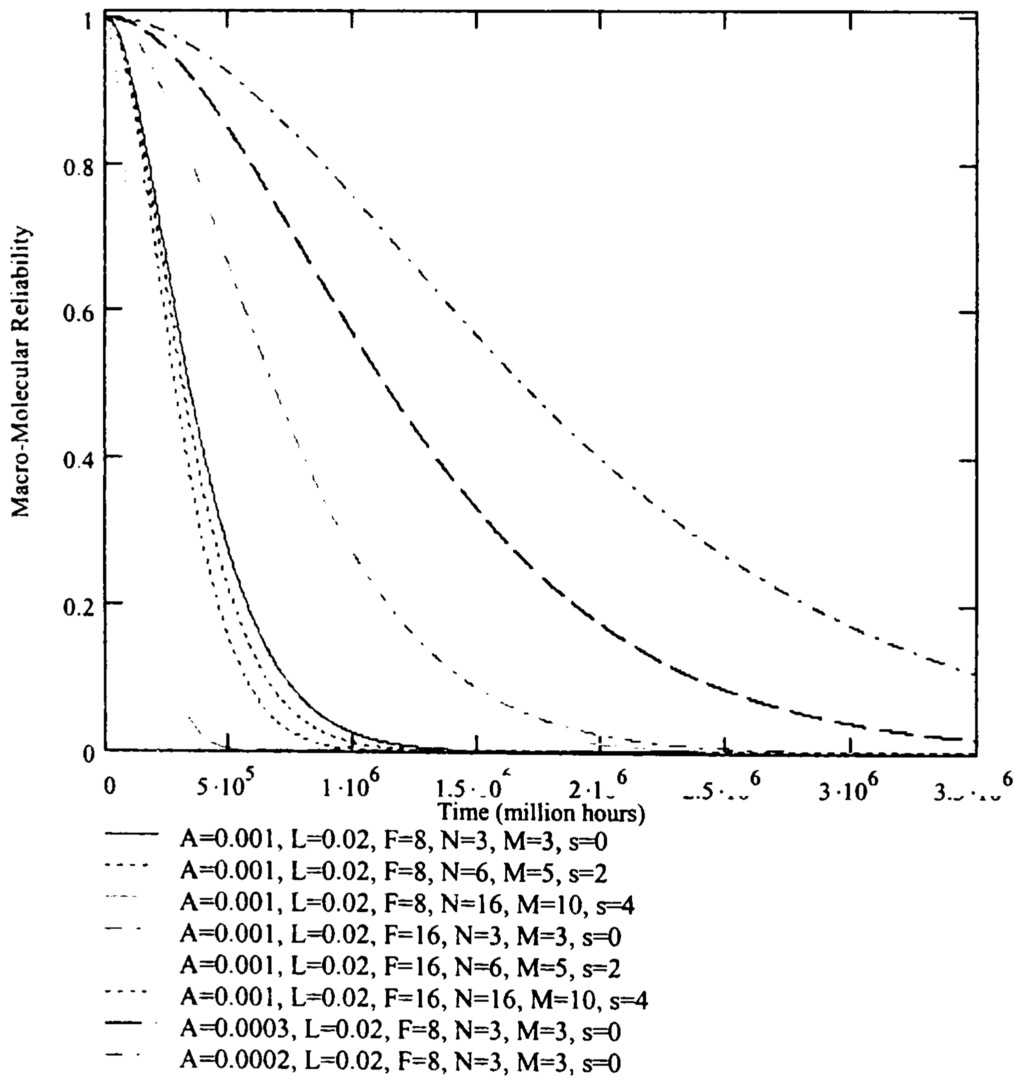


Figure 4-11: Macro-molecular reliability for different SEC configurations using Poisson assumptions.

$$P_1(t) = \frac{a(N-s)\lambda t}{FM} e^{-\frac{a(N-s)\lambda t}{FM}} \tag{41}$$

The probability of a double failure $P_2$ (case B) can be regarded as the conditional probability $P_{11}$ of a single fail, given the fact that another single fail already occurred in the same row. Because our macro-molecule's architecture prevents the existence of "nested" or "hierarchical" failure types, all failures being independent and exponentially distributed, we have:

$$P_2(t) = P_{11}(t) \simeq (P_1)^2(t) = \frac{a^2(N-s)^2\lambda^2t^2}{F^2M^2}e^{-\frac{2a(N-s)\lambda t}{FM}} \tag{42}$$

Then the probability $P_{Row}$ that at time $t$ a certain row did experience at most a double failure, which is actually its reliability value, will be the sum of probabilities $P_0$, $P_1$, and $P_2$ (equation 43, Figure 4-12), and gives the reliability function for an entire row:

$$\begin{aligned} R_{Row}(t) &= (P_0+P_1+P_2)(t) \\ &\simeq e^{-\frac{a(N-s)\lambda t}{FM}}\left(1+\frac{a(N-s)\lambda t}{FM}+\frac{a^2(N-s)^2\lambda^2t^2}{F^2M^2}e^{-\frac{a(N-s)\lambda t}{FM}}\right) \end{aligned} \tag{43}$$

Using the notation $B=\dfrac{a(N-s)\lambda}{FM}$ the mean time to failure for a memory row becomes:

$$\begin{aligned} MTTF_{Row} &= \int_0^\infty\left(e^{-Bt}(1+Bt)+B^2t^2e^{-2Bt}\right)dt \\ &= \frac{1}{B}+\frac{1}{8B^2}-\frac{1}{B}(1+Bt)e^{-Bt}\Big|_0^\infty+\frac{1}{2}\left(t-Bt^2\right)e^{-2Bt}\Big|_0^\infty \end{aligned} \tag{44}$$

Given our previous assumption concerning failure distribution, the reliability and *MTTF* of an entire macro-molecule are then given by equations 45 and 46, respectively, while reliability graphs are represented in Figure 4-12 for different configurations:

$$\begin{aligned} R_{MMol}(t) &= (R_{Row})^M(t) \\ &\simeq e^{-\frac{a(N-s)\lambda t}{F}}\left(1+\frac{a(N-s)\lambda t}{FM}+\frac{a^2(N-s)^2\lambda^2t^2}{F^2M^2}e^{-\frac{a(N-s)\lambda t}{FM}}\right)^M \end{aligned} \tag{45}$$

and the mean time to failure for the macro-molecule:

$$\begin{aligned} MTTF_{MMol} &= \int_0^\infty R_{Mem}(t)dt = \int_0^\infty (R_{Row})^M(t)dt \\ &\simeq \int_0^\infty e^{-\frac{a(N-s)\lambda t}{F}}\left(1+\frac{a(N-s)\lambda t}{FM}+\frac{a^2(N-s)^2\lambda^2t^2}{F^2M^2}e^{-\frac{a(N-s)\lambda t}{FM}}\right)^M dt \end{aligned} \tag{46}$$

### 4.3.5. Fault-Tolerance at the Molecular Level

In this case all occurring failures are tolerated locally (i.e. inside the molecule) and therefore the size of the protochip shrinks to the size of one molecule. In order to compute the reliability function of an entire macro-molecule (which will result quite complex) it is necessary to evaluate the failure rate for its basic brick, which is the flip-flop. The reliability of a molecule is already known and can be alternatively written as the corresponding product of flip-flop reliabilities (see equation 47):

$$\left.\begin{array}{l} R_{Mol}(t) = (R_{FF})^F(t) \\ R_{Mol}(t) = e^{-\lambda t}, R_{FF}(t) = e^{-\lambda_{FF} t} \end{array}\right\} \Rightarrow \lambda_{FF} = \frac{1}{F}\lambda \qquad (47)$$

## 4.3.5.1. SEC Strategy

If each molecule can tolerate at most one single error, then its reliability is defined as:

$$R_{Mol}(t) = \text{Prob}\{\text{no flip-flop fails}\} + \text{Prob}\{\text{single flip-flop fail}\}$$

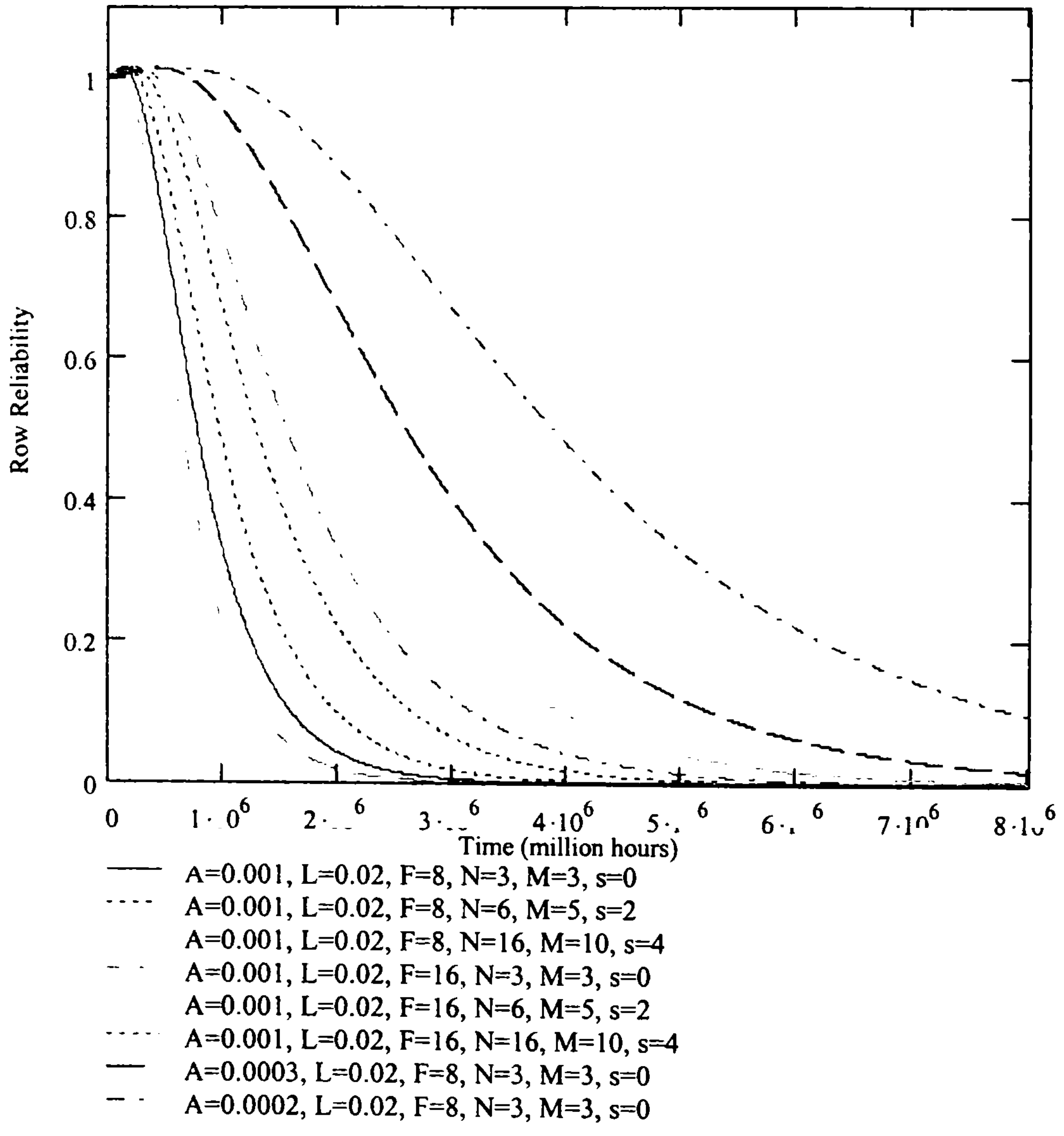


Figure 4-12: Row reliability for different configurations.

$$R_{Mol}(t) = e^{-\lambda t} + F\left(1 - e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t} \qquad (48)$$

#### 4.3.5.2. DEC Strategy

If each molecule can tolerate at most two errors, then its reliability is defined as:

$$R_{Mol}(t) = \text{Prob}\{\text{no fails}\} + \text{Prob}\{\text{single fail}\} + \text{Prob}\{\text{double fail}\}$$

$$R_{Mol}(t) = e^{-\lambda t} + F\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t}\left(1+(F-1)\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{2}{F}\right)t}\right) \quad (49)$$

Macro-molecular reliability when DEC strategy is employed is presented in Figure 4-13 while both SEC and DEC strategies are comparatively shown in Figure 4-14; due to the better fault tolerance, the graphs for DEC strategy give better values than those for the SEC strategy. Comparing the graphs corresponding to $\lambda$=0.02 and $F$=8, the memory macro-molecule keeps its reliability value over 90% for a period of 28.4 million hours (SEC) and 63.3 million hours (DEC); the threshold of 50% reliability is reached in our example after 89.8 million hours in the case of SEC strategy, while DEC strategy requires 154.5 million hours of operation time. Extending equations presented in SubSection 4.3.2, the row reliability in case of SEC and DEC strategies, and the macro-molecular reliability, respectively, can be computed as $R_{Row}(t) = (R_{MMol})^{N-s}(t)$, $R_{Mem}(t) = (R_{Row})^{M}(t)$, given by equations 50–53.

$$R_{Row}(t) = \begin{cases} SEC: & \left(e^{-\lambda t} + F\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t}\right)^{N-s} \\ DEC: & \left(e^{-\lambda t} + F\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t}\left(1+(F-1)\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{2}{F}\right)t}\right)\right)^{N-s} \end{cases} \quad (50)$$

$$R_{MMol}(t) = \begin{cases} SEC: & \left(e^{-\lambda t} + F\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t}\right)^{M(N-s)} \\ DEC: & \left(e^{-\lambda t} + F\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{1}{F}\right)t}\left(1+(F-1)\left(1-e^{-\frac{\lambda t}{F}}\right)e^{-\lambda\left(1-\frac{2}{F}\right)t}\right)\right)^{M(N-s)} \end{cases} \quad (51)$$

Overall macro-molecule reliability is represented in Figure 4-15 for an example of memory area dimensions $M$=5, $N$=6, and $s$=2 (these dimensions were chosen for illustrative purpose only and have no particular impact on the reliability functions). The case when no fault tolerance is employed performs worst, while the reliability shows a significant improvement when SEC and DEC strategies are present (values are better for the last case). Differences between $F$=8 and $F$=16 configurations also show significant increase when changing from SEC to DEC fault tolerance strategy.

Due to the fact that the simplifying assumptions when using the protochip concept are no longer used, the equation describing the row reliability results quite complicated and difficult to analyze. Therefore the mean time to failure expression will also result under a complex form (equation 46).

$$MTTF_{Row} = \begin{cases} SEC: & \displaystyle\int_0^\infty \left( e^{-\lambda t} + F\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{1}{F}\right)t} \right)^{N-s} dt \\ DEC: & \displaystyle\int_0^\infty \left( e^{-\lambda t} + F\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{1}{F}\right)t} \left(1 + (F-1)\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{2}{F}\right)t}\right) \right)^{N-s} dt \end{cases} \tag{52}$$

$$MTTF_{MMol} = \begin{cases} SEC: & \displaystyle\int_0^\infty \left( e^{-\lambda t} + F\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{1}{F}\right)t} \right)^{M(N-s)} dt \\ DEC: & \displaystyle\int_0^\infty \left( e^{-\lambda t} + F\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{1}{F}\right)t} \left(1 + (F-1)\left(1 - e^{-\frac{\lambda t}{F}}\right) e^{-\lambda\left(1-\frac{2}{F}\right)t}\right) \right)^{M(N-s)} dt \end{cases} \tag{53}$$

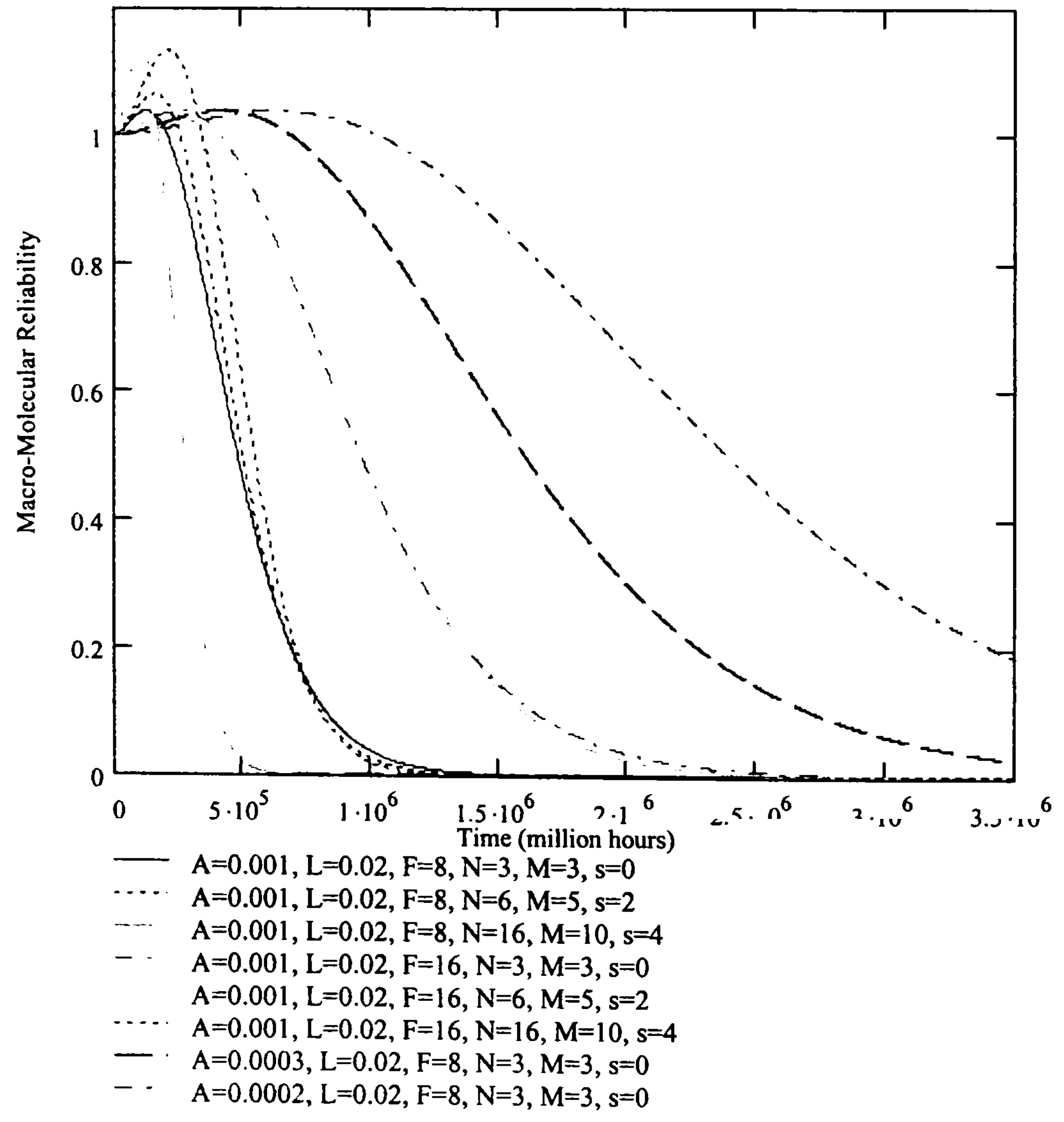


Figure 4-13: Macro-molecule reliability for different configurations.

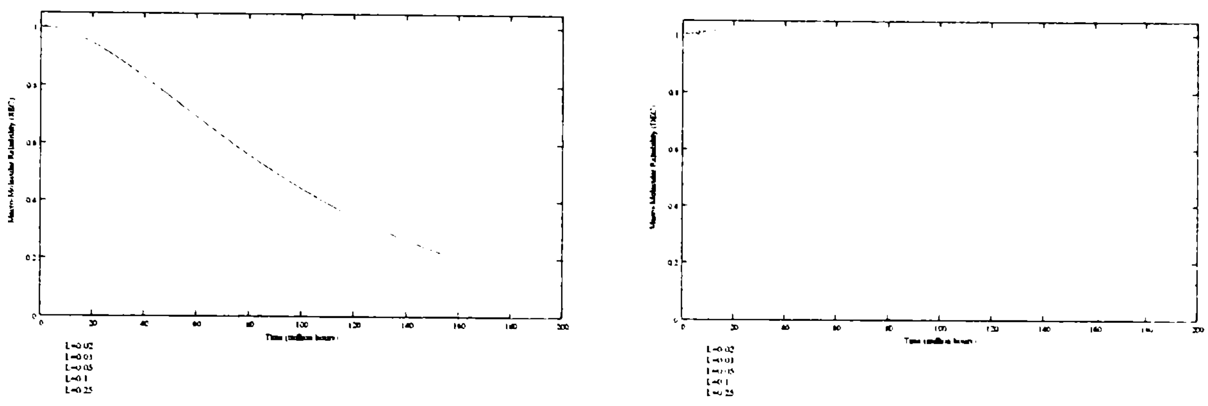


Figure 4-14: Macro-molecular reliability graphs for SEC (left) and DEC (right) strategies.

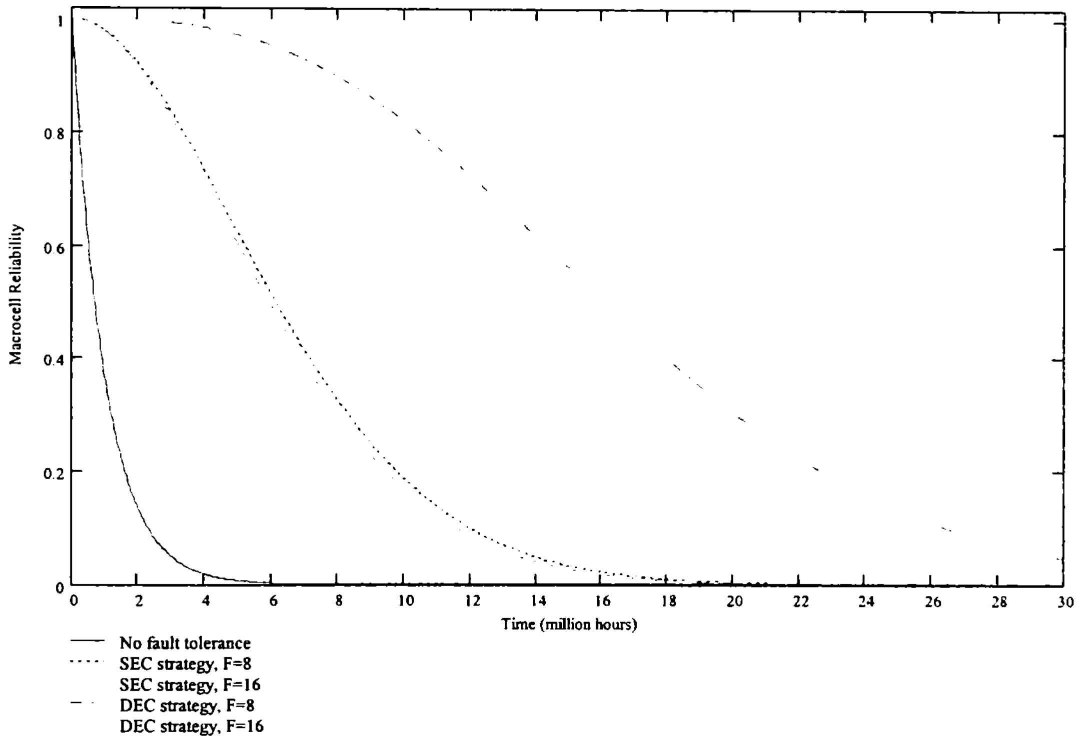


Figure 4-15: Reliability of memory macro-molecule in case when no fault tolerance is in place, and when SEC and DEC strategies are implemented.

## 4.3.6. Cosmic Ray Influence On Reliability

Equations presented in subsections 4.3.4 and 4.3.5 represent the final expressions of the *mean time between failures* (*MTTF*) for a row and for a macro-molecule. While they all look quite complicated, there are some key aspects that may benefit from further insight.

The failure rate λ is an essentially empirical parameter, which can only be determined by extensive measurements. However, exposure to aggressive environments such as cosmic space may reveal that radiation levels affect the values of λ, transforming it from a constant parameter (at sea-level and during standard environment conditions), into a variable (at high altitudes or in outer space and during non-standard conditions). Furthermore, during its entire development period, Embryonics experimented with various implementations, on different platforms, thus making a reliability analysis based on λ alone seem not very useful.

Considering the issues discussed in SubSection 4.4.1, where λ represents the failure rate for a storage flip-flop from inside a molecule, the molecular reliability is $R_{Mol}(t)=e^{-\lambda Ft}$ and a single row of a macro-molecule has a reliability function of $R_{Row}(t)=e^{-\lambda F(N-s)t}$. Because a macro-molecule is assembled by $M$ identical such columns, the overall reliability is $R_{MMol}(t)=R_{Row}^{M}(t)=e^{-\lambda FM(N-s)t}$. The mean time to repair (derived as equation 25) is based on the macro-molecular reliability, but one should also take into account the influence of upsetting particles [98]:

$$MTTF_{MMol}=\min\left(\int_0^\infty R_{Row}^{M}(t)dt,\ \Delta T_{med}\right) \tag{54}$$

where $\Delta T_{med}$ is the mean period between two consequent upset events inside the macro-molecular area. This interval is of yet difficult to be determined as it depends on a number of parameters such as the particle flux, energy levels and others [162].

Subsequently, if the same meaning for parameter λ is preserved and considering SubSection 4.4.4.1, which provides an analysis of macro-molecular reliability when single errors are tolerated, the reliability function for a macro-molecular row becomes:

$$R_{Row}(t)=e^{-\lambda F(N-s)t}+F(N-s)\left(1-e^{-\lambda t}\right)e^{-\lambda(F(N-s)-1)t} \tag{55}$$

Then, the reliability of the entire memory structure (which is composed of $M$ identical rows) results as:

$$R_{MMol}(t)=R_{Row}^{M}(t)=\left(e^{-\lambda F(N-s)}+F(N-s)\left(1-e^{-\lambda t}\right)e^{-\lambda t(F(N-s)-1)}\right)^{M} \tag{56}$$

The influence of upsetting particles over the mean time to failure for a macro-molecule can then be expressed as:

$$MTTF_{MMol}=\min\left(\int_0^\infty R_{Row}^{M}(t)dt,\ 2\Delta T^{*}_{med}\right) \tag{57}$$

If we consider the particle flux as being isotropic, then it is directly proportional with the macro-molecule's storage surface *FM(N-s)* and the interval between two upsetting events happening on the same macro-molecular row ($\Delta T^{*}_{med}$) is larger than the same interval on the entire macro-molecule ($\Delta T_{med}$) by a factor of M [98]:

$$\Delta T^{*}_{med}=M\Delta T_{med} \tag{58}$$

Examples presented previously have taken into account different macro-molecule configurations; while the dimensions considered appear to have reasonable values, they may prove to be far too small for real-life genetic applications. It would therefore be more difficult to estimate accurately the parameters involved, and therefore the overall reliability for such a large macro-molecule, even by employing the *protochip* concept [5].

A reliability analysis should also take into consideration a model of error appearance inside an embryonic memory structure, and, if possible, suggest means of implementing fault tolerance without any disturbing side effects with respect to mechanisms already implemented by Embryonics. Numerous techniques of achieving fault tolerance, covering a broad range of specific applications, are readily available in the literature; however, attempting to bring improvements to an existing project (and Embryonics is such a case) greatly affects the existing choices due to critical effort or costs required.

Unfortunately, when considering soft fails, there is no such model available (or at least, not yet), as understanding the causes and modeling the soft fails (which were argued upon in SubSection 4.2.) is a hot field of research. Phenomena involved in affecting memory cells seem to have a stochastic appearance and therefore an accurate model should involve specific probability/likelihood estimations.

## 4.4. FTRAM-MuxTreeSR: Fault Tolerant Macro-Molecules

The self-repair mechanisms present in Embryonics stretch over the first two base levels of organization (see Figure 2-2), with the first as a reconfiguration process at the molecular level and the second as a reconfiguration at the cellular level. This hierarchical approach of self-repair allows a flexible and efficient way of tolerating faults: its flexibility lies in the capacity of reconfiguring according to different severity levels (first addressed being the least severe faults, represented by faulty molecules, followed by addressing those most severe, represented by faulty cells), while its efficiency comes from inflicting minimal resource loss through reconfiguration (it is less expensive to replace just one faulty molecule than the entire cell to which it belongs).

If functional stress proves harmful to electronic devices in time, memories are in fact in no way an exception [34, 162]. Even if the nature of the information stored in a memory chip is static, i.e. the binary configuration from inside the memory does not change over a reasonably long period of time, this does not mean it cannot be distorted. As argued in subsection 4.2, studies over how radiations affect semiconductor devices in general (including memory devices) have dramatically grown since their beginning in the 1980s and constitute a hot field of research today. Electronic devices shrinking coupled with human expansion into space, the exploration and manipulation of hazardous environments, are bound to experience important doses of radiation, thus requiring protective techniques in order to maintain good functionality of devices and data integrity. Since bio-inspiration aims at endowing artificial machines with the highest degrees of robustness, as encountered in nature, Embryonics should not face potential applications with incomplete self-repairing mechanisms. Therefore, a form of fault tolerance should complement its memory mode features [98].

Increased reliability (and therefore dependability) may be achieved by using two fundamental techniques. *Fault prevention* (also known as fault intolerance) acts towards eliminating all possible faults at the initial moment and is already present in Embryonics: the initial self-testing phase [128] (see Figure 3-30, phase

2) reconfigures the system so that both ribosomic and operative genomes be charged into a fully operational structure. *Fault tolerance*, on the other hand, allows valid computations, even in the presence of faults, by employing redundancy.

Although covering a large area of possible faults, the self-testing and repairing mechanisms of the RAM-MuxTreeSR design expand only over the functional unit (*FU*) of each molecule, thus leaving the part used for the storage of the operative genome (*CREG*) with virtually no data protection (both resources are shown at the molecular level in Figure 2-2). Since the genome governs over functionality, any error at this level would ultimately manifest as a functional failure. Should an error be detected inside a memory molecule – though such a detection alone represents a separate issue – by triggering the same reconfiguration process described in SubSection 3.7, the respective molecule "dies" and its stored data is transferred by using spares. Because in this case the transfer involves erroneous genetic data (as opposed to functional attributes), the final result is an activated spare molecule behaving *exactly* as the faulty one it replaces. Instead of fault recovery, the final outcome only wastes molecular resources.

It is hardly surprising the presently-employed self-repairing mechanism cannot cope with memory structures since fault tolerance techniques are different for memories, which require special coding in order to localize and correct occurring errors. We will present an implementation of such techniques in order to extend the robustness degree in Embryonics by adding self-repairing capabilities to its memory arrays. Though a less than accurate description, we will call these macro-molecules *fault tolerant*, simply because bringing self-repairing features makes them become tolerant towards soft fails.

### 4.4.1. Error Correcting Coding Techniques

When considering adding a degree of fault tolerance to a memory area one has to choose between recovering after an upset event occurred or just detecting it; limiting the process to detection of errors only is of course less costly than further steps required by the correction, which includes pinpointing the place where the error appeared. In some cases the correction may even not be possible, because of various limitations such as operating speed, space constraints, or even the lack of a mechanism of localizing the error.

However, due to special features present in the Embryonics design, some more insight into the matter is necessary. When an error is detected than the reconfiguration mechanism is triggered, the faulty molecule is marked as being "dead" and a spare molecule takes its place; one should not forget that for the moment all is happening from a functional point of view. But should an error be detected inside a molecule operating in any of the memory modes, let us suppose the same reconfiguration process would, somehow, be triggered. Similarly, this would mean the "death" of the respective molecule and the transfer of its role by using spares. But because of the transfer of the CREG contents from the faulty molecule to a spare one, the final result is the faulty molecule "dies" and the spare one starts behaving *exactly* like the faulty molecule before being "killed". Instead of obtaining some positive results, the final outcome is a very similar situation but this time with resources down a molecule. Obviously, error correcting is a must if fault tolerance for the memory is to be achieved. In order to do so, triggering of the reconfiguration mechanism is no more necessary; each error affecting the memory being of the *soft* type (as argued in Section 4.2), it can be corrected merely by inverting the respective binary value in the process of memory shifting (see SubSection 4.3.1).

The process of detecting and localizing a fault can be done with relative ease by using Hamming-class codes [46, 104]. As argued in subsection 4.3, the predominant type of error affecting semiconductor memories is that affecting a single binary unit, with an estimated at least 93% out of the total soft errors experienced. Such a percentage leads to the conclusion that a most suited code for endowing the Embryonics memory with fault tolerance is a Hamming type Single Error Correcting, Double Error Detecting code (or simply SEC-DED) [46, 104]. Such a code has a Hamming distance equal to 3 and requires a number of additional check bits determined from relationship $2^k \geq k+t+1$, which is equivalent to:

$$k = \lceil \log_2(1+k+t) \rceil \tag{59}$$

where $k$ is the number of check bits and $t$ is the number of data bits.

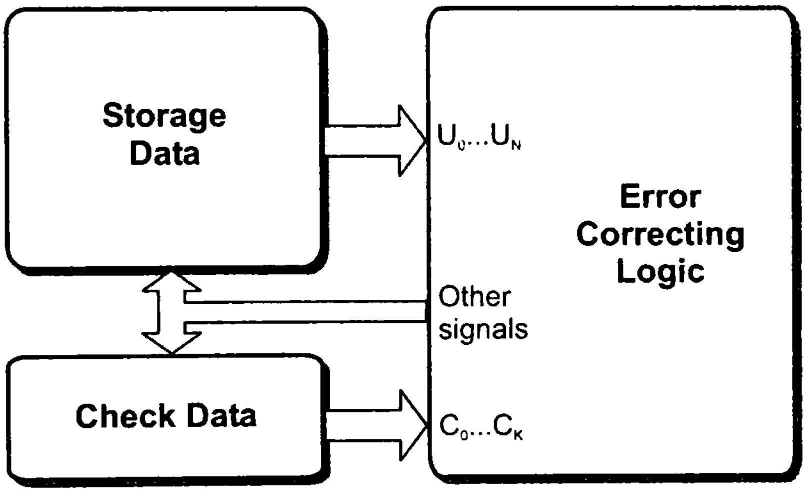


Figure 4-16: Block schematic for a fault tolerant memory in Embryonics.

Adapting Hamming codes to Embryonics would require a structure similar to that presented in Figure 4-16. Molecules operating in memory mode form two structures denoted as *Storage Data* (user data bits $U_0 \div U_N$) and *Check Data* (redundant bits $c_0 \div c_K$) while molecules operating in logic mode are grouped inside a structure called *Error Correcting Logic*. The basic operation of the whole could be described as follows: the user data and the check bits are synchronously shifted (that is, when user data is shifted one step, the check bits are also shifted one step) and at each step a new set of check bits is generated based on the current user data output. The newly generated check bits are then compared with the output from the check data memory, any resulting difference indicating an error. An error syndrome is then derived by applying a XOR operation to both sets of check bits, which will in turn localize the place where the error was detected. The *Error Correcting Logic* structure also generates the required signals in order to recover from a detected fault.

A number of 4 bits of data gives $t=4$ in Equation (59), leading to $k=3$, which means that each set of data will be bound to a set of 3 check bits. This ensures that a single or double fault that may occur is successfully detected, and every single fault can also be corrected. The check bits are generated by a *k*-grade primitive polynomial:

$$G(x) = x^3 + x + 1 \tag{60}$$

There are several ways of implementing such a code [46, 104]. Figure 4-17 shows a block diagram for a code generator based on this polynomial, using a Multiple Input Shift Register (*MISR*). This special purpose register is serially fed with bits from the data codeword (marked as *U(x)* in Figure 4-17) that is to be protected. After a number of clock cycles (3 for our example), the register will compute the redundant bits that are part of the Hamming-type code, as specified by the temporal Equation (61):

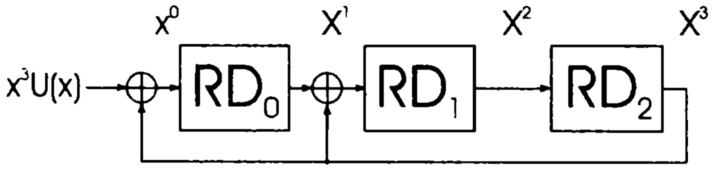


Figure 4-17: MISR Hamming code generator.

$$RD_1(\tau+1) = RD_0(\tau) + RD_2(\tau) \tag{61}$$

From Equation (61) the Hamming matrix can be derived as follows:

$$\begin{array}{r} RD_0 \rightarrow \\ H = RD_1 \rightarrow \\ RD_2 \rightarrow \\ \\ \end{array} \begin{array}{c} \left[\begin{array}{ccc|cccc} 1 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 1 & 1 & 1 \end{array}\right] \\ \begin{array}{ccccccc} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ c_0 & c_1 & c_2 & u_0 & u_1 & u_2 & u_3 \end{array} \end{array} \tag{62}$$

where columns from the left to right represent addresses that can be attributed to $c_0$, $c_1$, $c_2$ (check bits) and $u_0$, $u_1$, $u_2$, $u_3$ (data bits). Therefore we have:

$$\begin{cases} c_0 = u_0 \oplus u_2 \oplus u_3 \\ c_1 = u_0 \oplus u_1 \oplus u_2 \\ c_2 = u_1 \oplus u_2 \oplus u_3 \end{cases} \tag{63}$$

At each clock the syndrome generator will calculate the check bits $c_0$, $c_1$, $c_2$ and will compare them to $c_0'$, $c_1'$, $c_2'$ already stored by the Check Data memory (see Figure 4-16). Then the syndrome is computed from the differences between the newly computed and the stored check bits (see Equation (63)) and corresponds to a column from the Hamming matrix, thus giving the exact address of the faulty bit.

$$\begin{cases} s_0 = c_0 \oplus c_0' \\ s_1 = c_1 \oplus c_1' \\ s_2 = c_2 \oplus c_2' \end{cases} \tag{64}$$

The data correction process requires the exact address be decoded. This is done by employing Equation (64):

$$\begin{cases} c_0' = u_0 \oplus u_2 \oplus u_3 \\ c_1' = u_0 \oplus u_1 \oplus u_2 \\ c_2' = u_1 \oplus u_2 \oplus u_3 \end{cases}, \qquad \begin{cases} u_0' = u_0 \oplus s_0 \cdot s_1 \cdot \overline{s_2} \\ u_1' = u_1 \oplus \overline{s_0} \cdot s_1 \cdot s_2 \\ u_2' = u_2 \oplus s_0 \cdot s_1 \cdot s_2 \\ u_3' = u_3 \oplus s_0 \cdot \overline{s_1} \cdot s_2 \end{cases} \tag{65}$$

Let us consider the following example of a data word $U = u_0u_1u_2u_3 = 1101$. Using Equation (62) the corresponding check bits result $C = c_0c_1c_2 = 000$. If an error affects the useful bits, transforming $U$ in $U' = 1001$, then the newly computed check bits will result as $C' = 011$. Any non-zero syndrome indicates the presence of an error. In this case the result is $S = s_0s_1s_2 = 011$, which constitutes an address in the Hamming matrix showing that the affected position corresponds to $u_1$. If the error appears in the check bits zone, for instance $C' = 010$, then the corresponding syndrome will indicate that the erroneous bit corresponds to $c_1$. Correcting an error is done by simply inverting the value of the erroneous bit, as shown in Figure 4-18.

The Hamming code presented in this example works well for single errors; however, the situation when two or more binary positions are affected is also possible. In this case the existence of an error will be successfully indicated, but an attempt to correct the error will mistakenly correct a single binary position, instead of the two erroneous bits. Let us assume the same data word is affected by a double error, transforming $U$ in $U' = 1000$. Then the resulting syndrome will be $S = 110$, mistakenly leading to the "correction" of the binary position corresponding to $u_0$ (Figure 4-19), which was not even affected by the error! A similar case is induced by the situation when bits $u_1$ and $c_1$ are affected by error, the resulting syndrome will successfully indicate the presence of an error, but will mistakenly indicate the "correction" of bit $c_2$!

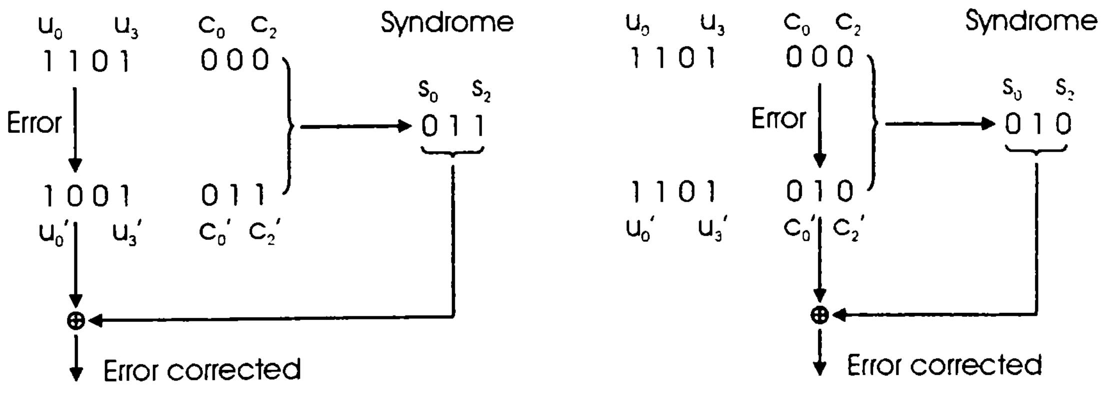


Figure 4-18: Single error detection and correction using Hamming codes.

From those mentioned above one can note as a downside that the code presented as an example cannot correct double errors. Control bits in this case occupy almost 43% of the total memory space, which may perhaps be considered too high a percentage to justify the existence of the code; however, as the number of data bits increases, the proportions change in the favor of data (for instance, 16 bits of data require 5 check bits which means less than 32%).

Integrating a code capable of detecting and correcting single errors into the Embryonics project, although justified by their frequency (see subsection 4.3), may be considered only as a strategy of self-repair at the molecular level. The reason is errors that affect single data bits can be recovered from at the molecular level, but there are no actions to be taken should a multiple error affect the protected data. Such an event would typically trigger the self-repair process at the superior level (the cellular level), however, the triggering decision lies beyond this code's capabilities.

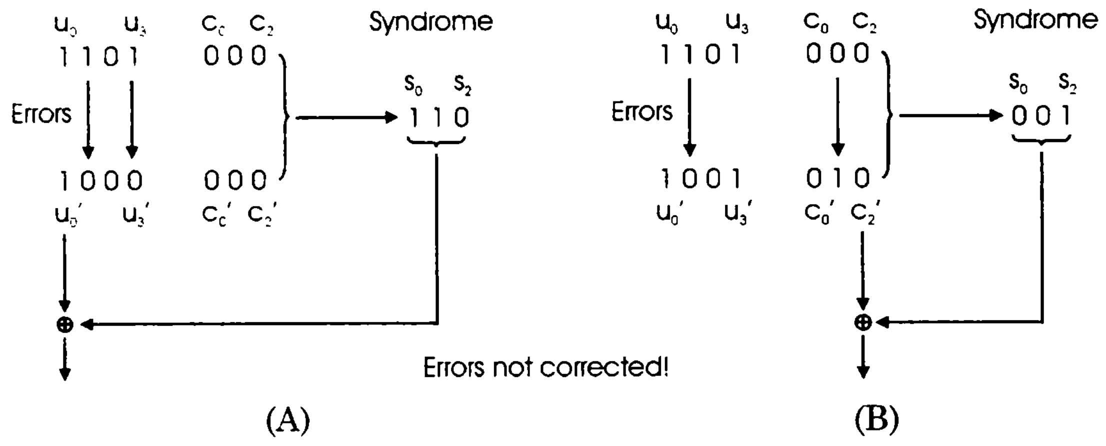


Figure 4-19: Double error detection but dummy correction of $u_0$ (A) and $c_2$ (B).

### 4.4.2. Single Error Correction Codes with Double Error Detection

In general, the situation when two or more binary positions are affected is also possible, although it is far less probable [5, 34]. If such is the case, there are two scenarios that have to be taken into consideration:

- if the error produces a syndrome that is recognizable as an address (a column) in the Hamming matrix, then the presence of the error will be successfully detected and corrective measures may be allowed;
- if the error produces a syndrome that is not recognizable as a column in the Hamming matrix, then the presence of the error is shown by the non-zero syndrome, but corrective measures cannot be taken.

The successful detection of a multiple fault assumes that the resulting syndrome does not fall over a matrix column such as it would falsely indicate a single error of a data or check bit. It is possible that a certain double error produces a syndrome typical to a single error; though the error's presence is promptly reported, there are no corrective measures to be taken.

Integrating a code capable of correcting single errors and also detecting the presence of double errors into the Embryonics project may be preferred, since the decision of triggering the self-repair at the cellular level can be actually assimilated to the successful detection of a multiple error. Therefore, if the macro-molecular fault tolerance levels are exceeded at the molecular level, that is, an unrecoverable (multiple) error has occurred, the detection of such a scenario ensures the activation of the KILL signal (see subsection 3.7), which will then trigger reconfiguration processes at the cellular level.

The previous Hamming code, capable of successfully detecting and correcting any single error, had a codeword distance $d_{min} = 3$ (that is, the difference between any two codewords lies in at least 3 bits). Detecting any double fault that may affect the data requires $d_{min} = 4$, which modifies the Hamming matrix as follows [44, 46, 104]:

$$H = \begin{array}{r} RD_0 \rightarrow \\ RD_1 \rightarrow \\ RD_2 \rightarrow \\ \\ \\ \end{array} \begin{array}{cccc|cccc} 1 & 0 & 0 & 0 & 1 & 0 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 1 \\ \hline 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ c_0 & c_1 & c_2 & c_3 & u_0 & u_1 & u_2 & u_3 \end{array} \tag{66}$$

As can be seen from Equation (65), the original Hamming matrix received an additional column (marked as *c3*, between the dashed lines) and an additional line, therefore bringing the code generated to a minimal codeword distance equal to 4. The equations used in order to compute the syndrome and to decode the error's address change as follows [104]:

$$\begin{cases} s_0 = c_0 \oplus c'_0 \\ s_1 = c_1 \oplus c'_1 \\ s_2 = c_2 \oplus c'_2 \\ s_3 = \sum_{i=0}^{3} {}^{\oplus} u_i \oplus \sum_{j=0}^{3} c'_j \end{cases} , \quad DDE = s_0 s_1 s_2 + \overline{s}_3 \tag{67}$$

$$\begin{cases} c'_0 = u_0 \oplus u_2 \oplus u_3 \\ c'_1 = u_0 \oplus u_1 \oplus u_2 \\ c'_2 = u_1 \oplus u_2 \oplus u_3 \\ c'_3 = \sum_{i=0}^{3} {}^{\oplus} u_i \oplus \sum_{j=0}^{2} c_j \end{cases} , \quad \begin{cases} u'_0 = u_0 \oplus s_0 \cdot s_1 \cdot \overline{s}_2 \cdot s_3 \\ u'_1 = u_1 \oplus \overline{s}_0 \cdot s_1 \cdot s_2 \cdot s_3 \\ u'_2 = u_2 \oplus s_0 \cdot s_1 \cdot s_2 \cdot s_3 \\ u'_3 = u_3 \oplus s_0 \cdot \overline{s}_1 \cdot s_2 \cdot s_3 \end{cases} \tag{68}$$

Let us consider the example from Figure 4-19, where the lack of a dedicated parameter used for indicating the presence of a double error led to a dummy correction. For the same data bits $U = u_0 u_1 u_2 u_3 = 1101$, the redundant code bits result as $C = c_0 c_1 c_2 c_3 = 0001$ (the only difference from the case from Figure 4-19 being the existence of an additional check bit *c3*). The final codeword will therefore be $UC = 11010001$. The situation when the codeword is affected by a single error will not be discussed since the modified Hamming matrix does not affect the recovery process described in the previous subsection. Therefore, let it be considered a double error affecting this codeword through bits *u1* and *u3* as $UC' = 10000001$. At each clock cycle, the redundant bits are re-computed and compared to those from within the codeword; in this case the redundant bits compute as $C = c_0 c_1 c_2 c_3 = 1101$, with the resulting syndrome being $S = s_0 s_1 s_2 s_3 = 1100$. While the non-zero syndrome indicates the presence of an error, equation (67) computes *DDE* (for *Detection of a Double Error*) as:

$$DDE = s_0 \cdot s_1 \cdot s_2 + \overline{s}_3 = 1 \cdot 1 \cdot 0 + \overline{0} = 1 \tag{69}$$

therefore pointing to the double error scenario (see Figure 4-20).

ECC codes constitute a standalone direction of research. While the scope of this thesis focuses on expanding upon the Embryonics project rather than proposing an incursion into the field of codes, techniques of correcting single errors and detecting double errors can be successfully implemented within the Embryonics platform. However, the discussed example may be considered less than usual, because it makes use of all columns from the Hamming matrix. In real world applications, if single error correction is the main target, then the size of the codeword will not require a full use of the Hamming matrix; on the other hand, if the detection of all double faults is desired, this will perhaps prohibit a reasonable implementation. Therefore, a reasonable goal (in terms of complexity and necessary resources for implementation within Embryonics) would be not only to correct any single error, but to also detect the maximum number of double faults possible.

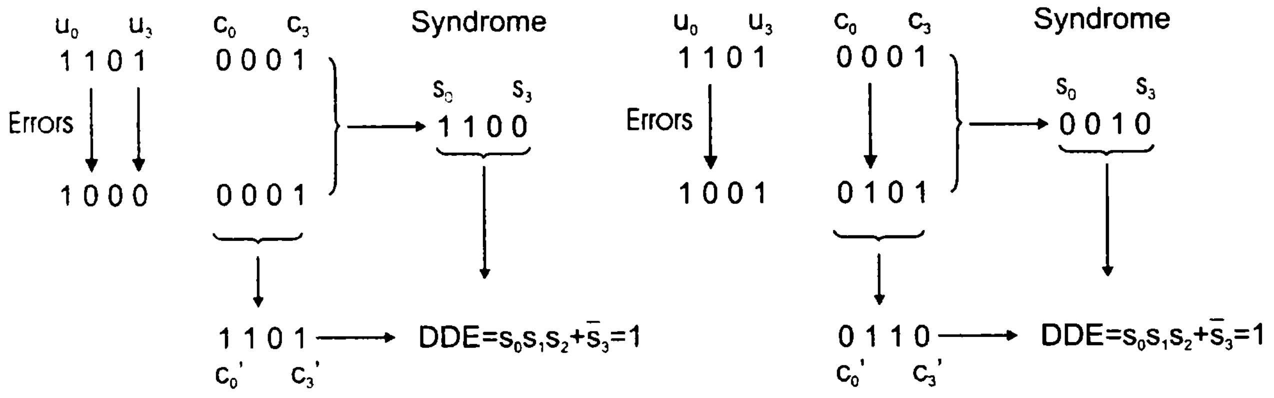


Figure 4-20: Double error detection using modified Hamming codes.

### 4.4.3 Possible Error Scenarios in a Macro-Molecule

Because the macro-molecular architecture consists of both logic- and memory-operating molecules, there exists a range of error situations that may occur.

First, it is possible that a number of errors affect only logic-operating molecules situated outside the macro-molecular structures, therefore exerting no influence over the storage data. Such a situation requires the activation of self-repairing mechanisms implemented in logic molecules [128], where both off-line and on-line self-testing strategies are employed [45]. The configuration register *CREG* is self-tested at run-time (therefore making it an off-line testing strategy), any stuck-at-type error triggering the reconfiguration at the molecular level, thus the faulty molecule being eliminated from the array, as described previously in Section 3.7. Since the *CREG*'s role (when in logic mode) is to store the binary configuration for the switch block SB and some additional connections concerning the functional unit *FU* [100, 128], this off-line strategy was chosen to be implemented, based on the assumption that the probability of damage to the *CREG* data is minimized by its property of being static. The functional unit *FU* uses resource replication and majority voting in order to implement an on-line self-testing strategy; any error that occurs dynamically can therefore be corrected.

Second, it is possible that errors occur only inside de storage area of a macro-molecule. In this case, information stored by the memory molecules inside the configuration register *CREG* is shifted every clock cycle, requiring an on-line self-testing and repairing technique be implemented in order to recover from faults. As long as the fault does not overcome the capacity of the ECC code of recovering damaged data (that is, each row can sustain and recover from a single error), the result is a successful correction of the damaged data bit. If the ECC code cannot handle the occurring errors, the macro-molecule's data cannot be recovered and this results in the failure of the entire cell. Therefore the reconfiguration process is triggered at the higher, cellular level (see Figure 3-25).

Another possible scenario affecting a macro-molecules takes place when the presence of an unrecoverable error is by itself reported erroneously; this is to say that no error is physically present inside the memory array, but the Error Correcting Logic mistakenly reports the existence of an error and tries to initiate corrective measures. This scenario is made possible by the fact that combinational logic is not completely resilient to the damaging effects of soft fails, which are not covered by the self-testing procedures employed by molecules operating in logic mode.

Taking into consideration the example from subsection 4.4, the implementation of the ECC code requires an Error Correcting Logic structure that may seem too large in comparison with the macro-molecule's overall size (the storage data macro-molecule and the three redundant data macro-molecules), and therefore more prone to errors. It is therefore possible that the larger structure (the Error Correcting Logic) would "detect" the presence of an error inside the macro-molecular ensemble, even if there is no actual physical error. Whenever a couple of checker structure-checked structure exists, the case of the checker becoming faulty is always possible, even if the purpose is to have it much less error-prone than the checked structures. If such a situation arises, regardless of corrective measures, it is the result of *an error*, even if that particular error was not detected successfully. There are two ways out of this scenario, after corrective measures are initiated:

- Because there was no physical error inside the memory area, at least one bit worth of data will be altered. When the corrective process is finished, the result is a macro-molecule that contains altered data, but the cell retains a certain level of overall functionality.
- The "detected" error is perceived as non-recoverable and therefore the *KILL* process is initiated. One way of dealing with such a situation would be to have the Error Correcting Logic effectively stop the data shifting process inside all of the macro-molecule's components; since the data content has been compromised, there is no reason an external entity should continue the execution of the stored genetic program. This strategy would allow for a strict containment of an error-affected memory while preserving the logic functionality of the cell (and of any other macro-molecules not having been affected by the error). The other way (which we chose for implementation) would be to have the Error Correcting Logic trigger the *KILL* mechanism, and thus deactivate an entire cell column. In this case, there are no active entities "crippled" by errors and the use of the second level of reconfiguration in Embryonics is preserved (as opposed to the first scenario, which does not benefit from the hierarchical architecture in Embryonics). The result is the death of the entire cell.

Whether the first or the second way is a better choice if such an erroneous situation occurs may constitute a subject of debate, since it is difficult to say if having a functional, but crippled, cell has any advantages over not having that cell at all. Nature itself encounters a similar problem, since cellular mutation does not necessarily mean the organism becomes non-viable; however, altered cellular information often leads to damaging effects and illnesses, such as cancer.

In order to ensure the possibility of cellular self-repair in case of memory structures also, we decided to implement the second strategy, thus extending the original trigger of cellular self-repair from logic molecules to also work with macro-molecules. The *UNKILL* mechanism remains virtually the same. Its role is to re-initialize the entire cellular structure bottom-up with the original genome, meaning that the initial configuration is again loaded into each molecule. Both the macro-molecular structure (including storage data) and the logic ensemble that implements (but is not limited to) the Error Correcting Logic are coded within the genome. If a majority of soft fails accumulated and triggered the reconfiguration at the cellular level, the result of the *UNKILL*ing process is an architecture free of errors, and with a minimum of resources deactivated (those affected by permanent faults).

### 4.4.4. Architecture of a FTRAM-MuxTreeSR Molecule

Based on the notions introduced by previous sections, we present the architecture of a complete fault tolerant memory structure; for this purpose, we will consider the theoretical approach from subsection 4.4.1 where the data word to be protected is 4-bit-wide. One of the possible scenarios is for the data bits being provided by a single macro-molecule built of 4 columns of memory molecules. Another plausible scenario assumes that data bits are independently provided by 4 separate macro-molecules. Choosing the first scenario over the second one for our example should have virtually no influence on the architectural complexity, while giving the same perspective on the technique used to manipulate the resulting structure. Of course, when larger memory structures are needed, several macro-molecules may be protected by the same code, therefore the second scenario should be preferred in this case.

#### 4.4.4.1 Architecture of a SEC Macro-Molecule

Applying a single error correcting code requires for every 4 data bits 3 additional check bits at each clock cycle. This actually imposes the existence of 3 additional memory macro-molecules built as independent single columns, each providing one check bit. If we consider the storage capacity of the data macro-molecule as 48 words, which means a structure of 3x4 molecules operating in the long memory mode (see SubSection 3.5.2.3) then each check column will contain 3 molecules operating in the same (long memory) mode. Summarizing, the final structure will consist of 3 blocks (pointed out in Fig. 4-23) [98]:

- One macro-molecule for genome data storage, denoted as Genome Memory. For this example it is a single 3x4 macro-molecule.
- A second memory structure, denoted as Control Memory, delivering the control data used by the ECC coding. For this example it is made of 3 columns, each composed of 3 memory molecules.
- The syndrome generation and error correction logic, a structure built of molecules operating in logic mode, which is also responsible with feeding the memory structures with the necessary control signals.

Moreover, the 3 blocks necessary for building a fault tolerant memory structure may not be the only components of the cell, therefore other molecules, operating in any mode may lie inside the cellular area.

The implementation of the Hamming code requires a decision with respect to the structure of the Control Memory, which could either make up for a second macro-molecule or could leave each control memory column operate independently. The first situation unifies the control memory into a single structure that requires a detailed analysis of a macro-molecule's data path specifics [98] in order to operate correctly (see Figure 4-21). It is important the two memories maintain synchrony even after data permutation, in order to preserve the consistency of between Equation (70) and Equation (71), situation settled by the theorem enounced next [95].

The second situation relies on using separate data macro-molecules, each delivering one bit worth of data. Regardless of their horizontal and vertical dimensions, data macro-molecules need to match their storage capacity, while control data will be delivered by independent memory columns, properly dimensioned so as to match the storage space of the genome data to be protected. An example will be presented in Subsection 4.4.5.

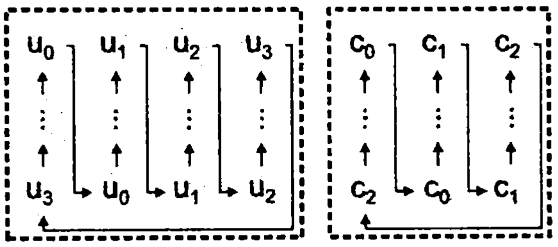


Figure 4-21: Data shifting inside Genome and Control Memory for a (7,3) Hamming code implementation.

***Theorem:***

A (7, 3) Hamming-type SEC code can be implemented by two macro-molecules composed of 4 and, respectively, 3 columns, as data provided will remain in synchrony through shifting processes as described in Section 4.3.1.

***Proof:***

Control data is computed at time *t* according to Equation (70), where $c_{2\text{-}0}$ are the redundant bits output by the Control Memory and required for error correction:

$$\left\{\begin{array}{l} c_0 = u_0 \oplus u_2 \oplus u_3 \\ c_1 = u_0 \oplus u_1 \oplus u_2 \\ c_2 = u_1 \oplus u_2 \oplus u_3 \end{array}\right|_t \qquad (70)$$

The word $u_0u_1u_2u_3c_0c_1c_2$ (Figure 4-22), which was read at time *t* by the Error Correcting Logic (or *ECL*), will shift into $u_3u_0u_1u_2c_2c_0c_1$ at time *t*+1. If the data macro-molecule has a vertical dimension of *M* lines (each molecule storing *F* data bits), then at time $t+1+F(M-1)$ (which is necessary for the data to travel from the bottom to the output ports situated at the top of the macro-molecule [98, 102]) the *ECL* will read $u_3u_0u_1u_2c_2c_0c_1$. Computing the control data for this new configuration is done by Equation (71), the identity with Equation (70) confirming the two macro-molecules (data and control) remain in synchrony for a (7,3) Hamming code [95].

$$\left\{\begin{array}{l} c_2 = u_3 \oplus u_1 \oplus u_2 \\ c_0 = u_3 \oplus u_0 \oplus u_2 \\ c_1 = u_0 \oplus u_1 \oplus u_2 \end{array}\right|_{t+1+F(M-1)} \qquad (71)$$

The situation however changes when a (15,4) Hamming code is employed (Figure 4-23). The Hamming matrix produced by the polynomial $G(x)=x^4+x+1$ is given by Equation (72) while the codewords at time $t$ and $t+1+F(M-1)$ are given by Equations (73) and (74), respectively.

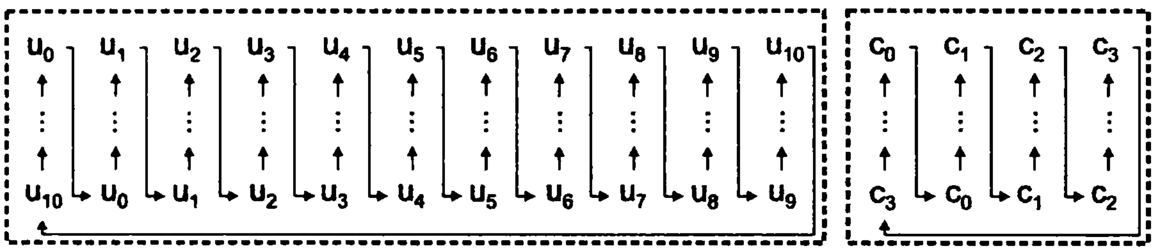


Figure 4-23: Data shifting inside Genome and Control Memory for a (15,4) Hamming code.

The synchrony between Genome and Control Memories is lost as shown by the comparison between Equation (73) and Equation (74): the equations for $c_0|_t$ and $c_0|_{t+1+F(M-1)}$ reveal differences. However, it may be possible that synchrony be maintained by choosing a different coding or by using data words that are less than 11-bits wide.

$$H = \begin{array}{c} RD_0 \rightarrow \\ RD_1 \rightarrow \\ RD_2 \rightarrow \\ RD_3 \rightarrow \\ \\ \end{array} \begin{array}{c} \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix} \\ \begin{array}{ccccccccccccccc} \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow & \uparrow \\ c_0 & c_1 & c_2 & c_3 & u_0 & u_1 & u_2 & u_3 & u_4 & u_5 & u_6 & u_7 & u_8 & u_9 & u_{10} \end{array} \end{array} \qquad (72)$$

$$\left\{ \begin{array}{l} c_0 = u_0 \oplus u_3 \oplus u_4 \oplus u_6 \oplus u_8 \oplus u_9 \oplus u_{10} \\ c_1 = u_0 \oplus u_1 \oplus u_3 \oplus u_5 \oplus u_6 \oplus u_7 \oplus u_8 \\ c_2 = u_1 \oplus u_2 \oplus u_4 \oplus u_6 \oplus u_7 \oplus u_8 \oplus u_9 \\ c_3 = u_2 \oplus u_3 \oplus u_5 \oplus u_7 \oplus u_8 \oplus u_9 \oplus u_{10} \end{array} \right|_t \qquad (73)$$

$$\left\{ \begin{array}{l} c_3 = u_{10} \oplus u_2 \oplus u_3 \oplus u_5 \oplus u_7 \oplus u_8 \oplus u_9 \\ c_0 = u_{10} \oplus u_0 \oplus u_2 \oplus u_4 \oplus u_5 \oplus u_6 \oplus u_7 \\ c_1 = u_0 \oplus u_1 \oplus u_3 \oplus u_5 \oplus u_6 \oplus u_7 \oplus u_8 \\ c_2 = u_1 \oplus u_2 \oplus u_4 \oplus u_6 \oplus u_7 \oplus u_8 \oplus u_9 \end{array} \right|_{t+1+F(M-1)} \qquad (74)$$

In case of implementing a SEC-DED Hamming code, the necessary structures are those indicated by the ennounced lemma [95].

***Lemma:***

A (7, 3) Hamming-type SEC-DED code can be implemented by three macro-molecules composed of 4, 3 and, respectively, 1 columns, as data provided will remain in synchrony through shifting processes as described in Section 4.3.1.

***Proof:***

Similar to the proof of theorem, the codeword read by the Error Correcting Logic block at time $t$ is $u_0u_1u_2u_3c_0c_1c_2c_3$. The first macro-molecule provides data bits $u_0u_1u_2u_3$, while the second macro-molecule provides control bits $c_0c_1c_2$, the third macro-molecule providing only control bit $c_3$. Therefore, the codeword transforms at time $t+1+F(M-1)$ into $u_3u_0u_1u_2c_2c_0c_1c_3$. As proven by the theorem, the synchrony is maintained between macro-molecules delivering data bits $u_0u_1u_2u_3$ and control bits $c_0c_1c_2$. Because the third macro-molecule, delivering control bit $c_3$, is actually a memory column, at time $t+1+F(M-1)$ it will deliver the same bit $c_3$, therefore maintaining the synchrony between all macro-molecules [95].

### 4.4.4.2 Architecture of a SEC-DED Macro-Molecule

The dotted rectangles (Figure 4-24) represent the confinement area for data pertaining to the same memory structure and also indicate the existence of data access ports (the data memory has four data access ports while each check memory column has one data access port), while the black arrows mark the presence of control signals for each memory structure. Essentially, the memory control signals are divided in two categories:

- the *HOLD* signal enables the data shifting process; commanding this signal assumes that the molecule from the west of the left corner (LC) of a macro-molecule drive the required value from its *SIN* entry to its *EOUT* output port; active on logic 0, was described in Section 3.6. Each memory has its own *HOLD* signal, denoted as *MHi*, where $i \in \{D,\ C_{2:0},\ C_3\}$ in Figure 4-24.
- the *INVi* (from *INV*ert, $i \in \{D_0 - D_3,\ C_0 - C_3\}$) signals are distributed on the entire south border of each macro-molecule and each affects the corresponding southernmost molecule; when on logic 0, data entering that molecule is complemented for as long as the signal retains its value.

A more detailed look upon a memory macro-molecule reveals the way control signals are routed (Figure 4-25). The internal datapath and *HOLD* signal spreading across the macro-molecules (which were presented in Sections 3.4.4 and 3.6) are shown as differently dotted lines. The *INV* signals are to be used in conjunction with the corresponding memory *HOLD*, as they govern any macro-molecule's operation. The main difference between macro-molecules without and with ECC codes corresponds to the presence of the *INV* signal, which is routed directly to the input data port of the configuration register *CREG*.

Table 4-6 presents possible use of these signals on the *Genome Data* macro-molecule. When the *HOLD* signal is on logic "1", the data shifting for the entire macro-molecule is disabled. Such an operation may be of use when there is a need to attain a certain bias between several macro-molecules in order to execute a NOP (NO Operation) or jump instruction. The execution of the genetic program in living creatures takes place ceaselessly, until the end of the organism's life, as what programmers call an infinite loop. That is the reason we conceived the macro-molecule as a cyclic memory that keeps shifting its data; because our memory stores a genetic program (or part of it), its continuous execution is essential for its "electronic life". However, as engineers, it is desirable to have access to some sort of control of the program execution, despite not having the proper mechanisms that ensure the existence of the conditional branches. Therefore the memory shifting can be stopped by setting the memory *HOLD* signal, a feature that may be of use when several genetic programs, stored by different macro-molecules, are running. At the beginning, they all run in synchrony; that is to say that all macro-molecular data are shifted at the same pace, given by the functional clock's frequency. If, for some reason, one program should wait until another reaches a certain state, then the synchrony may be altered through using the provided access to the *HOLD* signal.

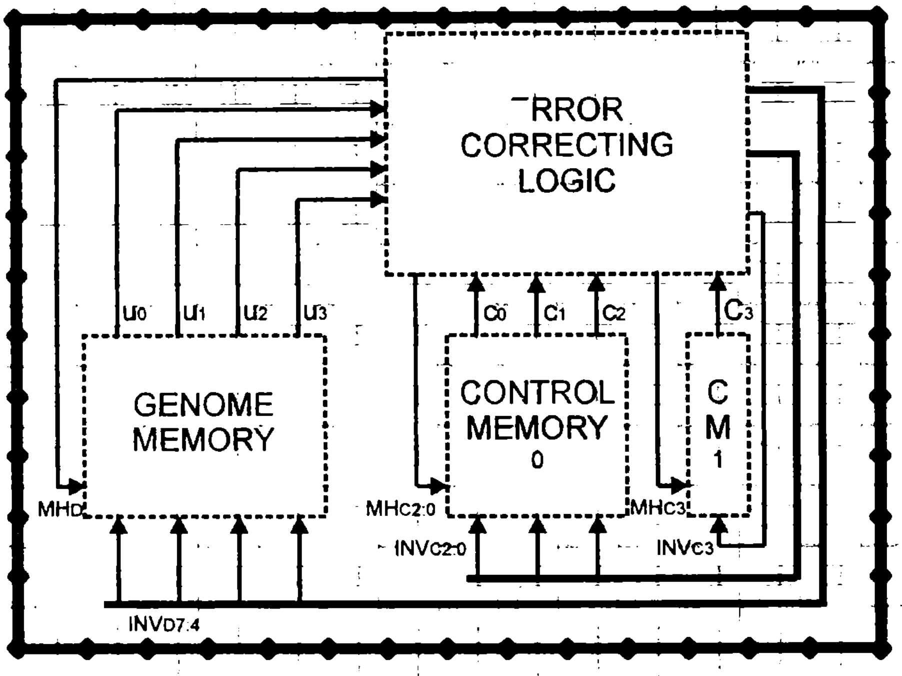


Figure 4-24: Block shematic of a complete cell with a fault tolerant SEC-DED memory structure.

When the *HOLD* signal is active (on logic "0"), the data is shifted inside the macro-molecule at each clock cycle. Depending on the value of the *INVi* signals (also active on logic "0"), data content can be altered for as long as they remain active. This feature can be used effectively to correct damaged data; since the presence of an error requires measuring the data output from the northern border of each macro-molecule, the inversion process may begin one clock cycle later, when erroneous data will have reached the southern border of the macro-molecule. Data entering the memory molecule affected by error reaches the corresponding storage register (*CREG*) after first passing through a *XOR* gate activated by applying the required *INV* signal (see Figure 4-26); therefore, the recovery process consists of obtaining a valid data bit through the controlled inversion of a damaged one.

Signals used in order to implement a fault tolerant macro-molecule are shown in Figure 4-24; however, their proper implementation requires logic molecules to drive them from the originating source to their final destination. Therefore the structure of a cell containing a fault tolerant memory structure consists of the following:

- one (or several) macro-molecules storing the operative genome (or parts of it) denoted as *Genome Memory*;

- the corresponding number of macro-molecules storing the redundant data used for genetic data protection and error recovery, denoted as $CM_i$ (*Control Memory*)
- the necessary number of logic molecules that implement the *Error Correcting Logic* structure
- the necessary number of logic molecules that drive the control signals to and from the Error Correcting Logic
- any additional molecules, external to fault tolerant memory structures.

| $MH_D$ | $INV_{D0:3}$ | Operation |
|---|---|---|
| 0 | 1111 | Normal memory shifting enabled |
| 0 | 0111 | Memory shifting with column 0 inverted |
| 0 | 1011 | Memory shifting with column 1 inverted |
| ⋮ | ⋮ | ⋮ |
| 0 | 1101 | Memory shifting with column n-1 inverted |
| 0 | 1110 | Memory shifting with column n inverted |
| 1 | ..0.. | Memory shifting disabled, memory global kill signal set |

Table 4-6: Control signals for operating the memory

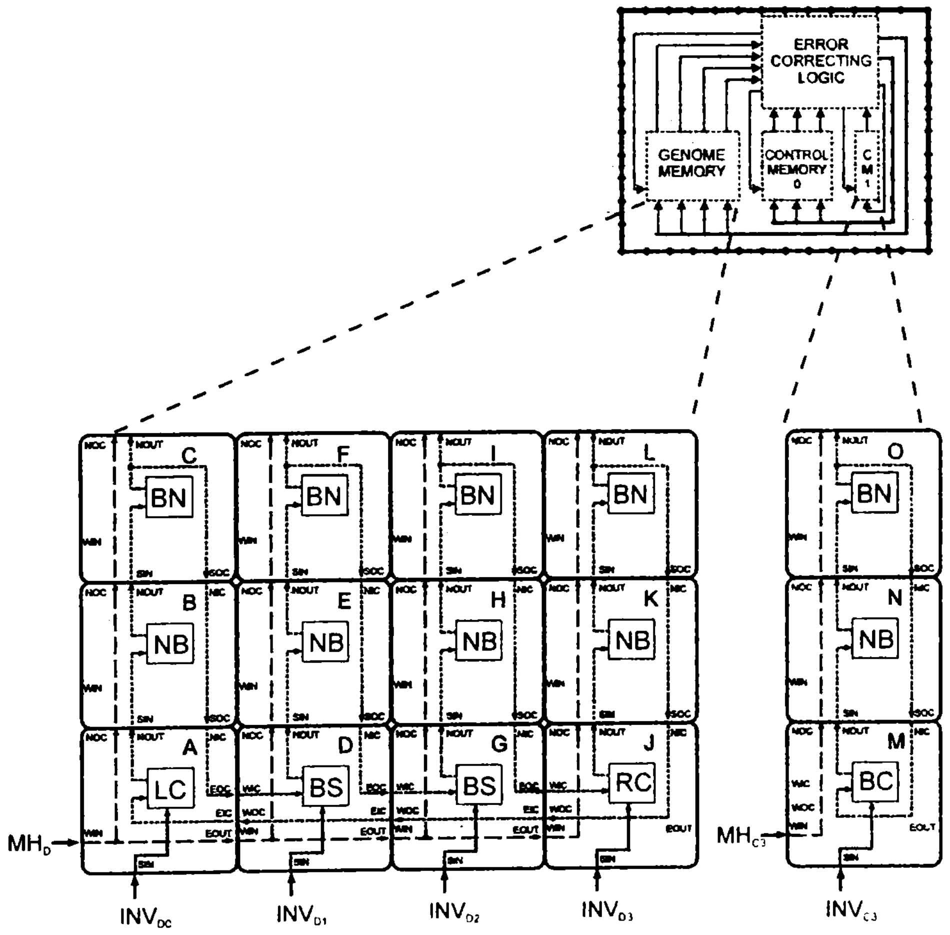


Figure 4-25: Internal routing of memory control signals.

Error recovery is possible through flipping erroneous bits; this process takes place at each molecule from the bottom of a memory structure, where data enters the CREG after being driven through a XOR gate together with the corresponding *INV* signal. Figure 4-26 depicts the datapath for a molecule situated at the south border of a memory column used for storing redundant data: molecule M operates as a *bottom of memory column* (BC) and therefore data that enters through the *north input connection* (*NIC*) can be modified by the *INV* signal by using a XOR gate (depicted as ⊕ in Figure 4-26), this feature being used during the error correction process. The *configuration register* (CREG) can then store corrected (or correct) data.

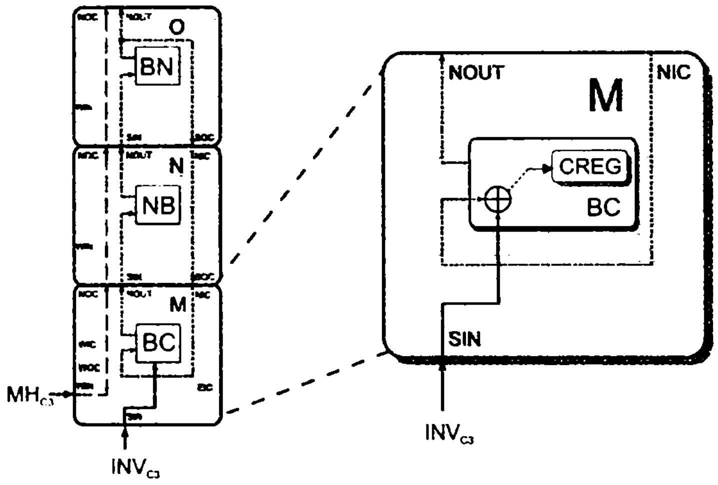


Figure 4-26: Damaged data can be recovered through controlled inversion before entering the configuration register CREG.

An essential part that gives a memory structure its fault tolerance attributes is the Error Correcting Logic block (see Figure 4-24), which computes Equation (66) and Equation (67) and consists of molecules operating in logic mode. Figure 4-27 describes the internal logic of the Error Correction Block, the input signals being $u_{0:3}$ (genome data), $c'_{0:3}$ (newly computed redundant code bits) and $s_{0:3}$ being the syndrome.

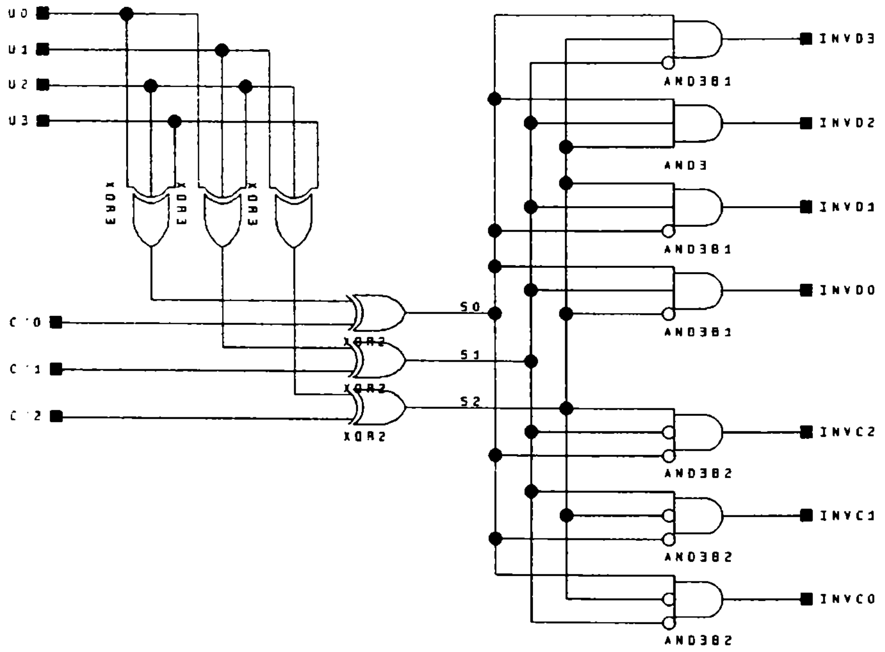


Figure 4-27: Internal schematic of the Syndrome Generation and Error Correction block.

The outputs are the $INV_{D0:3}$ signals (required for correcting the genome data), $INV_{C0:3}$ (required for correcting the redundant data) and the $MH$ signals (which may be used for synchronizing different memory structures or macro-molecules). Due to the fine graininess of the molecular level in Embryonics, the layout of a XOR gate consists of 6 molecules, the bus level and logic level representations being shown in Figure 4-28.

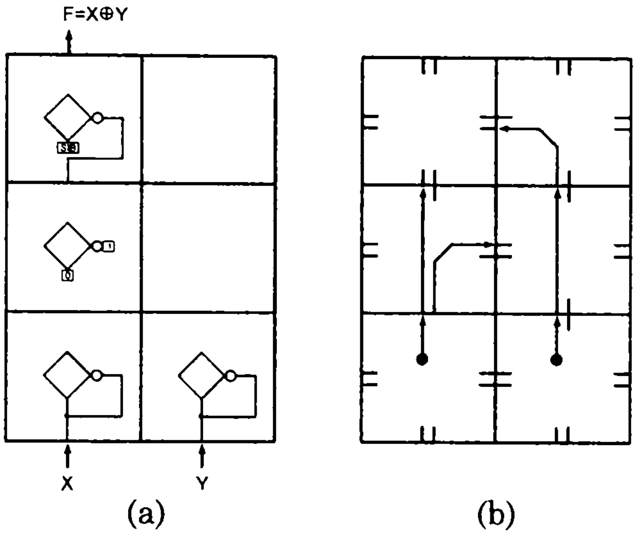


Figure 4-28: Layout of a XOR gate using logic mode operating molecules. (a) Logic level. (b) Bus level.

## 4.4.5. Fault-Tolerant Memory Arrays: An Example

In order to illustrate how memory fault tolerance is achieved by ECC codes implemented onto memory structures in Embryonics, we will go into more details by considering an example (based on the architecture shown in Figure 4-24) of a 4-column genome memory. The initial content of this memory is structured as follows:

- three columns describe the modulo-6 counter presented in section 3.5.2.4 and shown in Figure 3-19;
- whenever the microprogram executes a "do" instruction, an active signal is provided by the fourth column.

The content of both the *Genome Memory* and the *Control Memories* at time $t=0$ (when there are no errors present) is presented in Figure 4-29. All control signals are inactive, that is data shifting is enabled for all macro-molecules $MH_i=0,\ i\in\{D,\ C_0,\ C_1,\ C_2,\ C_3\}$ and there are no inversion signals set $INV_i=0,\ i\in\{D_0,\ D_1,\ D_2,\ D_3, C_0,\ C_1,\ C_2,\ C_3\}$. Data output is shown for each north molecule ($U_{0:3}$, $C_{0:3}$), and values for data entering each south molecule are shown below the configuration register.

Normal operation of the memory implies circling the data through continuous shifting, the situation at time $t+1$ being shown in Figure 4-30. The next moment, a soft error affects molecule $E$ by flipping data bit $E_4$, therefore transforming its data content into 308F, as indicated in Figure 4-31. The error position is marked in Figure 4-31 with a darker background. The fact that an error has occurred cannot be detected (and, by consequence, no correction measures can be taken) until the erroneous bit reaches the nearest data output

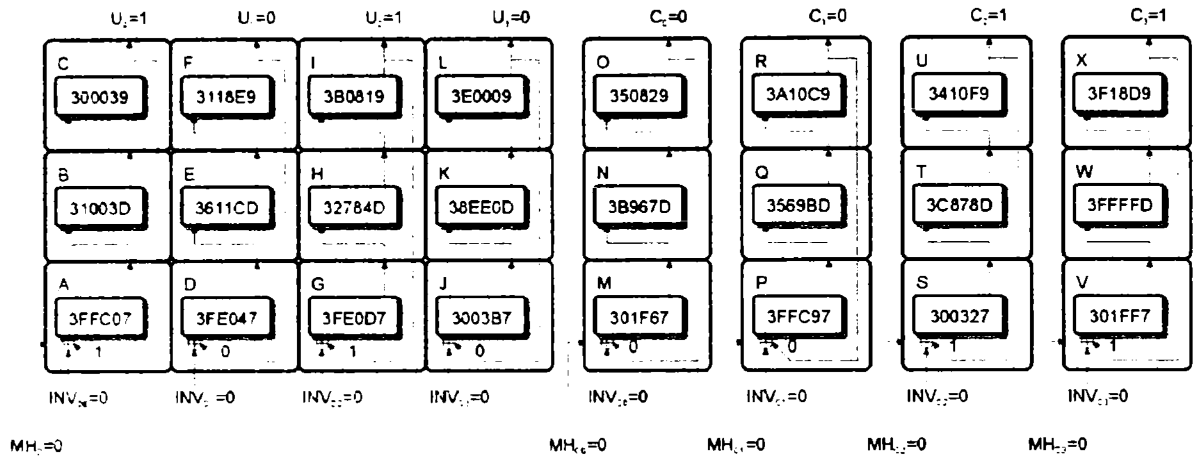


Figure 4-29: Snapshot of memory structures in a fault tolerant macro-molecule. No errors at time $t$.

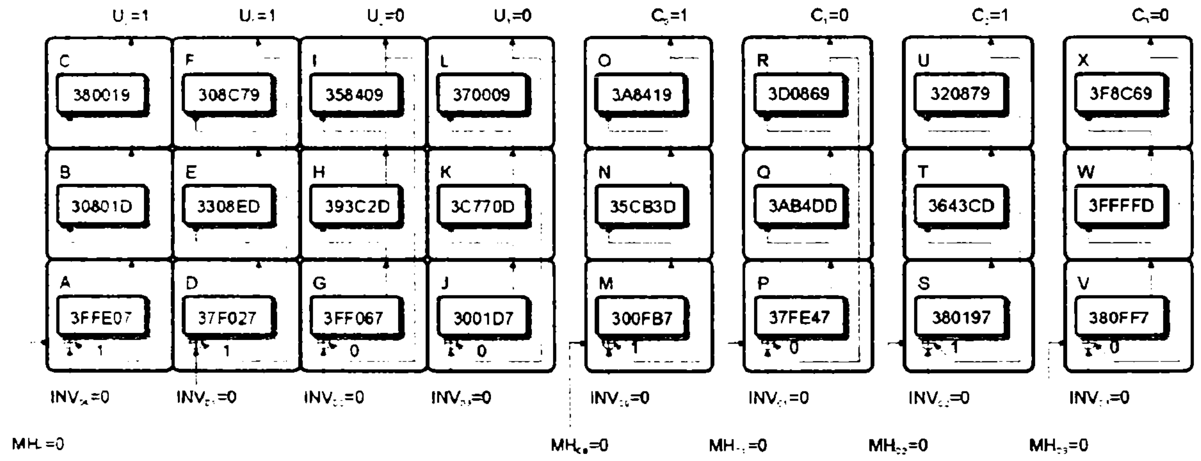


Figure 4-30: Snapshot of a fault tolerant macro-molecule. No errors at time $t$+1.

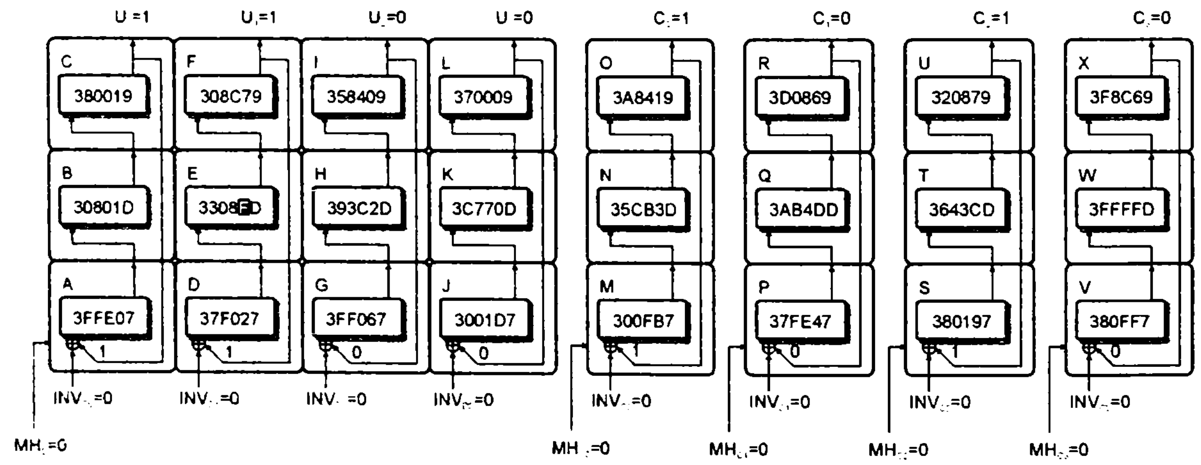


Figure 4-31: Snapshot of a fault tolerant macro-molecule immediately after time $t$+1. A single error has occurred inside molecule $E$.

port; in this case, it corresponds to that of molecule $F$, which will be reached after another 16 clock cycles.

At time $t$+17, the erroneous bit has reached the data output port of molecule $F$, with the memory configuration shown in Figure 4-32. At this moment, the Error Correcting Logic reads its inputs as $U=u_0u_1u_2u_3=1100$ and computes the new check bits $C'=c_0c_1c_2c_3=1010$, which are different than the stored check bits $C=c_0c_1c_2c_3=1101$. This makes up for a non-zero error syndrome $S=s_0s_1s_2s_3=0111$, indicating that a single error occurred (since $DDE=s_0\cdot s_1\cdot s_2+\overline{s_3}=0\cdot1\cdot1+\overline{1}=0$), and the error affected bit $U_1$ (see Equation (67)). Therefore, the Error Correcting Logic will activate signal $INV_{D1}$ (see Figure 4-32), the next clock cycle leading to the successful correction of the error. The situation at time *t*+18 is shown in Figure 4-33.

Let us suppose now that immediately after moment *t*+18 a double error will affect the content of the memory by flipping bits $I_5$ and $U_5$. The memory content at *t*+19 will therefore allow the error to be put into evidence by the Error Correcting Logic and is presented in Figure 4-34. Before the error occurred, the data content at the output ports was $U = u_0u_1u_2u_3 = 0100$, with the corresponding check data being $C = c_0c_1c_2c_3 = 0111$. The double error affects the memory content by transforming both data and check readings as $U^* = u_0^*u_1^*u_2^*u_3^* = 0110$ and $C^* = c_0^*c_1^*c_2^*c_3^* = 0101$. The Error Correcting Logic computes the new check bits, which result in $C' = c_0'c_1'c_2'c_3' = 1000$, which are different than the stored (and erroneous) check bits $C^* = c_0^*c_1^*c_2^*c_3^* = 0101$. This makes up for a non-zero error syndrome $S = s_0s_1s_2s_3 = 1101$ that indicates a double error occurred since $DDE = s_0 \cdot s_1 \cdot s_2 + \overline{s_3} = 1 \cdot 1 \cdot 0 + \overline{0} = 1$. The situation when a double error affects the memory content cannot be recovered and therefore the recovery procedures must be triggered at the higher, cellular, level; this is done by activating the *KILL* signal.

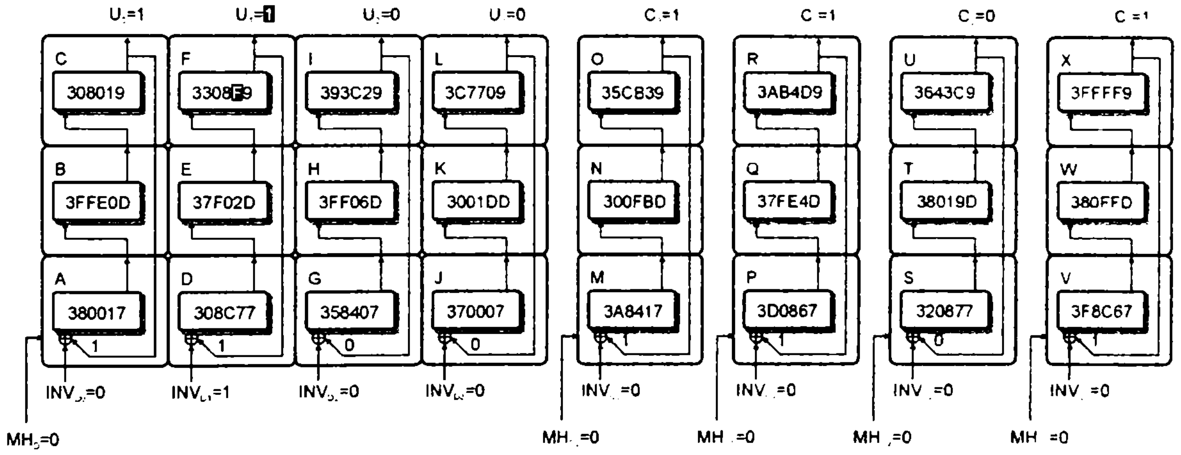


Figure 4-32: Snapshot of a fault tolerant macro-molecule at time *t*+17. The erroneous bit has reached the data output port of molecule F.

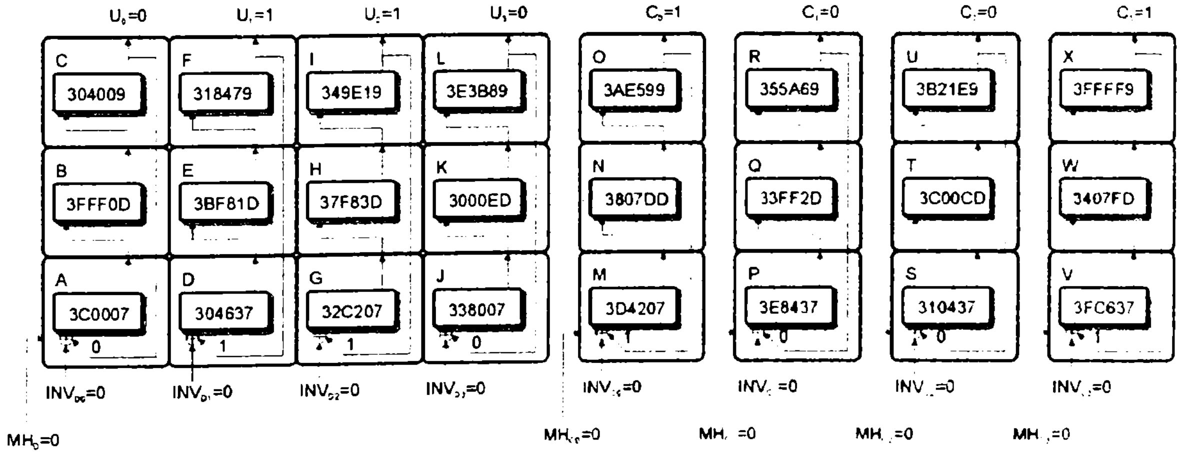


Figure 4-33: Snapshot of a fault tolerant macro-molecule at time *t*+18. The error has now been corrected.

One way of dealing with an unrecoverable situation inside a fault-tolerant macro-molecule would be to have the Error Correcting Logic activate *all MH* signals. This measure would effectively stop the data shifting process inside all of the macro-molecule's components; since its data content has been compromised, there is no reason an external entity (another cell or organism) should continue the execution of the stored genetic program. This strategy allows for a strict containment of an error-affected memory while preserving the logic functionality of the cell. However, there are no benefits from the hierarchical architecture in Embryonics, which is also shared by its self-repairing mechanisms.

As presented in Section 3.7, the *KILL* signal is triggered whenever an unrecoverable situation occurs at molecular level, i.e. there is a fault that cannot be corrected by employing the self-repairing strategies at this level. The *KILL* signal disables the entire cell, therefore triggering the self-repair at the higher level, which is the cellular level. The same strategy can also be used when macro-molecules are used within a cell: when an unrecoverable situation occurs inside a macro-molecule, the Error Correcting Logic will trigger the *KILL* signal, thus forcing the self-repair at the higher, cellular level. Table 4-6 indicates the combination of signals that has to be set by the Error Correcting Logic in order to initiate a *KILL* process. Whenever a southern border memory molecule receives from the Error Correcting Logic the combination of signals meaning that an inversion is required while the memory no longer shifts its data ($MH = 1$ and $INV = 0$), that molecule will activate the *KILL* signal, which will spread and deactivate *all* molecules from within the cell. The situation is presented in Figure 4-35.

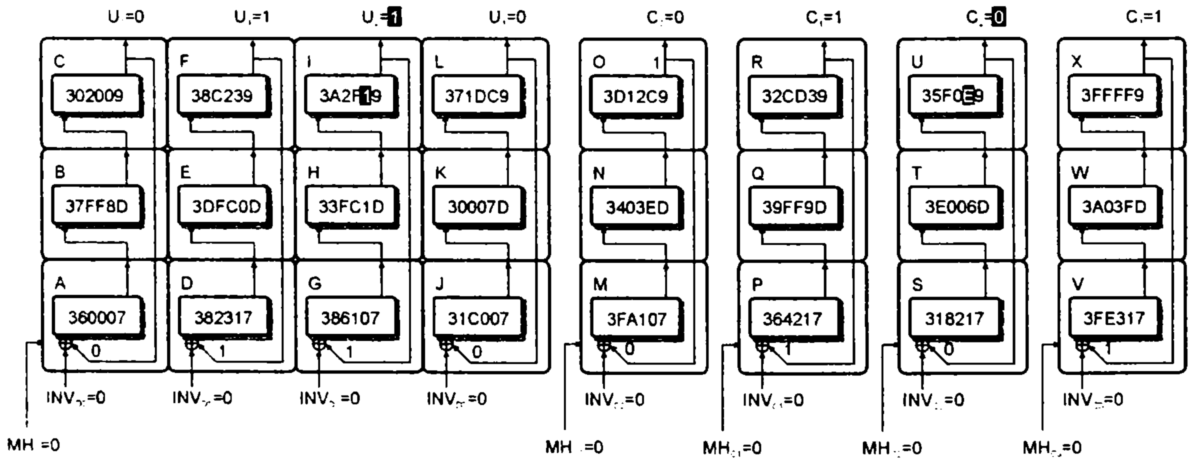


Figure 4-34: Snapshot of a fault tolerant macro-molecule at time $t$+19. There is a double error affecting bits $I_4$ and $U_4$.

## 4.5. Macro-Molecular Accuracy Threshold

As mentioned in Section 4.1.3, techniques for estimating dependability attributes have been already been considered within the field of quantum computing, therefore giving it the potential of being a source of inspiration for the Embryonics project. Before analyzing the particularities of dependability in Embryonics, the accuracy threshold estimation in quantum computing will be discussed.

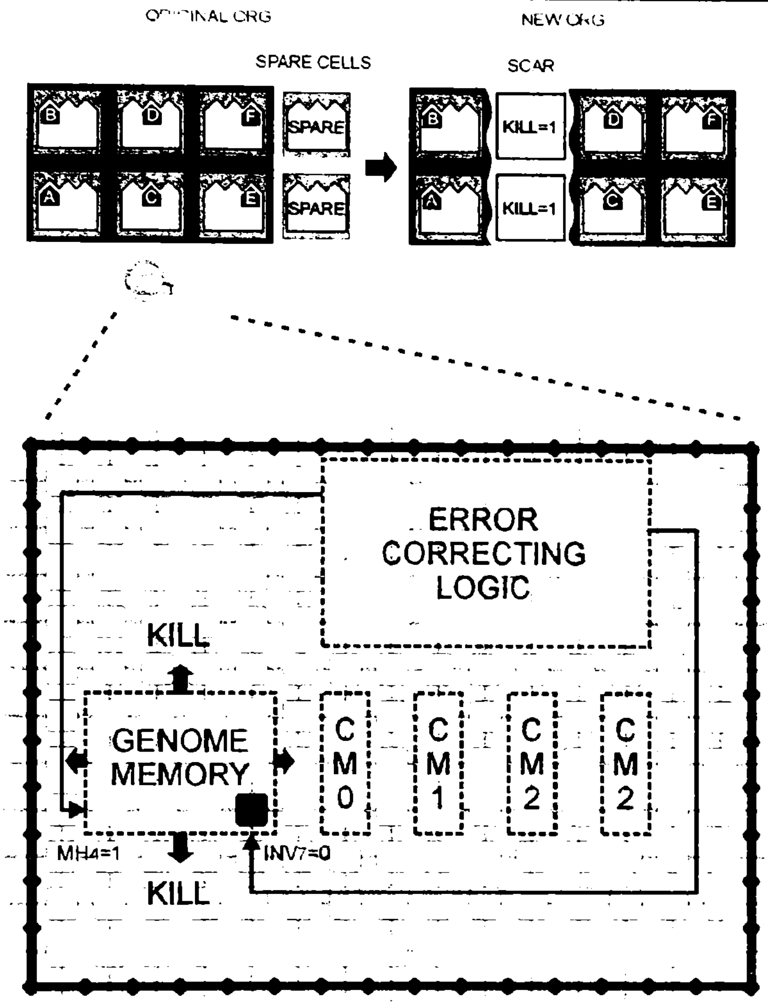


Figure 4-35: KILL at the molecular level, triggered by the Error Correcting Logic.

## 4.5.1 Quantum Dependability

An essential promise of quantum computing is solving in polynomial time problems that otherwise (in classical computing) have only exponential known solutions. Dealing with dependability issues constitutes a priority in quantum computing because of its inner erroneous nature: faults are *native* to the quantum environment. In order to attain reliable quantum computation, one has to deal with errors induced by the constant influence of the external environment upon computational processes. For this purpose, the following assumptions were made: errors appear randomly, are uncorrelated (either in space, or in time), there are no storage errors, and there are no leakage phenomena involved [94].

In order to recover from errors, redundant coding presents a choice of strategies for achieving fault tolerance. However, the recovery process is by itself a computational one, and therefore vulnerable to errors: as information is restored through the use of additional, redundant information, new errors may occur and affect data *during* the very recovery process. In order to ensure a sufficient level of fault tolerance, the following questions have to be raised: what is the accuracy threshold that still warrants valid computation? Or, what is the upper bound of the error frequency that would still allow a successful recovery? These questions were answered in the quantum context [94, 160]; we will however revisit the proposed qualitative assessment since we believe the same reasoning may also be applied to bio-inspired computing systems (Embryonics) and fault-tolerant digital systems in general.

If the redundant coding allows the correction of $t$ errors, then an uncorrectable error occurs if at least $t$+1 errors occur before the recovery process can be finalized. Therefore, if the probability of an error affecting the macro-molecular information is $\xi$, then an uncorrectable error will happen with a probability of the order $\xi^{t+1}$ [94, 160]. Apparently, choosing a reasonably high value for $t$ can make the probability of an unrecoverable error as small as desired; however, the complexity of the code shows a steep rise with the value of $t$, with a polynomial function of the form $t^b$, eventually leading to the situation when correcting the data takes so long that the appearance of an unrecoverable event becomes most likely. Then, the block error probability (BEP) of t+1 errors accumulating in a codeword before the recovery is complete (thus producing an unrecoverable event) will have the form [94]:

$$BEP(t) \sim \left(t^b \xi\right)^{t+1} \tag{75}$$

Minimizing the *BEP* function after parameter $t$ yields:

$$\frac{dBEP(t)}{dt} = 0 \Leftrightarrow BEP'(t) = 0 \tag{76}$$

which results in:

$$\ln t + \frac{1}{b}\ln\xi + 1 + \frac{1}{t} = 0 \Leftrightarrow te^{1/t} = e^{-1}\xi^{-1/b} \tag{77}$$

Solving equation (77) and assuming that $t$ is large [94] gives:

$$t \sim e^{-1}\xi^{-1/b} \tag{78}$$

Substituting this result into equation (75), the minimum block error probability *MBEP* then becomes of the form:

$$MBEP(\xi) \sim \exp\left(-e^{-1}b\xi^{-1/b}\right) \tag{79}$$

The result for $MBEP(\xi)$ is important with respect to estimating the required accuracy for a reliable computation. If we consider $T$ as the time interval without any unrecoverable error occurring, then:

$$T(\xi) \sim MBEP(\xi) \Rightarrow T(\xi) \sim \exp\left(\xi^{-1/b}\right) \tag{80}$$

From this equation, $\xi$ can then be extracted under the form:

$$\xi \sim (\ln T)^{-b} \tag{81}$$

For the situation when no codes are used at all, the accuracy decreases as the computation becomes longer and therefore gives:

$$\xi_{NoCodes} \sim T^{-1} \tag{82}$$

Equation (81) provides a qualitative assessment of the computational accuracy threshold with error protecting codes that is clearly superior to the case when no codes are used at all (equation (82)). Due to the lack of standardization when dependability measures are concerned [3], establishing precise values is difficult. However, this qualitative assessment delivers the necessary criteria for a dependability comparison between two functionally identical systems, before and after applying fault tolerance measures.

### 4.5.2 Quantum-Inspired Dependability in Embryonics

Strategies enumerated for achieving fault tolerance at the macro-molecular level rely on special coding in order to recover from errors affecting stored data. But as user information is restored through the use of additional, redundant information, errors may affect this newly added piece of information during the recovery process, thus raising the following question: what is the accuracy level required for the macro-molecule in order to warrant its correct operation?

If the redundant coding allows the correction of $t$ errors, then an uncorrectable error occurs if at least $t+1$ errors occur before the recovery process is finalized. Therefore, if the probability of an error affecting the macro-molecular information is $\xi$, then an uncorrectable error will happen with a probability of the order $\xi^{t+1}$ [94]. Apparently, choosing a reasonably high value for $t$ can make the probability of an unrecoverable error as small as desired; however, the complexity of the code shows a steep rise with the value of $t$, with a polynomial function of the form $t^b$, eventually leading to the situation when correcting the data takes so long that makes the appearance of an unrecoverable event most likely. Then, the probability of $t+1$ errors accumulating in a codeword before the recovery is complete (thus producing an unrecoverable event) will have the form:

$$\textit{Block Error Probability} = BEP(t) \sim \left(t^b \xi\right)^{t+1} \tag{83}$$

Minimizing the *Block Error Probability* function after parameter $t$ yields $\frac{dBEP(t)}{dt} = 0 \Leftrightarrow BEP'(t) = 0$, which results in:

$$\ln t + \frac{1}{b}\ln\xi + 1 + \frac{1}{t} = 0 \Leftrightarrow te^{1/t} = e^{-1}\xi^{-1/b} \tag{84}$$

Solving this equation [94] and assuming that $t$ is large gives:

$$t \sim e^{-1}\xi^{-1/b} \tag{85}$$

Substituting the result from Equation (75) into Equation (73), the *Minimum Block Error Probability* then becomes of the form:

$$\textit{Minimum Block Error Probability} = MBEP(\xi) \sim \exp\left(-e^{-1}b\xi^{-1/b}\right) \tag{86}$$

The result for $MBEP(\xi)$ is important with respect to estimating the required accuracy for a reliable computation. If we consider $T$ as the time interval without any unrecoverable error occurring, then

$$T(\xi) \sim MBEP(\xi) \Rightarrow T(\xi) \sim \exp\left(\xi^{-1/b}\right) \tag{87}$$

From the last equation $\xi$ can be extracted under the form

$$\xi \sim (\ln T)^{-b} \tag{88}$$

Of course, as long as a macro-molecule is concerned, $T$ represents the time frame required for an error to be corrected, the worst case being a fault occurrence placed furthest from its corresponding data output port; at the same time, $T$ is quite similar to parameters $\Delta T^{*}_{med}$ and $\Delta T_{med}$ introduced in subsection 4.3.6 and whose values can be estimated empirically. Such a situation occurs when the flipped data bit is positioned as the first bit from a bottom row molecule, the shifting path until it may be put into evidence and corrected being of length $F \cdot M$, where $F$ is the storage dimension of the memory molecule and $M$ is the vertical dimension of the macro-molecule (or the number of rows); thus $T = F \cdot M$.

Of course, when no techniques ensuring fault tolerance are implemented, $T$ is proportional with the size of the data and therefore

$$\xi \sim T^{-1},\ \xi \sim \left[FM(N-s)\right]^{-1} \tag{89}$$

As for parameter $b$, it depends on the size of the code as an expression of the gain in complexity with its dimension. In our case the size of the dataword to be protected results as $t = N - s$ bits, where $N$ represents the horizontal dimension of the macro-molecule (or the total number of columns) and s represents the number of spare columns. Therefore, the total size of the codeword, including the redundant bits results (as per Equation (59)) $t + k$, where $k = \left\lceil \log_2(1 + k + N - s) \right\rceil$, with the Hamming matrix being of dimensions $k \times 2^k$. As a result, any fault detection/correction process needs at most a number of computational steps that is given by the dimensions of the Hamming matrix, which is of the order $t \cdot \log_2(t)$. Parameter $b$ can be estimated as the power of $t$ that approximates best the number of necessary detection/correction steps, leading to the following equation:

$$t^b \sim c \cdot t \cdot \log_2(t), \text{ where } c \text{ is a constant.} \tag{90}$$

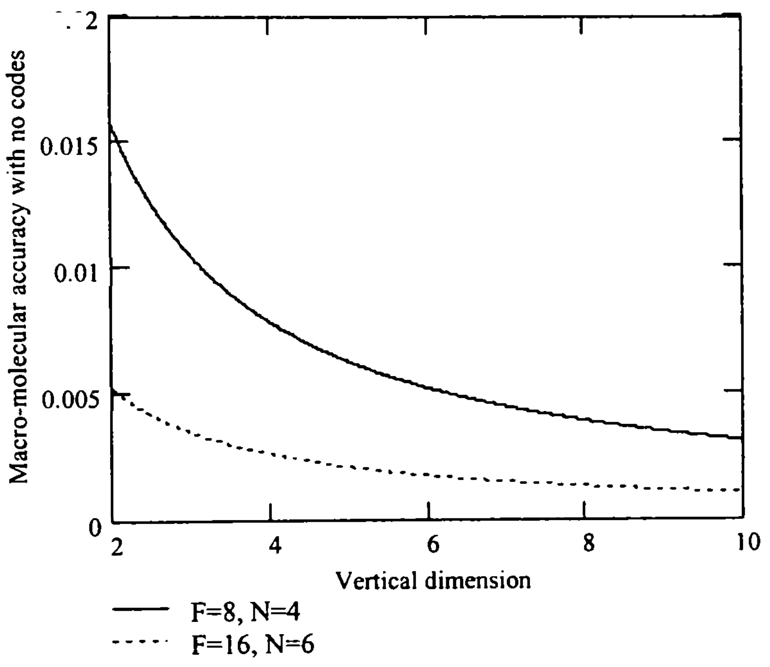
Because there are several algorithms performing the detection/correction process, we will choose the value covering the worst case scenario; following Equation (90) this value results as $b = 2$, the macro-molecular accuracy in case of integrated fault tolerance measures being

$$\xi \sim \left[\ln(FM)\right]^{-2} \tag{91}$$

Parameter $N$ does not appear directly in Equation (91) since its influence is quantified by the gain in the code's complexity defined by parameter $b$ ($N$ signifies the number of data bits that are to be protected, which in turn imposes the number of redundant code bits and the total length of the codeword) The final expressions (90) and (91) show how the macro-molecular accuracy scales for situations with and without error correction techniques. Plots for the accuracy trends are given in Figure 4-36, showing superior scaling as opposed to that when no codes are used at all.

For a macro-molecule with no data error protection mechanisms, the graph from Figure 4-36 (left) shows an accuracy decrease when the overall storage capacity increases. This is consistent with the fact that the probability of an incurring error is directly proportional with the area of the macro-molecule. The situation changes when ECC codes are used. If each row can recover from a single error, then the accuracy dependencies show an increased efficiency with the increase of storage area; however, the graph from Figure 4-36 (right) does not contain parameters involved in an exhaustive manner, which probably leads to the final results for the macro-molecular reliability being less optimistic but, at the same time, superior to the case when no fault tolerant measures are taken into account.

The plots given in Figure 4-36 and the fact that they are in complete agreement with the classical reliability analysis for both situations (without and with ECC codes implemented) demonstrate that the accuracy threshold estimation technique has been successfully taken from quantum computing and imported in bio-inspired computing, namely the Embryonics project.



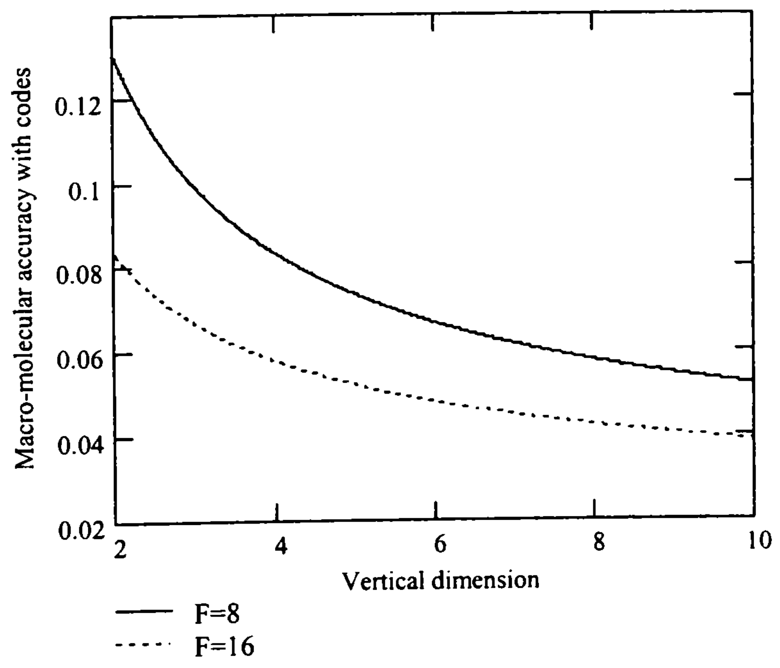


Figure 4-36: Macro-molecular accuracy variation when no codes are used (top), and when codes are in place (bottom).

## 4.6. Fault Tolerance Assessment in Embryonics

The key figure in Embryonics is the hierarchy within the general organization (see Figure 2-2), which in turn enables a hierarchical approach to tolerating faults. There are two different strategies of fault recovery, depending on the corresponding level of organization: at the molecular level, the reconfiguration strategy is based on the individual elimination of faulty units (in this case, molecules), whereas at the superior, cellular level a different strategy is employed, based on the elimination of the entire column containing the faulty unit (in this case, cell). Therefore an Embryonics machine may be regarded as a multi-resolution system, in which faults are tolerated through reconfiguration at two different levels of granularity. While such an approach of hierarchical self-repair certainly offers more in terms of robustness, it also makes estimating its programmable parameters a more difficult task [97].

The main issue here is to establish the requirements in terms of spare resources (the frequency of columns of spare molecules at the molecular level and the frequency of columns of spare cells at the cellular level) so that the overall reliability of a complete Embryonics machine does not drop below an acceptable level $\varepsilon$. Therefore, if the implementation of a complete computing machine in Embryonics may be assimilated to an organism (a regular cell structure), then its reliability function at time $T$ has to verify the following inequality:

$$R_{Org}(t)\big|_T \geq \varepsilon \tag{92}$$

In order to assess the requirements implied by inequality (92), it is necessary to decide on the fault tolerance strategy employed by macro-molecular memory structures; from now on, we will consider the SEC strategy at the macro-molecular level, described in Subsection 4.3.4.1.

### 4.6.1 Reliability at the Molecular Level

When regarded at molecular scale, an entire cell consists of two parts. First, there is the cellular membrane, which is also called space divider (see Chapter 3, Subsection 3.8.1); it has no functional role whatsoever (that is, does not participate actively to any logical machine implementation), its only purpose being that of specifying the borders of a cell. The second, and most important, part of a cell consists of its molecules, their functionality being dictated by the mode they operate in. There are no restrictions over the proportions in which molecules may operate in a certain mode, being possible for a cell to be made either of molecules operating in logic mode only, or molecules operating in any of the memory modes, or any mixture between logic and memory modes.

Therefore, estimating the reliability of a cell is not a trivial task, since it depends on the reliability of its components, which may operate differently. Furthermore, the reliability analysis has to be carried out separately for logic molecules and memory molecules, due to their different strategies in case of incurring faults. On one hand, a faulty logic molecule will be eliminated through reconfiguration, a spare one being activated in order to take its place, whereas a fault detected inside a macro-molecule does not trigger any reconfiguration measures.

Figure 4-37 and Figure 4-38 illustrate the case when a cell suffers a double fault, a single fault affecting one of its logic molecules and another single fault affecting a memory element within a macro-molecule; faults are detected at time t and fault recovery finishes at time $t+\Delta t$ through reconfiguration (for the case of the logic molecule) and data correction (for the case of the memory molecule). The cell consists of a 6x6 array of molecules and contains a 3x3 macro-molecule; there are also two columns of spare molecules (columns 3 and 6). Figure 4-37 shows the molecular configurations, where faulty molecules are considered to be logic molecule (2,1) and memory molecule (4,2).

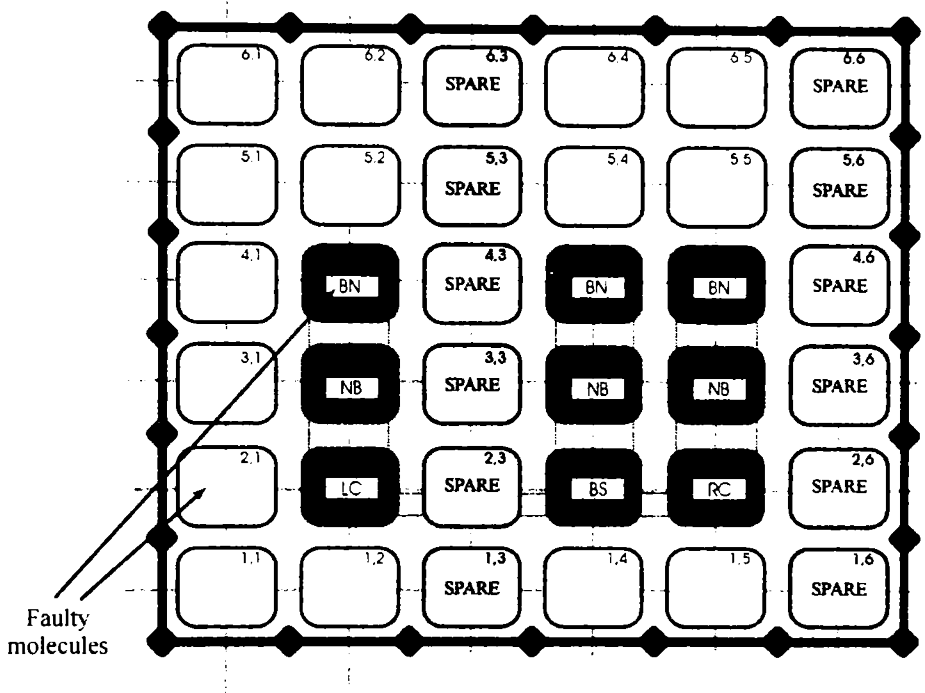


Figure 4-37: A 6x6 cell affected by two faults: one over a logic molecule and another over a memory molecule.

At time *t*, the built-in self-testing mechanism present in each molecule detects a fault affecting molecule (2,1), which operates in logic mode. At the same time, a fault is detected inside the macro-molecule (the mechanism responsible for fault detection and correction was described in SubSection 4.4.2), affecting the data stored at that particular moment by molecule (4,2), which operates in one of the two possible memory sub-modes. The faulty logic molecule will next enter the state "dead", all molecular configurations between itself and the first spare molecule in the row being shifted one position to the right. At the same time, the closest spare molecule in the row will become active in order to be loaded with the configuration data from its left, active neighbor. All communications are being rerouted so as to bypass the faulty molecule, the final situation being shown in Figure 4-38.

The situation inside the macro-molecule is very different. There was a single fault detected inside molecule (4,2) but since the fault is not permanent (all molecules are tested at run-time against permanent faults and, if such is the case, repaired [128]) it is likely that a non-permanent, soft fail has occurred. Such a fault is non-destructive and may be repaired by re-writing the correct value of the flipped data bit. Since there is no reconfiguration involved as recovery measure, the situation of the macro-molecule remains unchanged, unless the reconfiguration triggered by the self-repairing mechanism in the case of logic molecules affects the general layout. The final state of the fully recovered cell is shown in Figure 4-38.

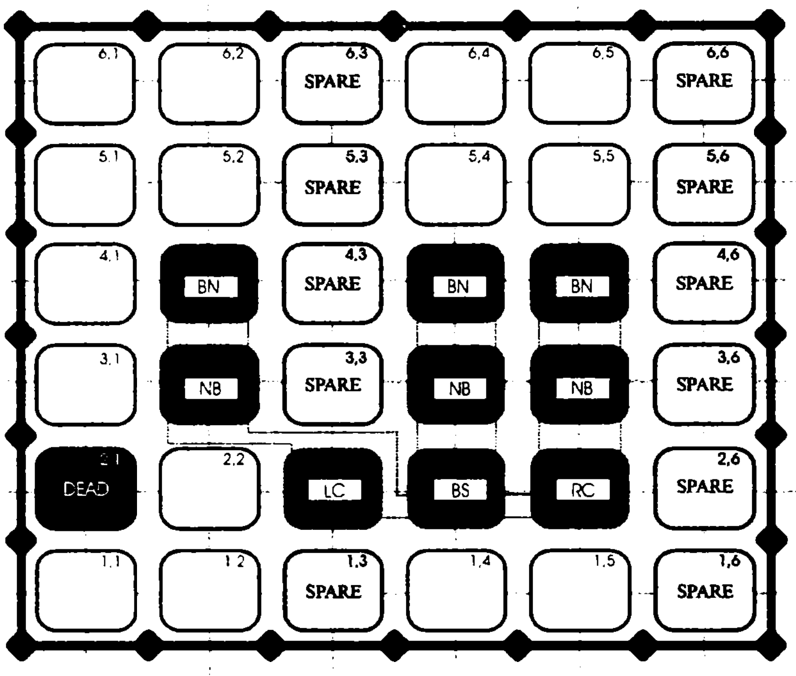


Figure 4-38: A 6x6 cell recovers from a double fault by two faults: one over a logic molecule and another over a memory molecule.

After analyzing the possibilities for a cell to be affected by faults and recover from them, it is now possible to formalize the cellular reliability function by stating its general defining relation:

$$R_{Cell}(t) = Prob\left\{ \textit{ an unrecoverable combination of errors has not yet occurred at time } t \right\}$$

where the parameter to be estimated is a fatal event with two aspects:

i. in the case of memory macro-molecules, a single error can be recovered from, therefore the structure's reliability is the probability that no multiple errors have occurred on the same row at time t;
ii. in the case of logic molecules, no more than S errors may occur in the same row.

#### 4.6.1.1 Reliability of a Macro-Molecule

We will start our analysis by considering a general macro-molecule consisting of a memory array of $M$ lines and $N$ columns (of which $S$ are spares) of molecules, each storing $F$ bits worth of data and with no fault tolerance in place. As argued by subsection 4.3.2, considering that $\lambda$ is the failure rate for a single flip-flop, the reliability of the entire macro-molecule is then given by Equation (28):

$$R_{MMol}(t) = e^{-\lambda FM(N-S)t}$$

On the other hand, adding single fault tolerance capabilities to this macro-molecule leads to the employment of $k$ additional columns or arrays of $M$x1 memory molecules, required by storing redundant data. Parameter $k$ represents the smallest integer that satisfies the following relation:

$$k = \left\lceil \log_2(1+N-S+k) \right\rceil \qquad (93)$$

Then the reliability function for a macro-molecular row can be redefined as follows:

$$R_{Row}(t) = Prob\{\ no\ FF\ fails\ \} + Prob\{\ single\ FF\ fail\ \}$$

$$R_{Row}(t) = e^{-\lambda F(N-S+k)t} + C^{1}_{F(N-S+k)}\left(1-e^{-\lambda t}\right)e^{-\lambda[F(N-S+k)-1]t} \tag{94}$$

which gives the overall reliability function of

$$R_{MMol}(t) = \left[R_{Row}(t)\right]^{M} = \left[e^{-\lambda F(N-S+k)t} + C^{1}_{F(N-S+k)}\left(1-e^{-\lambda t}\right)e^{-\lambda[F(N-S+k)-1]t}\right]^{M} \tag{95}$$

#### 4.6.1.2 Reliability of an Ensemble of Logic Molecules

The reliability analysis of embryonic structures made entirely by logic molecules has been done already [84, 85, 86]. We will, however, reconsider such an analysis as the molecular internal architecture has been changed with the addition of the memory-operating mode (see Chapter 3). Let us consider that the logic molecules make up a rectangular structure of $M^*$ lines and $N^*$ columns, of which $S^*$ are spares. Parameters $M^*$, $N^*$ and $S^*$ are generally different than parameters $M$, $N$ and $S$ considered in subsection 4.5.1.1 since they characterize completely different entities. Furthermore, the failure rate $\lambda$ considered for the elementary memory unit (the flip-flop) may prove to be different than the failure rate $\lambda^*$ used in case of a logic molecule (which typically employs other resources than a memory one), in which situation flip-flops may be used under different operating conditions or may not be used at all.

Such a logic structure was analyzed as being based on the $k$-out-of-$m$ reliability model, that is, the proper function of the system as a whole is ensured as long as at least k units out of a total of m are operating normally [84]. In our case, considering that any detected fault inside a molecule triggers a reconfiguration strategy that leads to the "death" of the respective molecule, this means that no more than $S^*$ errors (or faulty molecules) can be tolerated in a single row. Therefore the reliability of a single row becomes:

$$R_{Row}(t) = \sum_{i=N^*-S^*}^{N^*} C^{i}_{N^*} . e^{-i\lambda^* t}\left(1-e^{-\lambda^* t}\right)^{N^*-i} \tag{96}$$

Because the logic ensemble is built of $M^*$ rows, its overall reliability can now be estimated as:

$$R_{LogicEnsemble}(t) = \left[R_{Row}(t)\right]^{M^*} = \left[\sum_{i=N^*-S^*}^{N^*} C^{i}_{N^*} . e^{-i\lambda^* t}\left(1-e^{-\lambda^* t}\right)^{N^*-i}\right]^{M^*} \tag{97}$$

### 4.6.2 Reliability at the Cellular Level

Any cell within the Embryonics project is made of molecules operating either in logic mode or in any of the memory modes. A full reliability analysis at the cellular level requires estimating the individual reliabilities of the two component structures, macro-molecules and logic ensemble, which are given by Equations (95) and (97), respectively. All component structures are required to perform properly in order to ensure the normal operations of the cell; therefore the cell can be considered as a series system in which each subsystem (be it macro-molecule or logic ensemble) has to function if the system as a whole is to function [44]. Therefore the cellular reliability function may be derived as the product of the reliability functions of its component subsystems as follows:

$$R_{Cell}(t) = R_{LogicEnsemble}(t)\prod_{i=1}^{n}\left(R_{MMol}\right)_i(t) \tag{98}$$

where $n$ is the number of macro-molecules present in the cell. Assembling the terms from Equation (98) gives the final result for the cellular reliability function:

$$R_{Cell}(t) = \left[\sum_{i=N^*-S^*}^{N^*} C_{N^*}^{i} . e^{-i\lambda^* t}\left(1-e^{-\lambda^* t}\right)^{N^*-i}\right]^{M^*} \prod_{i=1}^{n}\left[e^{-\lambda F(N-S+k)t} + C_{F(N-S+k)}^{1}\left(1-e^{-\lambda t}\right)e^{-\lambda[F(N-S+k)-1]t}\right]^{M}\Big|_i \tag{99}$$

### 4.6.3 Reliability at the Organismic Level

Let us consider an array of cells that make up an organism, with dimensions of $M_c$ lines and $N_c$ columns, including Sc columns of spare cells. Such an organism plays in Embryonics the role of a general computing system, its various functions being performed by the internal components, which are the cells. The central problem that initiated the previous reliability analyses was to provide some sort of formalism in order to assess the requirements that ensure a certain reliability level for the computing system, or in our case, the organism. For this purpose it is necessary to investigate first the situations that lead to a cell becoming faulty and the reconfiguration strategy at this level.

It is worth reconsidering the aspects that might lead to the "death" of an entire cell. As discussed in subsection 3.7.1, when there are more faults affecting the logic molecules of a cell than available spares in a single row, then a special signal named *KILL* becomes active, which spreads across the cell, effectively disabling all the molecules. The same situation occurs when a macro-molecule is affected by a multiple error in a single row [98]; if such is the case, the Error Correcting Logic activates the *KILL* signal, thus killing the entire cell.
At the cellular level, the catastrophic event of more faults affecting a cell than available possibilities of repairing is perceived as a cell becoming faulty, situation that activates a reconfiguration process, which will eliminate the entire column of cells (including the faulty cell). Ongoing cellular processes from the marked column will be taken over by a spare column by shifting them to the right. In order to illustrate the reconfiguration process, Figure 4-39 presents a cellular structure that will be affected by faults at this (cellular) level, the cells being affected being (2,3) and (4,7). Since any fault detected at the cellular level triggers a column-elimination strategy, Figure 4-40 shows the organism's layout after the reconfiguration. Because cell (2,3) was faulty, this means the entire 3rd column will be disabled, its role being transferred to the closest spare column to the right, which will become active (4th column in Figure 4-40). A similar situation occurs with faulty cell (4,7), the 7th column being killed; the closest spare column to the right will become active and functional processes will be transferred from column 8 to 9 and from column 7 to 8.

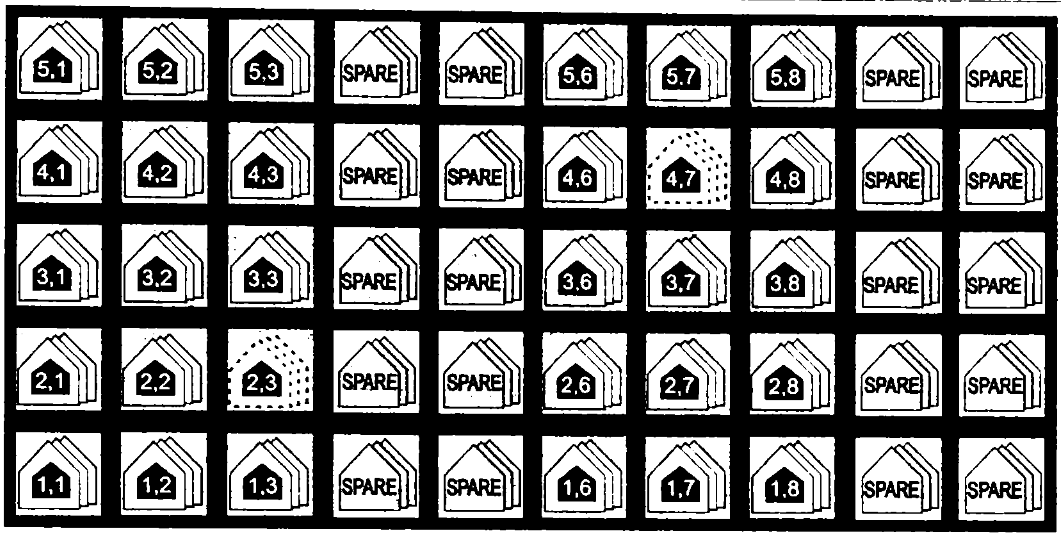

5,1 5,2 5,3 SPARE SPARE 5,6 5,7 5,8 SPARE SPARE
4,1 4,2 4,3 SPARE SPARE 4,6 4,7 4,8 SPARE SPARE
3,1 3,2 3,3 SPARE SPARE 3,6 3,7 3,8 SPARE SPARE
2,1 2,2 2,3 SPARE SPARE 2,6 2,7 2,8 SPARE SPARE
1,1 1,2 1,3 SPARE SPARE 1,6 1,7 1,8 SPARE SPARE

Figure 4-39: A complete organism may also be affected by faults; cells (2,3) and (4,7) will soon die, forcing a reconfiguration at the cellular level.

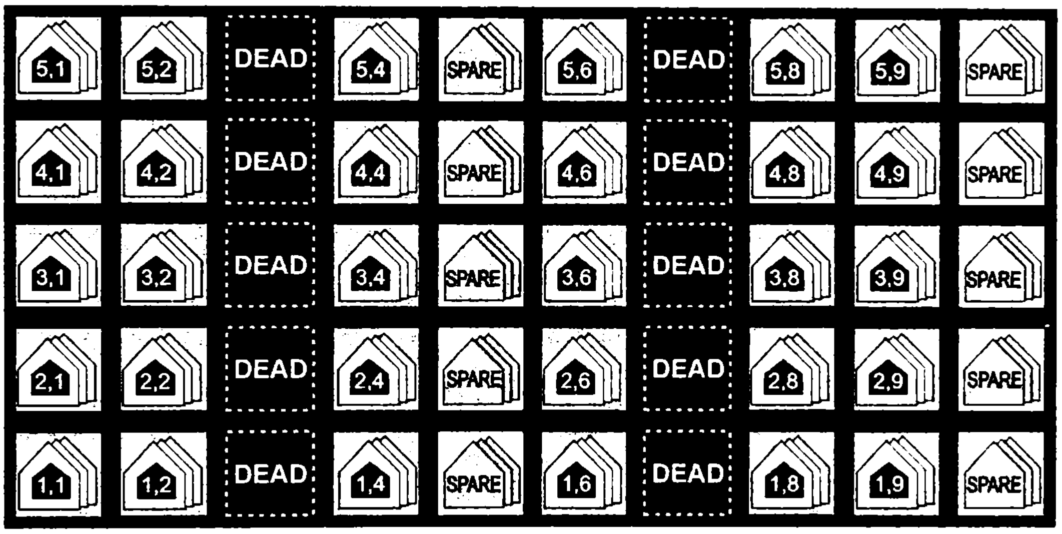

5,1 5,2 DEAD 5,4 SPARE 5,6 DEAD 5,8 5,9 SPARE
4,1 4,2 DEAD 4,4 SPARE 4,6 DEAD 4,8 4,9 SPARE
3,1 3,2 DEAD 3,4 SPARE 3,6 DEAD 3,8 3,9 SPARE
2,1 2,2 DEAD 2,4 SPARE 2,6 DEAD 2,8 2,9 SPARE
1,1 1,2 DEAD 1,4 SPARE 1,6 DEAD 1,8 1,9 SPARE

Figure 4-40: The organism remains functional after the reconfiguration; the functions of faulty cells (2,3) and (4,7) were taken over by (2,4) and (4,8), respectively.

The above considerations justify a reliability analysis of an organism as being also based on the $k$-out-of-$m$ model, where the successful operation of the organism is ensured by the proper function of at least $k$ columns out of a total of $m$. Let us consider the organism's dimensions as being of $M_c$ lines and $N_c$ columns, including $S_c$ spares. Therefore, the reliability of the organism is given by the fact that, at any moment, at least $N_c - S_c$ columns are operational:

$$R_{Org}(t) = \sum_{i=N_c-S_c}^{N_c} C_{N_c}^{i} R_{Column}^{i} \left(1 - R_{Column}\right)^{N_c - i} \qquad (100)$$

Since a column is fully operational if all $M_c$ component cells are functional, the reliability function for a column results as:

$$R_{Column}(t) = R_{Cell}^{M_c}(t) \qquad (101)$$

Therefore, the final expression for the reliability function of an organism (or of any cellular structure) is given by:

$$R_{Org}(t) = \sum_{i=N_c-S_c}^{N_c} C_{N_c}^{i} R_{Cell}^{iM_c} \left(1 - R_{Cell}^{M_c}\right)^{N_c - i} \tag{102}$$

## 4.7. Bridging Quantum and Bio-Inspired Computing

At a first glance, the boundaries between bio-inspired computing and quantum computing seem to discourage unveiling any common ground between the two fields. Though technology may be essentially different [82], they both share the same error model and employ techniques for achieving fault tolerance from classic computing [95, 96]. Moreover, the accuracy threshold $\xi$ in the quantum computing context and the failure rate $\lambda$ in the bio-inspired computing context are not dissimilar: while $\lambda$ gives the error probability, $\xi$ gives the upper bound for the error probability so as the computation still remains valid. Therefore, we have:

$$\max(\lambda) \sim \xi \tag{103}$$

As long as the error rate $\lambda$ is below the accuracy threshold, valid computations can be recovered from the damaging effects of occurring errors. However, these estimations only cover the time frame between an error occurrence and the end of the recovery process, that is, the period between data damage and data restoration. While a reasonable accuracy can be obtained by using error-correcting codes, the occurrence of errors becomes more likely as the length of the computation increases [94]. Since machines based on the Embryonics platform are intended to operate over long periods of time (therefore involving long computations), this primarily affects the memory structures in Embryonics, since its logic structures already have protective measures implemented [67]; therefore measures for extending the valid computation length have to be taken, and, once again, quantum computing offers a source of inspiration.

### 4.7.1 From Multiple-Level Self-Repairing to Multiple-Level Coding

The computation length limit in fault tolerant quantum computing [94, 96, 97] can be overcome by employing concatenated codes; when viewed at a higher resolution, each quantum bit is encoded by *a block* of quantum bits. Such a hierarchical encoding appears to be particularly well suited for the Embryonics project since its architecture offers an intrinsic hierarchy, one level (molecular) corresponding to a higher resolution view of the next superior level (cellular). With information being encoded at each level, Embryonics seems natively endowed for implementing concatenated codes, the principles being presented in Figure 4-41; a first idea of information coding in Embryonics for error detection purposes was presented in [101].

Instead of storing binary words worth of data, fault-tolerant macro-molecules can store binary words that would in turn assemble to provide data for the next hierarchical level as an encoded binary digit [97]. At the cellular level, genetic information may also be protected using similar Hamming codes as implemented at the molecular level. If such is the case, and we accept the error rate at the macro-molecular level as being $\varepsilon$, then an unrecoverable error will occur with a probability of $\varepsilon^2$. A concatenated code [94] in which each bit at the cellular level is encoded by 7 bits at the molecular level stored by fault-tolerant macro-molecules will give the probability of an unrecoverable error as $\varepsilon^{2^2} = \varepsilon^4$ (assuming errors are of stochastic nature and uncorrelated). This is where error coding and concatenation can work together against error influences: while error coding lowers the probability of an unrecoverable error, concatenation brings the possibility of making it arbitrarily small by adding sufficient levels of concatenation.

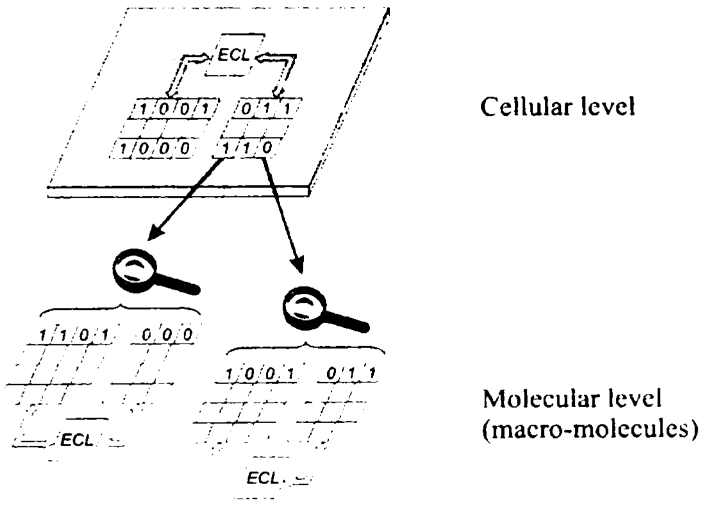


Figure 4-41: Two-level concatenated coding in Embryonics [97].

In Figure 4-41 the following scenario is being considered: at the molecular level, genetic information is divided and stored by fault-tolerant macro-molecules using a (7,3) single error correcting Hamming code [46]. Essentially, 4 bits worth of genetic data (stored by the GENOME MEMORY in Figure 4-42) are encoded into a 7-bit codeword, which makes up the elementary piece of information at this level. The redundant check bits, stored by the CONTROL MEMORY ($CM_{0\text{-}2}$ in Figure 4-42) are derived from Equation (70) [104].

At the cellular level, each 7-bit code word from the molecular level make up for a single higher-order bit of actual data, which will be called Bit from this moment; its value can be derived, for instance, as the parity value from Equation (104):

$$U^i = u_0^i \oplus u_1^i \oplus u_2^i \oplus u_3^i \oplus c_0^i \oplus c_1^i \oplus c_2^i, \text{ for } i = 0 \div 3 \qquad (104)$$

The same single error correcting, Hamming coding, from the molecular level (see Equation 22) can now be applied to the 4 Bits $U^{0:3}$ in order to generate the redundant check Bits $C^{0:2}$:

$$\begin{cases} C^0 = U^0 \oplus U^2 \oplus U^3 \\ C^1 = U^0 \oplus U^1 \oplus U^2 \\ C^2 = U^1 \oplus U^2 \oplus U^3 \end{cases} \qquad (105)$$

At this point, a structure that encodes genetic information in a hierarchical manner by using concatenated codes has been established [96]. At the molecular level, the basic units (the memory molecules) are assembled to build a fault tolerant macro-molecule (FTMM), which is shown in Figure 4-43 [98]. The

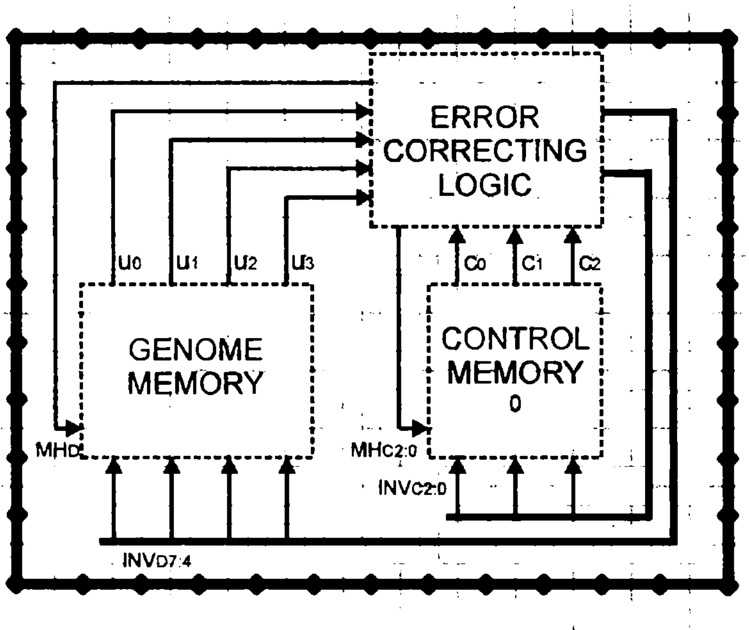


Figure 4-42: Block schematic of a complete cell with a fault tolerant SEC memory structure.

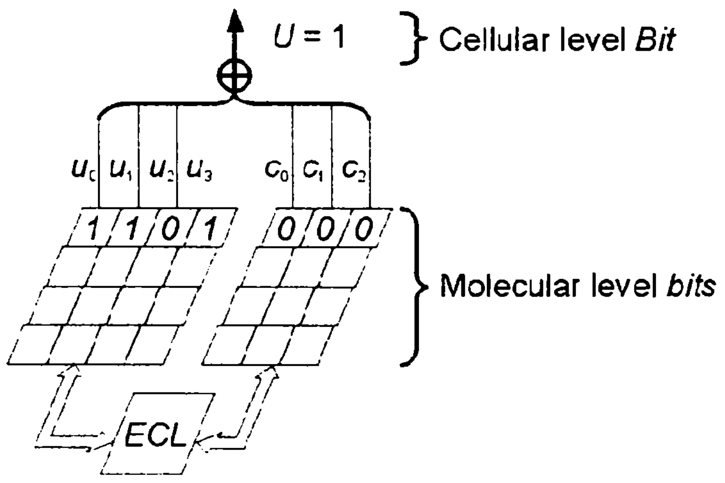


Figure 4-43: The first level in concatenated coding is the FTMM (Fault-Tolerant Macro-Molecule) [96].

*FTMM* computes the value of the corresponding Bit by implementing Equation (105), while the *ECL* keeps the code word accurate by implementing Equation (104).

At the cellular level a similar structure is assembled (see Figure 4-44), with the basic units being the *FTMM*s. Each *FTMM* computes a Bit, with 7 such Bits making up a (7,3) Hamming code. The correction mechanism at the cellular level is identical to that present at the molecular level [98]: whenever a single error affects a 7-Bit word, the error is located and the corresponding value inverted.

The check Bits provide vital information for recovering a code word from a single error at the cellular level. Their value is computed directly from the Bits that carry genetic information ($U^{0\text{-}3}$) and do not come from actual data from the molecular level. Therefore, there seems to be no real need for further encoding the check Bits. However, if the advantages of concatenated codes are to be preserved, the check Bits also require coding; if this process implies the derivation of a new value (the code) from several values known in advance (source data), in this case a reverse process is required: the value of the encoded data is known in advance at the cellular level (that is, the value of the check Bit) and the values from the molecular level (source data) need to be computed.

As it is implemented, the code word resulting from Equation (105) is also able to recover from an error affecting a single Bit. An unrecoverable situation occurs when a double error affects a code word at the cellular level. However, this can only happen if two sub-blocks fail simultaneously, which, in turn, means that each of the two (7,3) Hamming code words from the molecular level have to experience a double error. Because each Bit is encoded as suggested by Equation (105) (shown in Figure 4-43), such a concatenated code offers superior protection. Considering Equation (94) and substituting with 7 the length of a code word implemented by a macro-molecule with single fault-tolerance, its reliability becomes:

$$R_{bit\_word}(t) = e^{-7\lambda t} + 7\left(1 - e^{-\lambda t}\right)e^{-6\lambda t} \qquad (106)$$

Because the length of the code word and the fault-tolerance are similar at the cellular level, the reliability of the code word at this level is:

$$R_{Bit\_word}(t) = R^{7}_{bit\_word}(t) + 7\left[1 - R^{7}_{bit\_word}(t)\right]R^{6}_{bit\_word}(t) \qquad (107)$$

A direct comparison between Equations (106) and (107), which define the reliability function for the basic information unit at each level, confirms the superior protection offered by a second level of concatenated coding.

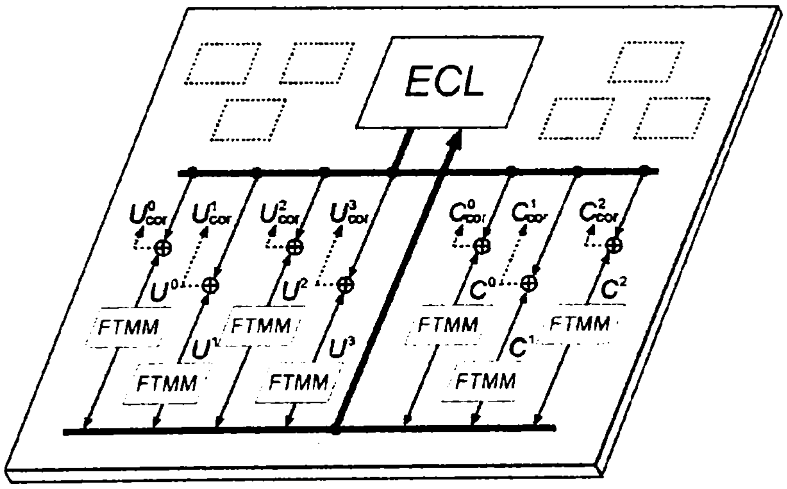


Figure 4-44: The second level in concatenated coding is made-up by Hamming-coded Bits.

## 4.7.2 Conclusions

Attaining superior dependability in environments inducing frequent faults constitutes a common problem in both bio-inspired and quantum computing. We have shown that the accuracy threshold estimation (inspired from quantum computing) can be linked to the reliability analysis of bio-inspired computing, both techniques producing similar qualitative results. Because applications targeted by both quantum computing [145, 146, 147] and bio-inspired computing (Embryonics included) [95, 96, 97, 98] share the same high dependability requirements, the accuracy threshold estimation is relevant for both fields. Therefore, concatenated coding also represents a possible solution for Embryonics. Its hierarchical architecture is structurally similar to that of concatenated coding, thus facilitating its implementation.

# CHAPTER 5

# CONCLUSIONS

In this chapter we will try to consider and argue upon the extent the architecture we have presented fulfills the initial goals (Section 5.1), outlined in Chapter 1, and specify the original contributions added on top of the state-of-the-art. We will attempt to give a two-fold perspective over the present of the Embryonics memory structures: one from an engineering standpoint (SubSection 5.2.1) and one that, while certainly being less technical, reaches the land of philosophy (SubSection 5.2.2). We will conclude by presenting possible future developments for Embryonics, as we see them.

## 5.1 Analysis of the Results

The main goal of this thesis was to integrate the operative genome concept (see SubSection 2.2.3.2) within the hardware implementation of the current Embryonics state-of-the-art. This meant that a new memory design had to be elaborated so that it would fit at the end into an already robust and powerful architecture.

In the first place, a memory structure had to be created without severely affecting the hardware overhead. Therefore we introduced the second operating mode (with two operating sub-modes) for each of MuxTree molecules, thus making better use of a resource (the configuration register *CREG*) that appeared to be suitable for this purpose. The memory had to exhibit certain less-than-usual features:

a. preserve bio-inspiration in its design and functionality;
b. at each clock cycle, it had to deliver the next data, thus rendering any addressing mechanism futile;
c. it had to integrate the existing self-repairing mechanisms at the molecular level: on one hand, the memory shouldn't present any negative impact onto the repairing process of a faulty logic molecule, and, on the other hand, it should also provide its own self-repairing capabilities;
d. it had to integrate the hierarchical self-repairing mechanism;
e. it had to integrate the self-replicating mechanism.

### a. Bio-Inspired Design and Functionality

Since the whole Embryonics project was conceived so as to follow the path of bio-inspiration as closely as possible, the memory also had to be designed in its spirit. The biological DNA is a strand of chemically-bonded molecules: it consists of molecules that store complex genetic information, but it is not a cell by itself. Therefore we called our memory structures macro-molecules, because they rely on molecules to store genetic information but do not constitute cells by themselves.

A particular challenging task was to settle for the most appropriate architecture that was to provide flexibility of use, while making the best out of the available resources. These were both demands and constraints, since the final MuxTree design was physically implemented and tested in a hardware prototype. Therefore we chose to implement the memory unit by employing the most suitable molecular resource: the configuration register (*CREG*). In order to assemble useful data words, several macro-molecules have to be used synchronously. However, if different memory timings are required, there is a mechanism (which we called *HOLD*) that inhibits information shifting in a particular macro-molecule for as long as desired. Functionality levels at the molecular level are kept to a maximum: if routing resources are needed along with data storage, the macro-molecule should be made of memory molecules operating in the short memory mode (SubSection 3.5.2.2); if, on the other hand, storage data space needs to be maximized, then the macro-molecule should be made of memory molecules operating in the long memory mode (SubSection 3.5.2.3).

**b. Memory Addressing**

Being the carrier of genetic information, which is executed continuously and sequentially in living creatures, we also designed our macro-molecule to provide data in a similar manner: it shifts its data indefinitely at each clock cycle.

Such an approach prevents the implementation of jump-type instructions. However, this engineering limitation can be, at least partially, compensated by using the *HOLD* feature (SubSection 3.6) to de-synchronize macro-molecules from each other, therefore simulating this kind of instructions (for instance, if a jump instruction could be seen as transfering the control to a procedure residing in a different macro-molecule, then this macro-molecule will only begin to shift its data at that particular moment).

**c. Self-Repair at the Molecular Level**

As far as self-repair at the molecular level is concerned, the design of the new macro-molecule had to deal with two aspects:

- integration of the existing self-repairing mechanisms, protecting the molecules operating in logic mode;
- implementation of a self-repairing mechanism specific to memory macro-molecules in order to provide protection for the stored genetic information.

The self-repairing mechanism for logic molecules relies on spare resources that are activated in case of faulty molecule detection in order to allow the functionality transfer. The particular features of this process inspired the mechanism of setting up a macro-molecule: the memory molecules are chained together in a similar way the initial configuration enters the electronic organism at set-up time. Repairing a logic molecule therefore results as transparent for any macro-molecule, thus ensuring a perfect integration with the existing self-repairing mechanisms for molecules operating in logic mode (SubSection 3.7).

Unfortunately, protecting *functionality* is essentially different than protecting *information*, therefore imposing a new self-repair mechanism be implemented in order to protect genetic data stored by macro-molecules. Such a mechanism had to operate transparently and in parallel with the existing self-repair mechanisms for molecules operating in logic mode. If functionality can be protected by employing redundant resources and majority voting, information recovery requires redundant information. For this purpose, we designed a fault-tolerant macro-molecule, implementing Hamming-type codes (SubSection 4.4).

**d. Hierarchical Self-Repair**

Living beings exhibit hierarchical strategies of fault-tolerance: at the molecular level (DNA is considered to be highly redundant), at the cellular level (faulty cells die and are replaced by newly grown ones), and even at higher levels (there are redundant organs, brain hemispheres are known to be able to transfer some functionalities in case of damage).

Embryonics implements a two-level self-repairing strategy; however, with the introduction of macro-molecular structures this strategy became only partially efficient, not being suited for memory data protection (SubSection 4.1.4). In order to provide similar self-repairing capabilities as in logic mode, fault-tolerance has been added to macro-molecular structures by implementing Hamming codes. A considerable research effort was spent for investigating the causes that lead to the appearance of soft errors and for providing a formal model of their impact over the reliability characteristics. The analysis over the frequency of error types (Section 4.3) suggested as most suited codes those capable of single error correction. Memory data protection was achieved at molecular level: design principles were discussed (SubSections 4.4.3 and 4.4.1) together with an architectural example of a fault-tolerant macro-molecules implementing single error correction and double error detection (SubSection 4.4.5).

The hierarchical self-repair required additional measures in case of failure of self-repair at the molecular level. Such measures were taken in order to ensure that self-repair at the cellular level is triggered whenever there is a shortage of spare molecules left for repair of logic, by employing the *KILL* signal. The same signal is also set by the *Error Correcting Logic* (*ECL*) block whenever more than one error is detected in a macro-molecular data word. As it was previously implemented [128] the self-repairing at the cellular level also had to be modified in order to provide memory data protection. Our solution (SubSection 4.7) proposes multiple-level information coding under the form of concatenated coding.

Assessing the efficiency of the fault recovery processes has also constituted a challenging task, unifying particular aspects from bio-inspired and quantum computing. A thorough reliability analysis was provided, together with a methodology of further increasing the robustness of computational processes in a MuxTree machine.

**e. Self-Replication**

Self-replication in Embryonics was entirely unaffected by the introduction of macro-molecules. They operate completely transparently with respect to self-replication and therefore their integration in the Embryonics original design is complete.

## 5.2 Original Contributions

This thesis represents an individual contribution to the effort carried by the Embryonics team, resulting from a rather unique privilege of working closely with the initiators of the project and assisting to the birth of a new kind of artificial, bio-inspired life form, called MuxTree. As such, describing a research effort integrated within a larger project encounters difficulties when it comes to pointing out the original contributions of the author.

Chapter 2 was intended to provide essential background to the state-of-the-art concerning the Embryonics project. As the inception of the project predates my arrival in the *Logic Systems Laboratory* (*LSL*) at the *Swiss Federal Institute of Technology* at Lausanne (*EPFL*), I cannot claim authorship for any of the concepts presented in this chapter. My contribution was therefore centered on providing an investigative top view over the project's state-of-the-art particular features in order to assess directions where the Embryonics architecture could further benefit from bio-inspiration.

Chapter 3 contains research results that are entirely original: the design of a bio-inspired memory architecture for Embryonics. When I first approached this task, the only memory resource of a molecule was part of the *Functional Unit* (*FU*), under the form of a flip-flop. Despite implementing the triple modular redundancy technique in order to achieve fault-tolerance, this resource alone was considered as insufficient for storing large pieces of genetic programs. I therefore had to look for alternative ways of providing data storage while keeping in mind the constraints represented by their successful integration within the Embryonics' architecture. The features that had to be exhibited by the new memory were concluded through the collective effort of a research team, which I was member of. The design and implementation of macro-molecules, together with the second operating mode (the memory mode) constitute the original results of my research done at the *LSL*. Implementing the hardware design of the new memory onto actual pieces of hardware constitutes another original result. This achievement followed what was already a tradition at the LSL: validating a hardware design not just by simulation, but also through hands-on experiments carried over a physical implementation.

Chapter 4 contains results of the research carried out at the *"Politehnica" University of Timisoara* (*UPT*), Romania and assembles two main parts. Since biological processes are essentially different than those taking place in digital devices, it appeared that building a fault tolerant bio-inspired memory has to also find justifications that go above bio-inspiration; therefore, the first part of this chapter (Sections 4.1 and 4.2) justifies the need for building fault-tolerant memories from an engineering point of view: soft fails are a constant menace for digital machines that have to exhibit extreme dependability levels, which also constitute the target of the Embryonics project. Although this part is not original, it provides solid arguments for the second part of the chapter. The second part of the chapter contains original results toward achieving a dependable macro-molecule. It begins with a formal model describing the normal operation of a macro-molecule but also allowing for error injection. A reliability analysis is provided here in order to investigate different strategies for tolerating faults. Section 4.4 is dedicated to designing a single-fault-tolerant memory structure and to settling all issues that arose with respect to integrating the new architecture within the Embryonics platform. The design issues for implementing single error-correcting without or with double error detecting Hamming codes, the provided example, and the integration of macro-molecular self-repair into the self-repairing hierarchy present in Embryonics, are all original contributions. An original contribution is finding common ground between two emerging fields in modern computing, the bio-inspired and the quantum computing, by proving that techniques already established in one field need not necessarily remain confined within that respective field but bring benefit to other fields also. The introduction of the accuracy threshold estimation technique to Embryonics, together with a complete reliability analysis and further increasing data integrity at higher levels through concatenated coding are entirely original contributions.

Beyond the realms of bio-inspired computing, the research involved over this thesis also point to an original contribution that I believe to be also valuable. Though it is almost non-technical, it argues upon similarities between Embryonics and biology that go beyond bio-inspiration [99] and therefore we felt it deserved a dedicated section, following next.

## 5.3 Electronic Stem Cells

**Quote:** "What makes stem cells special is that they're immortal, and they can become anything they want to be." – Dr. James Thomson, University of Wisconsin.

The incredibly huge number of some 60 trillion $\left(60\times10^{12}\right)$ cells make up a human being, with as many as 10 billion $\left(10^{10}\right)$ cells with 100 trillion $\left(10^{14}\right)$ interconnections concentrated in each of our brains (Mange &Tomassini 1998). Yet this entire structure emerges from a single cell, the zygote, giving birth to a completely functional organism that will, together with environmental influences, continue to develop and enhance its features throughout its entire life. However, there are some key questions that arise, driving biologists and not only [99]:

- how can a single cell divide for such a large number of times even us humans find difficult to imagine?
- what mechanisms direct the division process so perfectly that when it ends the result is a healthy organism?

**Stem Cells.** A special type of cell appears to answer some of the previous questions, a cell that can give birth to other identical cells, and all being able to become specialized cells themselves, such as muscular or nerve cells. After years of hard work, scientists succeeded in growing and replicating these mother cells. Called "stem cells", these basic units ultimately mature and differentiate to become the building material of all types of body tissue [99].

At its very beginning, each organism originates from a single cell, the zygote. Its development process takes place through successive cellular divisions and differentiations, proof for the existence of a very special cell, capable of differentiating into any kind of needed, specialized cell [137]. Christened *stem cell*, its unique transforming capacity could provide a solution for an organism's survival to extreme stress, such as the loss of a vital organ, giving it a huge potential in biology and medicine [74, 167, 170]. The zygote has the potential to form an entire organism as it has the remarkable feature of being totipotent, meaning that its potential is total. Immediately following the fertilization, it starts to divide into identical totipotent cells that begin to specialize after several cycles of cell division, forming a hollow sphere of cells, called a blastocyst [99]. Research conducted at Stanford reveals that in fact there are some stem cell "guardians", surrounding areas composed of stem cells, that determine which type of ordinary cell they will specialize in [150]. As far as we know, the most interesting characteristics of the stem cells are the following:

- they can give rise to specialized cells;
- undifferentiated, they seem to have the ability to divide for apparently indefinite periods in culture;
- any of these cells can potentially develop into a fetus.

Not all the genetic program (the genome), which is carried by all cells, is executed by any cell; instead, portions of the genome are selected and executed through gene selection. In Embryonics, the process of a cell deciding which gene to execute based on the coordinates from within its local environment, i.e. the surrounding cells, determines its functionality, not unlike cellular specialization in biological organisms. Furthermore, the access to the whole genetic program provides our electronic cell with universality, much as biological stem cells have the potential of becoming any type of specialized cell. There are no limits on how large Embryonics cells can be. The universality of the cell then becomes actually the capacity of executing variable tasks – the more computationally complex the task, the more molecules required for the cell structure. Furthermore, by carefully selecting which portion of the code is to be executed by molecules, their universality is also assured – a molecule can effectively replace any other one by simply adapting its internal code.

**The Membrane.** Every living cell's inner mass is delimited from the surrounding environment by the cellular membrane, which also acts as an interface with the exterior, allowing a limited exchange of substances. If the biological world allows and depends on exchanging substances, the world of silicon has more restrictive rules: the material replacing substances, but nonetheless allowed and dependent on its exchange, is information. Much as in nature, where substances entertain life by carrying energy and information, in the world of silicon electronic signals carry in a similar way the same ingredients, entertaining artificial life.

The artificial membrane (also called the space divider) has a triple role [99]:

- It acts like a spatial barrier, logically separating resources (molecules) belonging to different cells and ensuring an individual identity with respect to the environment.
- It acts as a guide for the entering configuration, which contains the operative part of the genome. All molecules pertaining to a cell are configured with the corresponding gene in a chain-like process, which does not allow information to get outside the cell and be wasted; in a similar manner, the existence of the cellular membrane in biological cells restricts the access of the environment to their inner part both ways, thus preventing any unwanted loss of substances or possible intrusions to or from the environment.

- Its presence triggers a mechanism determining which gene is to be executed by each molecule. This can be seen as a specialization at the molecular level, the surrounding membrane directing the whole process in a similar way the stem cells' "guardians" control cell specialization in biology.

**Hierarchical Self-Testing and Self-Repairing.** As in nature, where multiple self-testing happens in each and every living being, Embryonics also relies on more than one such mechanism [99]. The very first self-testing procedure employs test vectors and applies to the core of the molecule. Both off-line and on-line testing procedures are used at molecular level, thus allowing for successful recovery, or healing [128].

Nature does a similar process at the lowest level possible, contained by the very intimate structure of the DNA itself. The two strands that make up the DNA continuously test each other by subtle chemical bonds, preventing and detecting a majority of possible errors. During operation, the molecular core can act as an active memory, much as the DNA does. The self-testing is ensured by "breaking" its internal register into two halves storing complementary data and acting in a similar way the two DNA strands do [99, 101].

Due to the vast complexity of biological organisms, healing (self-repairing) mechanisms can only be effective if the task is hierarchically decomposed and dealt with accordingly. The healing processes that take place in nature assume the cell is capable of fabricating the resources required. This is, of course, impossible with current technology, the only way of providing additional resources at the molecular level being as spares. Embryonics uses a hierarchy composed of two self-repairing mechanisms, described in detail throughout the thesis. Whenever the self-repair at the molecular level is overcome, there is a second self-repair attempt, at the cellular level. This is again inspired by nature, where foreign bodies (objects or mutating cells) are isolated and eventually eliminated.

## 5.4 Embryonics: Present and Future

This thesis represents a step forward in the development of the Embryonics project and contains the concepts and implementation of its most recent design concerning the electronic molecule. From the first explorations with phylogenetic processes in hardware with the FireFly machine [26], milestones for Embryonics were set by the implementation of the MicTree electronic cell [62, 66] and then by the MuxTree electronic molecules [67, 100, 128]. Adding extended means for memory storage has been set as a future target in [128], which was accomplished by the present work under the form of RAM-MuxTree [100, 102]. The Embryonics landscape includes a range of applications that have demonstrated the hardware implementations (Figure 5-1):

- MicTree: Von Neumann universal computer [56], BioWatch;
- MuxTreeSR: Von Neumann universal constructor, BioWatch [124, 125];
- RAM-MuxTreeSR: BioWall [134, 135], POEtic tissues [76, 136];
- BioCube (Figure 5-2), which explores self-replication processes in a 3D space [168].

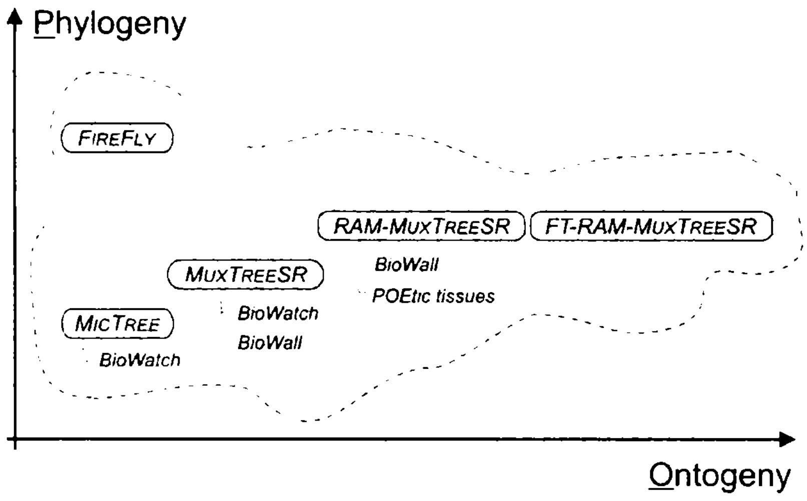


Figure 5-1: The current Embryonics landscape.

Figure 5-1 presents the current Embryonics landscape from an architectural point of view. Though not designed for implementing phylogenetic processes, the MuxTreeSR molecules offer a finer granularity for encoding the genome than the MicTree cell; the same comparison stands between MuxTreeSR and the RAM-MuxTreeSR, which introduces specialized memory structures. Over the ontogenetic axis, one can discover the maturing stages of the architectural concepts and hierarchy in Embryonics: the cellular level, with the architecture of the MicTree cell, was soon followed by the molecular level, with the MuxTreeSR molecules. With the RAM-MuxTreeSR molecule the ontogenetic processes now have a complete hardware platform that offers efficient ways of implementing both logic and memory.

Figure 5-2: The BioCube [Photo by André Badertscher].

The FTRAM-MuxTreeSR architecture brings new robustness to the Embryonics architecture, which we believe to be of prime relevance for the Embryonics future. As space exploration is set to regain the strong momentum it reached in the last decades of the XX$^{th}$ century, dependable computing machines seek alternative inspiration both in their design and computing paradigms. Though the existence of common field between seemingly so-different fields such as bio-inspired computing and quantum computing would appear unlikely, dependability requirements offer insights that bring these fields together: certain techniques evolved in one field may be well used by the other.

The future of Embryonics also appears to be manifold. Conceptually, there is still research to be done over fault-tolerance strategies at the cellular level; there has been little simulation and experimentation with reconfiguration at this level and the addition of the memory structures requires new strategies for error coding. Furthermore, there is room to improvement concerning the automatic assessment of fault-tolerance levels, from both qualitative and quantitative perspectives. Architecturally, complex computing systems (including quantum computers) should benefit from embryonic implementations; research efforts are already on their way toward building embryonic tissues, a considerable boost also coming from the acknowledgement of the field by the ITRS report [169]. From a physical point of view, implementations should go from the 2D to the 3D space, a first attempt being made by the BioCube [168].

# APPENDIX

# A HARDWARE IMPLEMENTATION

The research throughout this thesis was conducted in two places, the Logic Systems Laboratory (LSL) at the Swiss Federal Institute of Technology in Lausanne and the Advanced Computing Systems and Architectures (ACSA) at the "Politehnica" University of Timisoara. As an important part of the work carried out for this thesis, the birth of memory structures in Embryonics took place in the LSL, where it has become a tradition to have the designs implemented as actual hardware instead of limiting to software simulations only. Such an approach allows hands-on experimentation with physical devices, and often reveals aspects that could not be put into evidence in the software simulation phase and may benefit from further improvements.

This section presents the hardware implementation for the latest MuxTree design and some of the hardware configurations we employed in order to experiment with the macro-molecules.

## A.1 RAM-MuxTreeSR

In this section, we will describe the technical details concerning the implementation of memory structures (the macro-molecules) over the FPGA prototype developed for the Embryonics project. Rather than providing a complete, top-down view of the new electronic molecule, which we called RAM-MuxTreeSR, we will present the routing process involved at the molecular core and the hardware additions made in order to allow the existence of memory structures. We will give details concerning the existent mechanisms of self-repairing and self-replicating only insofar they are useful for a clearer understanding of the subject matter.

### A.1.1 Overview

Given the hierarchical structure of the hardware entities in Embryonics, one has to start with a molecular assembly in order to implement and experiment with any design. For this purpose, the platform of the Biodule 603 (from ***bio-inspired module***) was employed. Each such device is based on a Xilinx 4000 series FPGA and implements a single electronic molecule in a specially conceived package in order to offer the possibility of a puzzle-like assembly of many units (Figure A-1).

As described in Chapter 3, the Biodule 603 has been modified in order to offer a choice of two operating modes: the logic mode and the new memory operating mode, allowing two sub-modes (see SubSection 3.4.5):

- the short memory mode configures the molecule as an 8-bit storage space with switching capabilities;
- the long memory mode configures the molecule as a 16-bit storage space.

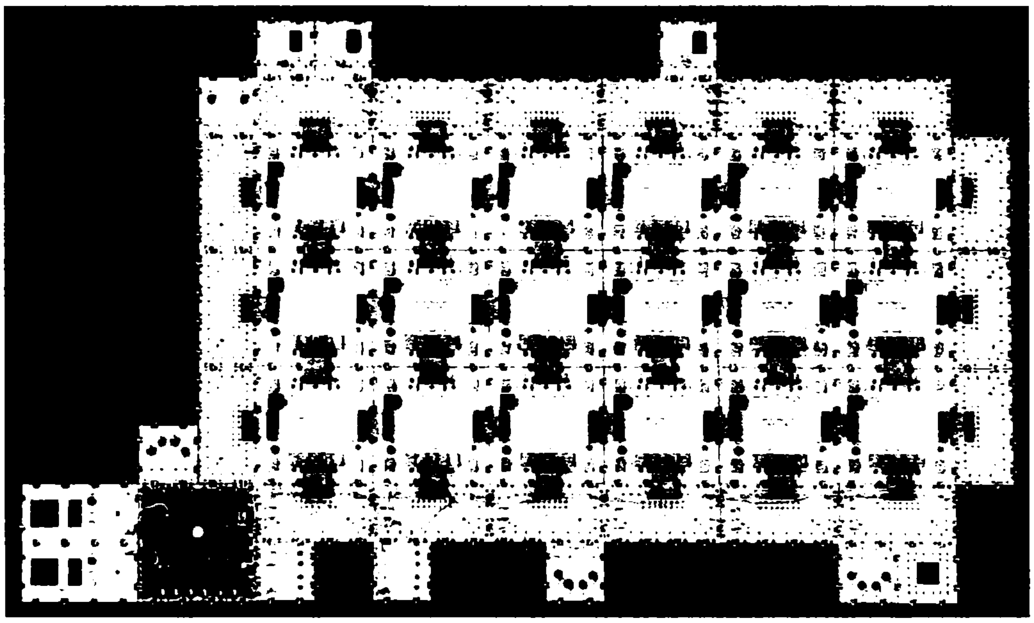


Figure A-1: The experimental platform, an array of 6x3 Biodules 603. [Photo by André Badertscher]

## A.1.2 Molecular Resources

The most important resource for implementing a memory structure is the *configuration register* (*CREG*); denoted as SR20RE in Figure A-2, it is implemented by a chain of 21 SR-type flipflops (SR0 being the least significant and SR20 being the most significant). The operating mode of the molecule is specified by the binary value stored by the SR20 flip-flop (denoted *M* in Section 3.5):

- if SR20 contains a logic "0" then the molecule operates in logic mode; the functional resources, the *switching block* (*SB*) and the *functional unit* (*FU*) are both available to the user through the configuration loaded into the *CREG*;
- if SR20 contains a logic "1" then the molecule operates in memory mode; the *FU* (denoted Q in Section 3.5) now has the role of configuring the memory operating sub-mode.

In order to enable any of the 8 or 16 bits choices of storage space, the original *CREG* design [128] was modified to use two groups of flip-flops, denoted as SR19-12 and SR 11-4 (shown in Figure A-3): they provide maximum storage together in the long memory mode (see SubSection 3.5.2.3) whereas in the short memory mode (see SubSection 3.5.2.2) only SR19-12 is used.

In Figure A-3 signal MEM enables normal memory operation, data shifting being allowed only for SR19-12 and SR11-4 according to the respective memory sub-mode. Multiplexers are used to route incoming data stream to SR19-12 through signal *TO_FF*, while output data stream is routed through signal *OUT_REG*. The outputs of SR19-12 and SR11-4 are selected for *OUT_REG* according to the memory sub-mode by using the content of the *FU*, provided by signal *FROM_FF*.

Several signals are necessary for enabling the memory mode. First, it is necessary to differentiate between the logic mode and the memory mode, since the nature of *CREG* information is operating mode dependent. In logic mode the information stored by the *CREG* is static, each bit of the MOLCODE being used in order to specify the logic configuration for the molecule (through the *FU*) and the interactions with its neighbors (through the *SB*). When in memory mode (regardless of the sub-mode), some of the information stored by the *CREG* become of dynamic nature, being shifted from one molecule to the next inside the macro-molecule, while some information remain static:

- in short memory mode, MOLCODE bits $MC_{21:20}$ and $MC_{11:0}$ are static (see SubSection 3.5.2.2);
- in long memory mode, MOLCODE bits $MC_{21:20}$ and $MC_{3:0}$ are static (see SubSection 3.5.2.3).

Second, there are functional requirements that have to be settled in order to ensure a proper molecular assembly that will make up for a complete macro-molecule. The data shifting process is synchronous to the functional clock FCK, selected by an active L_EN signal (L_EN="1") and enabled by an active MEM signal (MEM="1"). The data path is determined by the relative position of each molecule inside the macro-molecule; MOLCODE bits $MC_{3:1}$ are used in order to determine whether a certain molecule is positioned at the north or south border, or if it is a corner at the south border. Signals locating the molecule's position are provided by the CONF_BITS block, presented in Figure A-4 and also shown in Figure 3-14 (active on logic "1"):

- *BS* is active when the molecule is located at the south border of a macro-molecule containing at least two columns, but it is not a corner;
- *RC* is active when the molecule is located at the right corner to the south of a macro-molecule containing at least two columns;
- *LC* is active when the molecule is located at the left corner to the south of a macro-molecule containing at least two columns;
- *BC* is active when the molecule is located at the bottom of a macro-molecule containing one column only, also called a *memory column*;
- *BN* is active when the molecule is located at the at the north border of a macro-molecule;
- *BSX* is active when the molecule is part of the bottom row of any macro-molecule;
- *NB* is active when the molecule is located neither at the north border, nor at the south border of the macro-molecule;
- *BSRC* is active when the molecule is located either at the south border, or it is the right corner of a macro-molecule containing at least two columns.

The propagation of data is ensured through a group of multiplexers driven by signals provided by the CONF_BITS logic block. Because the reconfiguration mechanism that ensures the self-repair at the molecular level also uses multiplexers, the memory data propagation process was required to be settled first. Therefore, whenever a memory molecule is situated in the bottom of a macro-molecule, information will be shifting in from the *C_I_W* signal (*WIC* in Figure 3.5B) and whenever a memory molecule is situated at the north border of a macro-molecule, information will be shifted out through the *C_O_S* signal (*SOC* in Figure 3.5C).

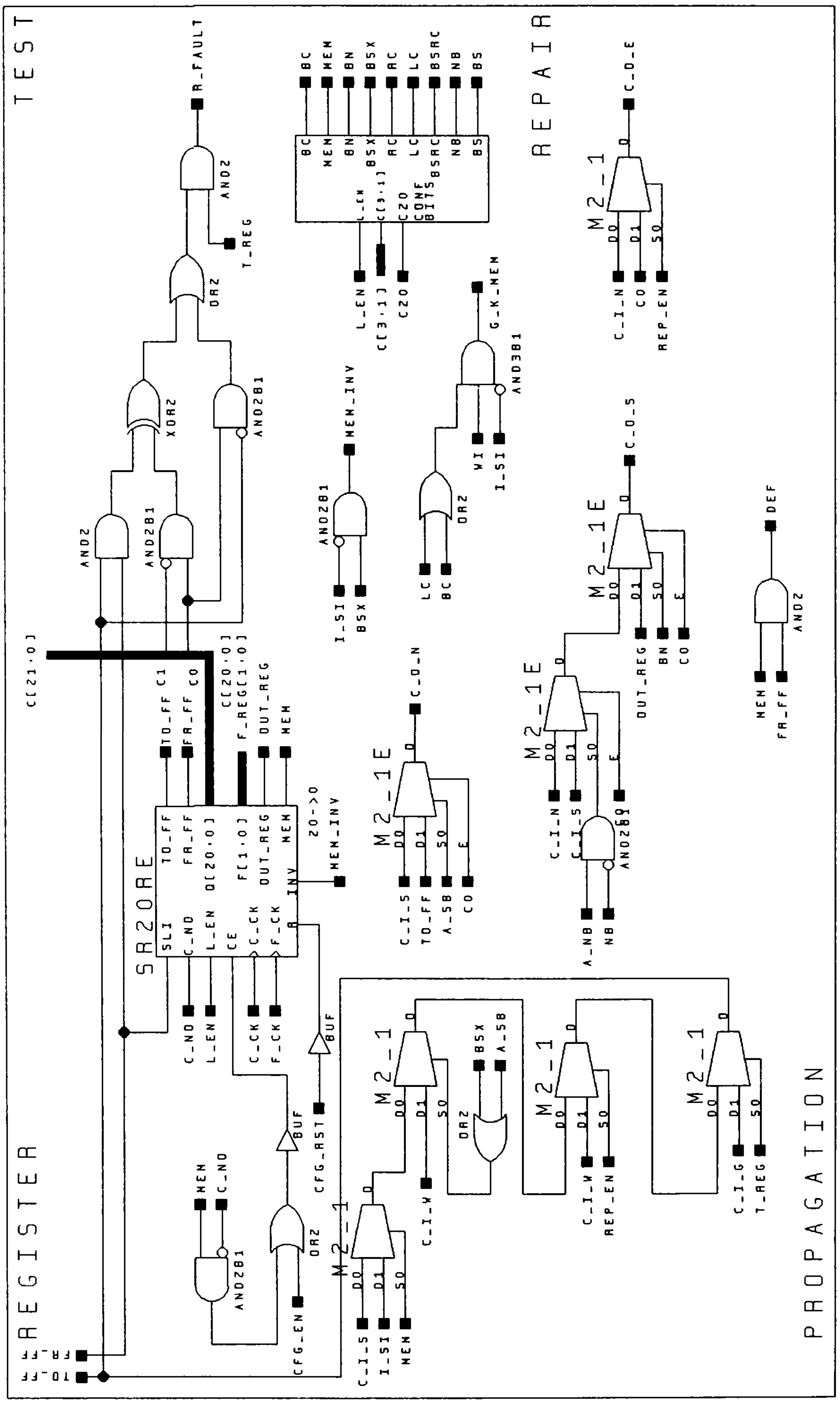


Figure A-2: Top view of the configuration register CREG.

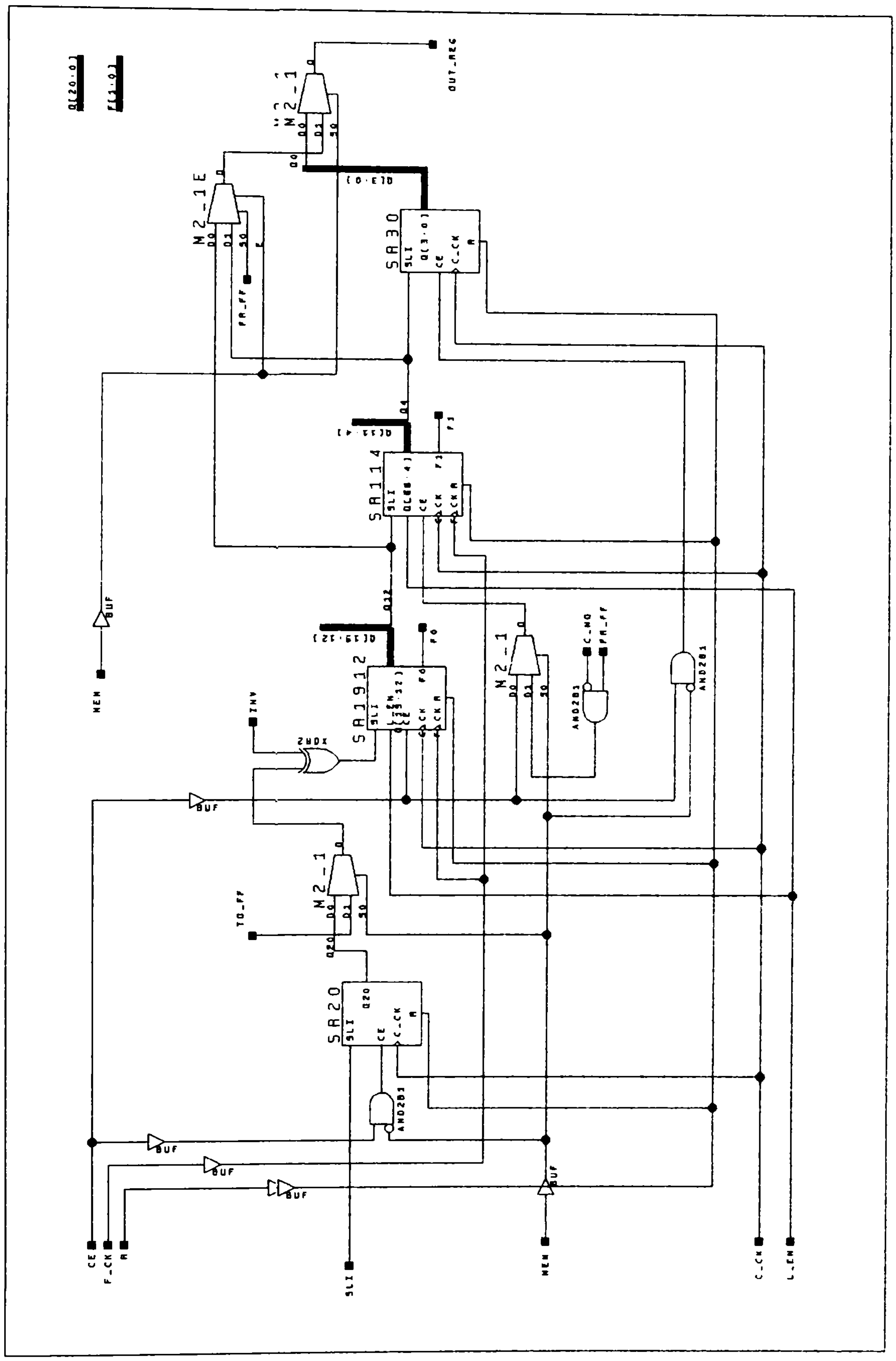
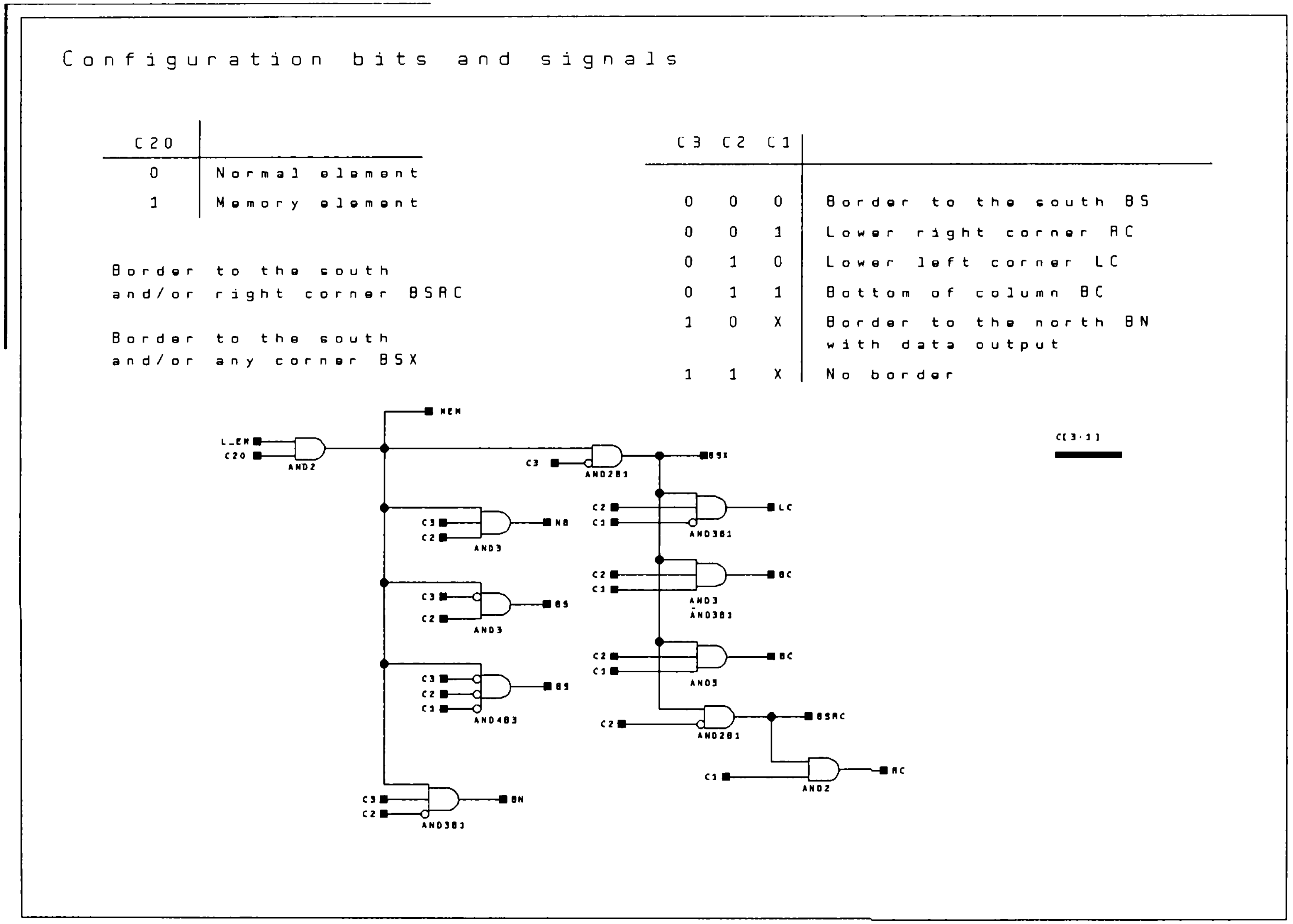


Figure A-3: Internal schematic for the configuration register CREG.

In Figure A-5 the coordinates of data shifting when a memory molecule is part of the bottom of a macro-molecule are shown. When the molecule is the bottom of a memory column, the information will be shifted inside the molecule


Configuration bits and signals

| C20 | |
|---|---|
| 0 | Normal element |
| 1 | Memory element |

Border to the south and/or right corner BSRC

Border to the south and/or any corner BSX

| C3 | C2 | C1 | |
|---|---|---|---|
| 0 | 0 | 0 | Border to the south BS |
| 0 | 0 | 1 | Lower right corner RC |
| 0 | 1 | 0 | Lower left corner LC |
| 0 | 1 | 1 | Bottom of column BC |
| 1 | 0 | X | Border to the north BN with data output |
| 1 | 1 | X | No border |

Figure A-4: Internal schematic for the configuration bits block CONF_BITS.

through signal *I_C_NI* (north input). If the molecule is a left corner, the information will enter through signal *EI* (east input) also shown in Figure 3-6A. If neither is the case, then information shifting is done through signal *WI* (west input), also shown in Figure 3-6B.

Shifting the data out of the memory molecule is a process similar to that of shifting the data in with respect to being driven by multiplexers. In order to assemble a complete macro-molecule, both input and output connections have to be properly configured. The output signals that need to be redefined are *EO* and *C_EO* (to the east), *WO* (to the west).

The implementation of signal *WO* is presented in Figure A-5, routing signal *NEW_EI* when the molecule is located at the bottom of the macro-molecule, but it is neither a left corner, not a bottom of column (signal *BSRC* active). Signal *NEW_EI* is obtained from signals *EI* (east input) and *I_C_NI* (north input), multiplexed through signal *RC*. When *RC* is active (the molecule is a right corner) the *WO* signal will take the value of the *I_C_NI* signal, the process being shown in Figure 3-16 where signal *NIC* (*I_C_NI*) goes to *WOUT* (*WO*). If *RC* is not active, then the *WO* signal will take the value of the *EI* signal (shown as *EIN* in Figure 3-16).

The implementation of signal *EO* is presented in Figure A-6. If the molecule is located at the south border of the macro-molecule (signal *BSX* active) then the value of signal *EO* will take the value of signal *I_C_NI* (*NIC* in Figure 3-18).

Data output ports are available at each molecule situated at the north border of a macro-molecule through the *N_OUT* signal shown in Figure A-7. The storage data is output from the *CREG* by using a multiplexer driven by the *MEM* signal (which is active when in memory mode only) and reaches the data output port (also shown in Figure 3-13 as using the *NOUT* signal).

### A.1.3 The *HOLD* signal

The *HOLD* mechanism was introduced in order to control the data shifting process inside a macro-molecule. When the *HOLD* signal is active (on logic "1") the memory data shifting process is inhibited. Each macro-molecule has an input *HOLD* signal to the molecule situated at the left corner or at the bottom of column, which is than driven both horizontally (from signal *WIC* to signals *EOC* and *NOC*) and vertically (from signal *SIC* to signal *NOC*).

The implementation of signals *C_NO* (signal *NOC* in Figure 3-20) and *C_EO* (signal *EOC* in Figure 3-20) is presented in Figure A-6. If the molecule is not located at the bottom of the macro-molecule then its value is given by signal *C_SI* (signal *SIC* in Figure 3-20); its value is determined by signal *WI* (signal *WIC* in Figure 3-20) if the molecule is either the left or the right corner or by signal C_WI otherwise.

If the molecule is located at the south border of the macro-molecule, but it is not the right corner (signal *BSX* active but signal *RC* inactive) then if the molecule is not the left corner or the bottom of a memory column the value of signal *C_EO* will take over the value from signal *C_WI* (signal *WIC* in Figure 3-20). On the other hand, if the molecule is located at the left corner or is the bottom of a memory column, the *C_EO* signal will drive the value of signal of signal *WI*; this is different than suggested in Figure 3-20, being changed with the implementation of the Fault-Tolerant RAM-MuxTree concepts and shown in Figure 4-24.

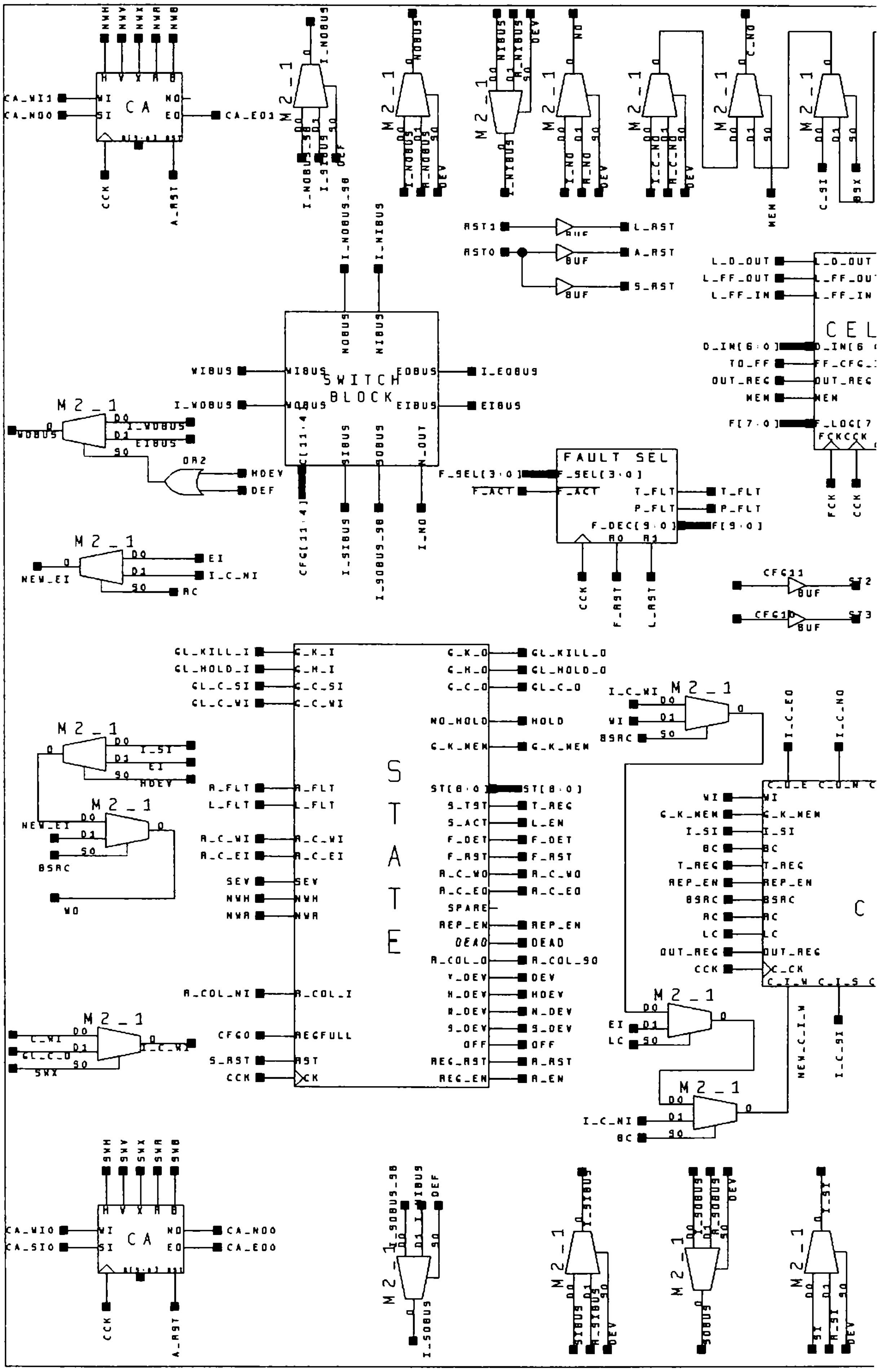


Figure A-5: Top view of the RAM-MuxTree molecule (left half).

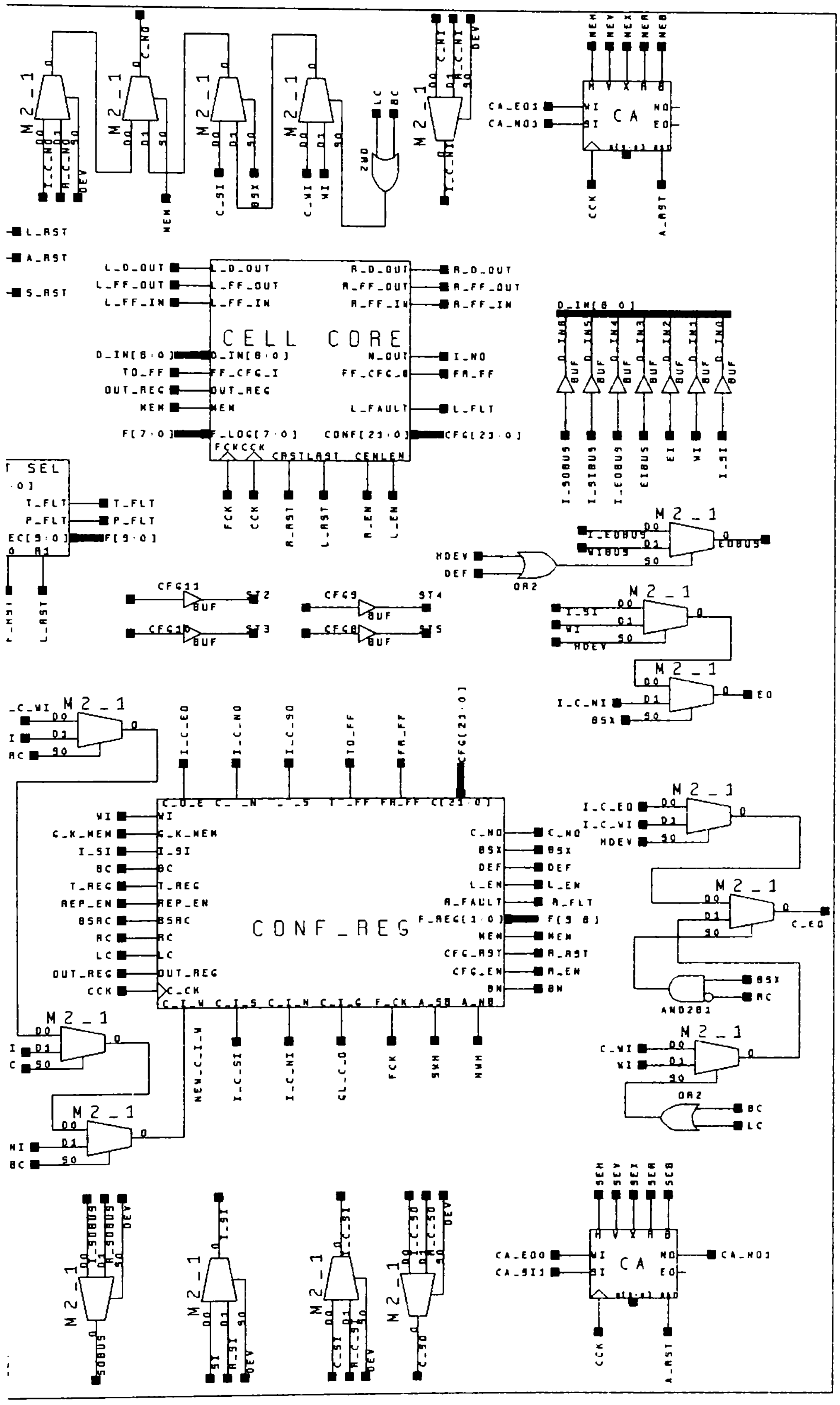


Figure A-6: Top view of the RAM-MuxTree molecule (right half).

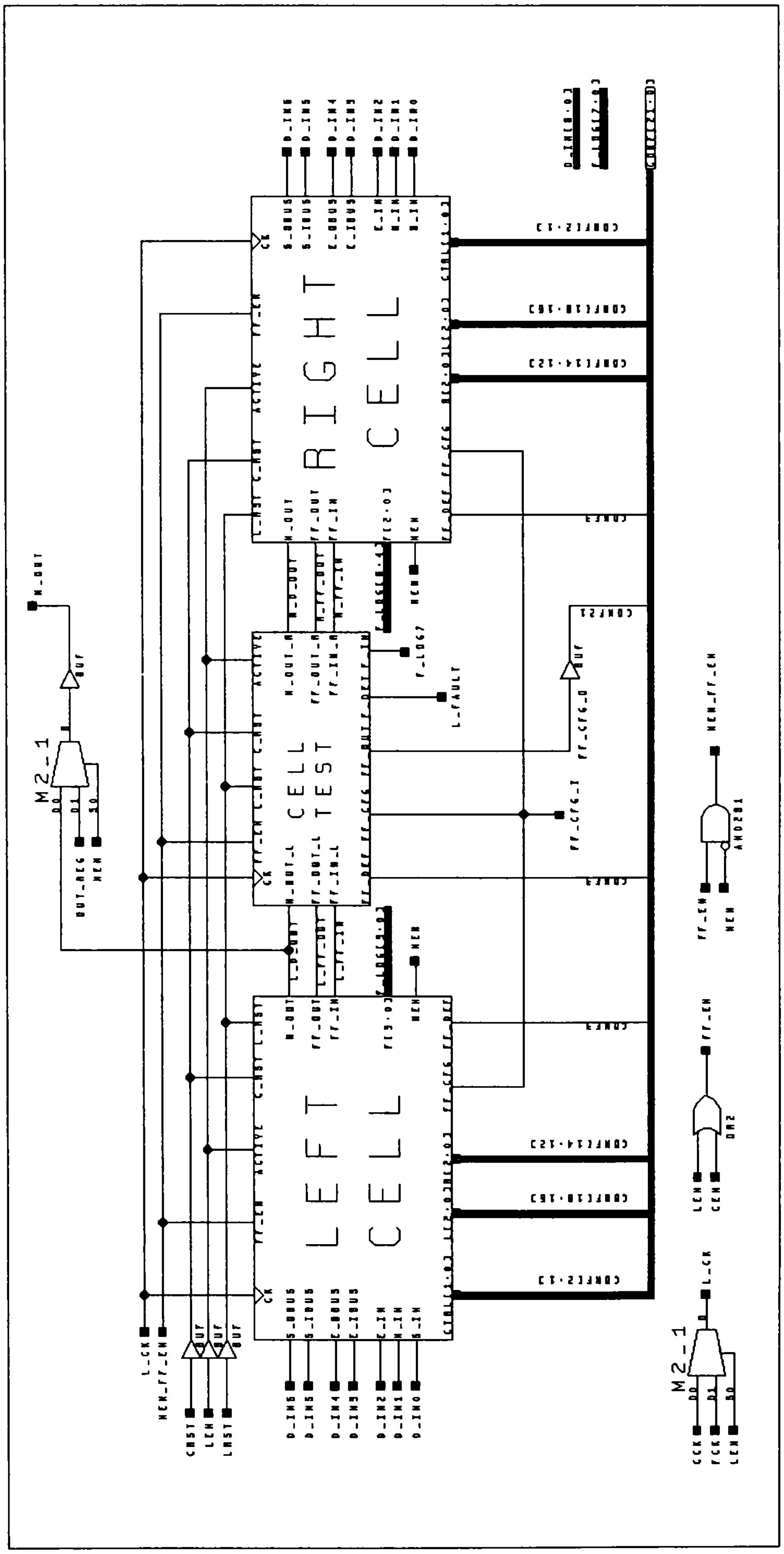


Figure A-7: Top view of the Functional Unit (FU).

## A.2 FTRAM-MuxTreeSR

The FTRAM-MuxTree molecule extends the design of the RAM-MuxTree by allowing the implementation of error-correcting codes within structures following the general architecture shown in Figure 4-16.

### A.2.1 Overview

There are two steps that have to be carried out in order to achieve fault tolerance inside a macro-molecule:

- fault location is enabled by the *Error Correcting Logic* (*ECL*), a structure that computes redundant data based on the genome data (Storage Data in Figure 4-16), compares the results with the reference (Check Data in Figure 4-16), and also generates the necessary *HOLD* signals;
- fault correction is also enabled by the *ECL*, which also generates the *INV* signals used in order to flip the value of erroneous bits. Taking into consideration the particular data path information follows inside a macro-molecule, data can be accessed at the north border of each macro-molecule, while the correction process takes place one clock cycle later, at the south border of the macro-molecule.

The operations that are to be carried out inside a fault-tolerant macro-molecule and their respective signal configurations are shown in Table 4-6.

### A.2.2 The *INV* signal

In order to describe the role of the *INV* signal we will revisit Figure 4-21, which presents two essential moments in time: data that is accessed through the data output ports at the north border of the macro-molecule at time *t* reaches the south border the next clock cycle, at time *t*+1. For this particular situation, the macro-molecule is a 3x4 array. Therefore, according to SubSection 4.3.1, at time t data is read under the form of $Data(t)=\begin{pmatrix}c_{31}^F & c_{32}^F & c_{33}^F\end{pmatrix}$ and, after computing the Hamming equations it results that the middle bit $\left(c_{32}^F\right)$ is erroneous. Therefore, the error matrix is $E(t)=\begin{pmatrix}0 & 1 & 0\end{pmatrix}$ and the relationship between the data accessed at time *t* and the data that needs to be restored results as: $Data(t)=Original\_Data(t)\oplus E(t)$. From this equation the data can be restored by $Original\_Data(t)=Data(t)\oplus E(t)$. We could also view the error matrix as a containing complemented *INV* signals; wherever there is a logic "1", the respective *INV* signal should be activated for the next clock cycle (*INV* signals are active on logic "0"). Therefore at time *t*+1 signal $INV_2$ has to be activated in order to recover the damaged data: $E(t)=\begin{pmatrix}1-INV_1 & 1-INV_2 & 1-INV_3\end{pmatrix}=\begin{pmatrix}0 & 1 & 0\end{pmatrix}$.

Each memory molecule located at the south border has an access port providing the local *INV* signal, shown in Figure A-3 and derived from the *MEM_INV* signal from Figure A-2.

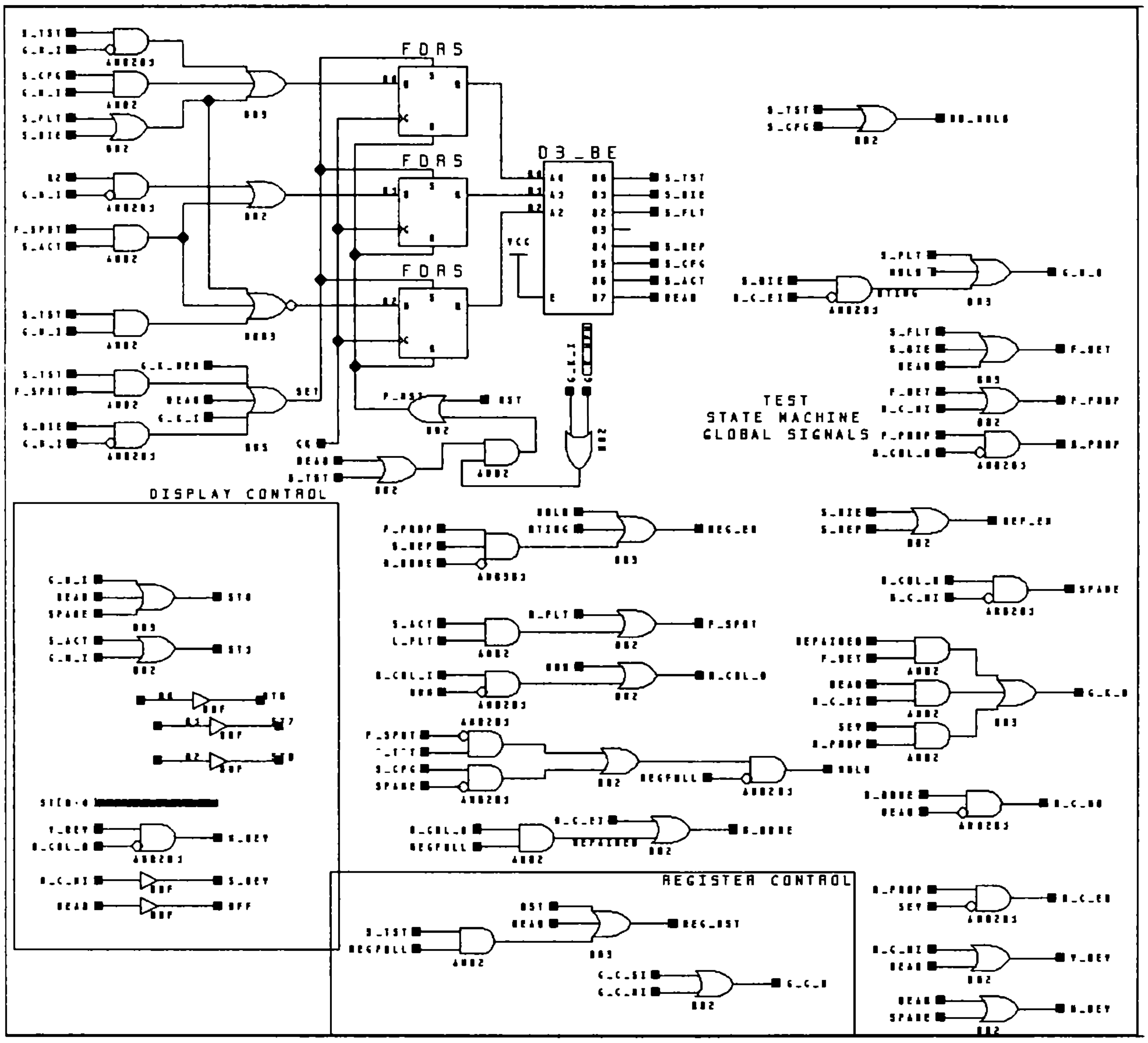


Figure A-8: Block schematic of a molecule's state automaton.

## A.2.3 The *KILL* signal

The occurrence of a non-recoverable error is also possible during normal operation of a fault-tolerant macro-molecule. If such is the case, the *ECL* triggers the *KILL* signal, which will effectively disable the entire cell and start the self-repairing processes at the higher, cellular level (Figure 4-35). The *KILL* signal is activated when the left corner of bottom of column molecule both the *INV* and the *HOLD* signals all active (*INV* set to logic "0" and *HOLD* set to logic "1"); the configuration for setting the *KILL* signal is given in Table 4-6. The *KILL* signal is shown in Figure A-2 as *G_K_MEM*.

In order to produce the same effect at molecular level, the *KILL* signal generated by the *ECL* (when in memory mode) has to be combined with the same signal produced when in logic mode. The state of each molecule [128] is stored by the automaton shown in Figure A-8, in which the *G_K_MEM* signal has been added in order to also reflect the death of the cell due to non-recoverable memory errors.

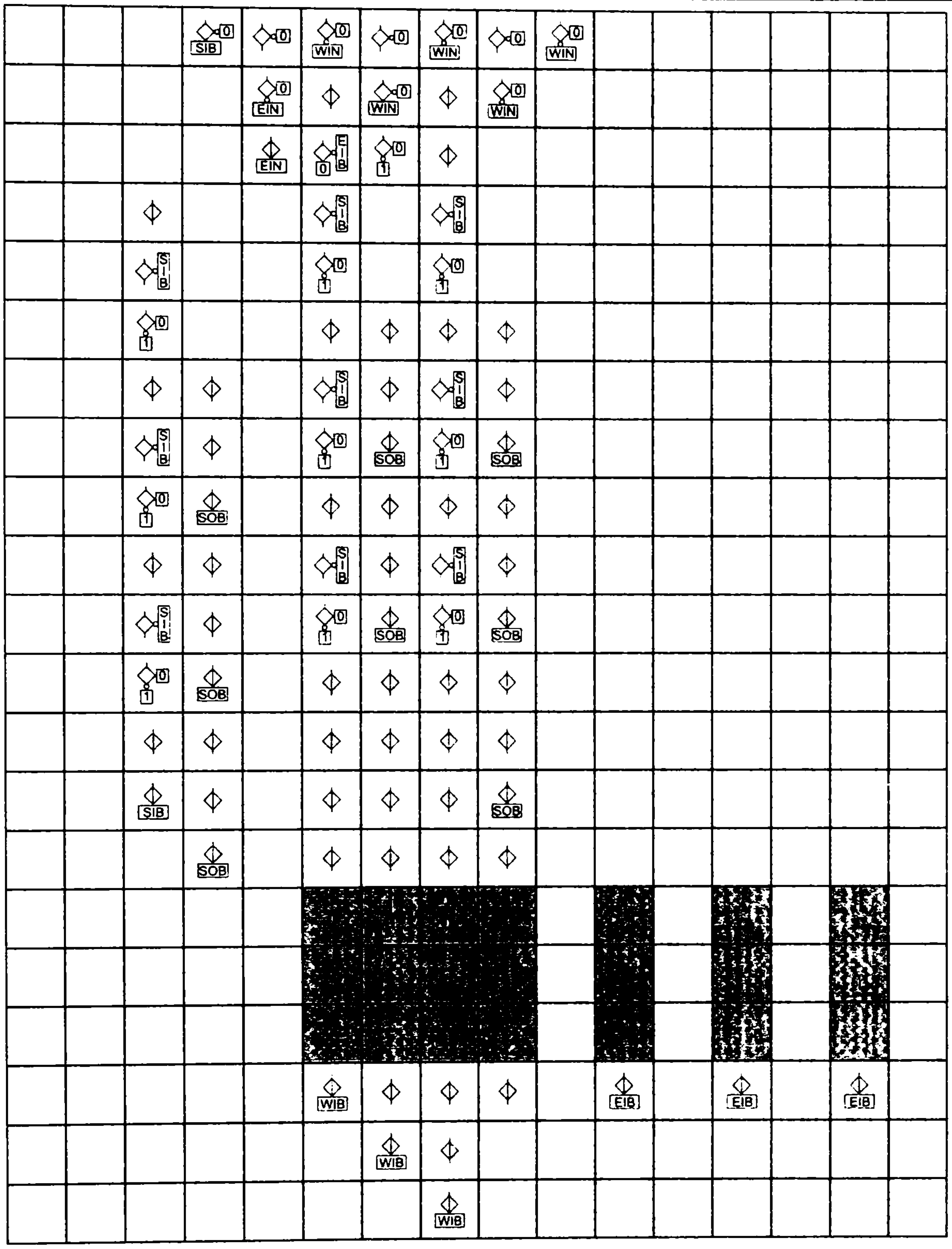


Figure A-9: Map of a fault tolerant macro-molecule (logic level).

## A.2.4 *ECL* Implementation for a (7,3) Hamming SEC code

The block schematic for an *ECL* implementing a single error correcting code was shown in Figure 4-27. All molecules that implement the ECL are operating in logic mode. Figure A-9 presents the logic level of the *ECL* (the configurations of the *FU* units), while Figure A-10 presents the bus level of the *ECL* (the configurations of the *SB* units) of all molecules that make up the *ECL* unit [98].

The notations are those used in [67]. Due to the fine graininess in Embryonics (which was a target from the very beginning), a significant number

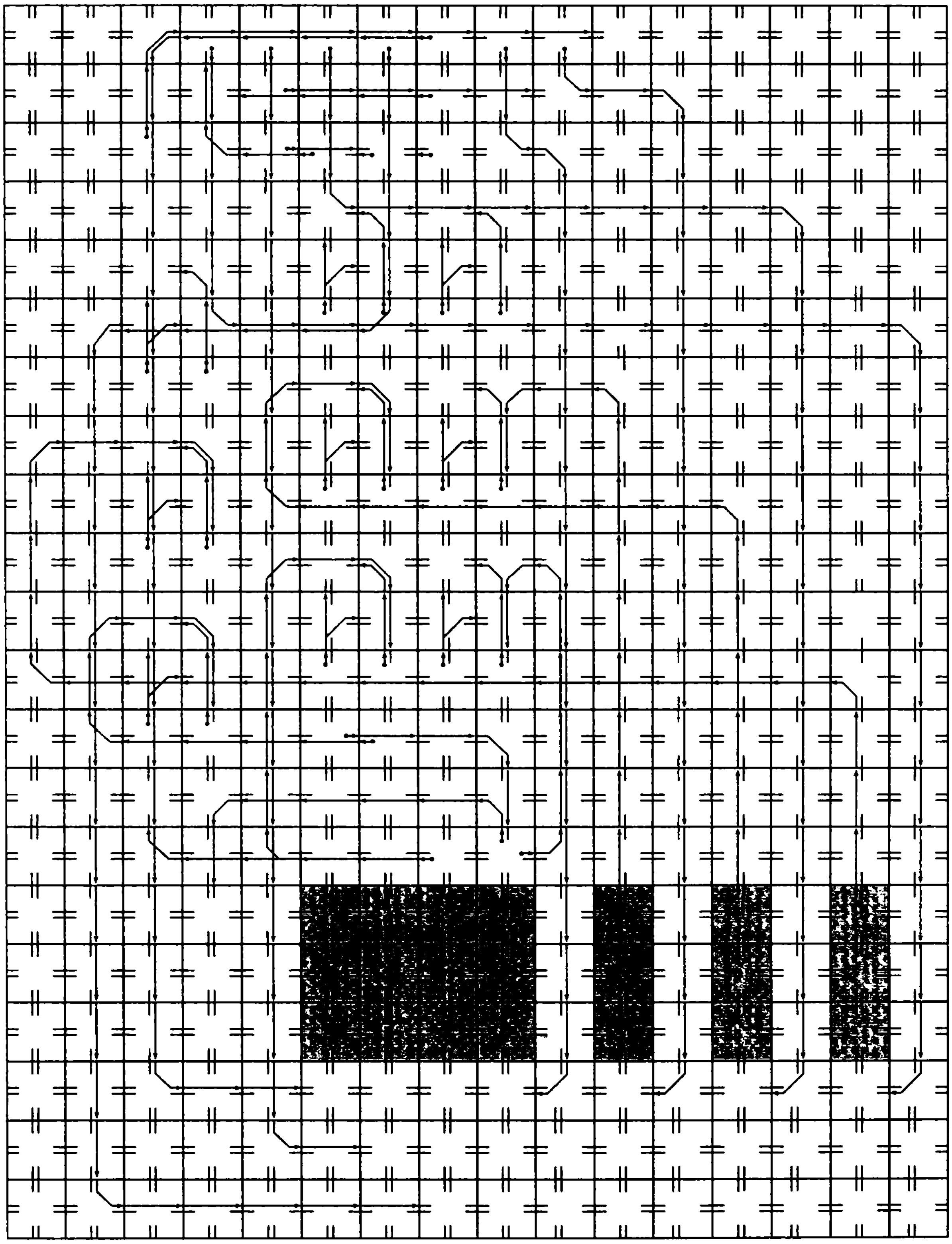


Figure A-10: Map of a fault tolerant macro-molecule (bus level).

of molecules (that is, 6 of them) have to be used in order to implement a single EX-OR gate. This fact contributes to a significant hardware overhead that could be improved by adding a programmable EX_OR gate to the *FU* units.

## A.2.5 Experiments with Macro-Molecules

Experimentation has been done with macro-molecules of different dimensions. With a number of 18 Biodules available, experiments were conducted with a 3x1 memory column (shown in Figure A-11), a 3x3 macro-molecular array (shown in

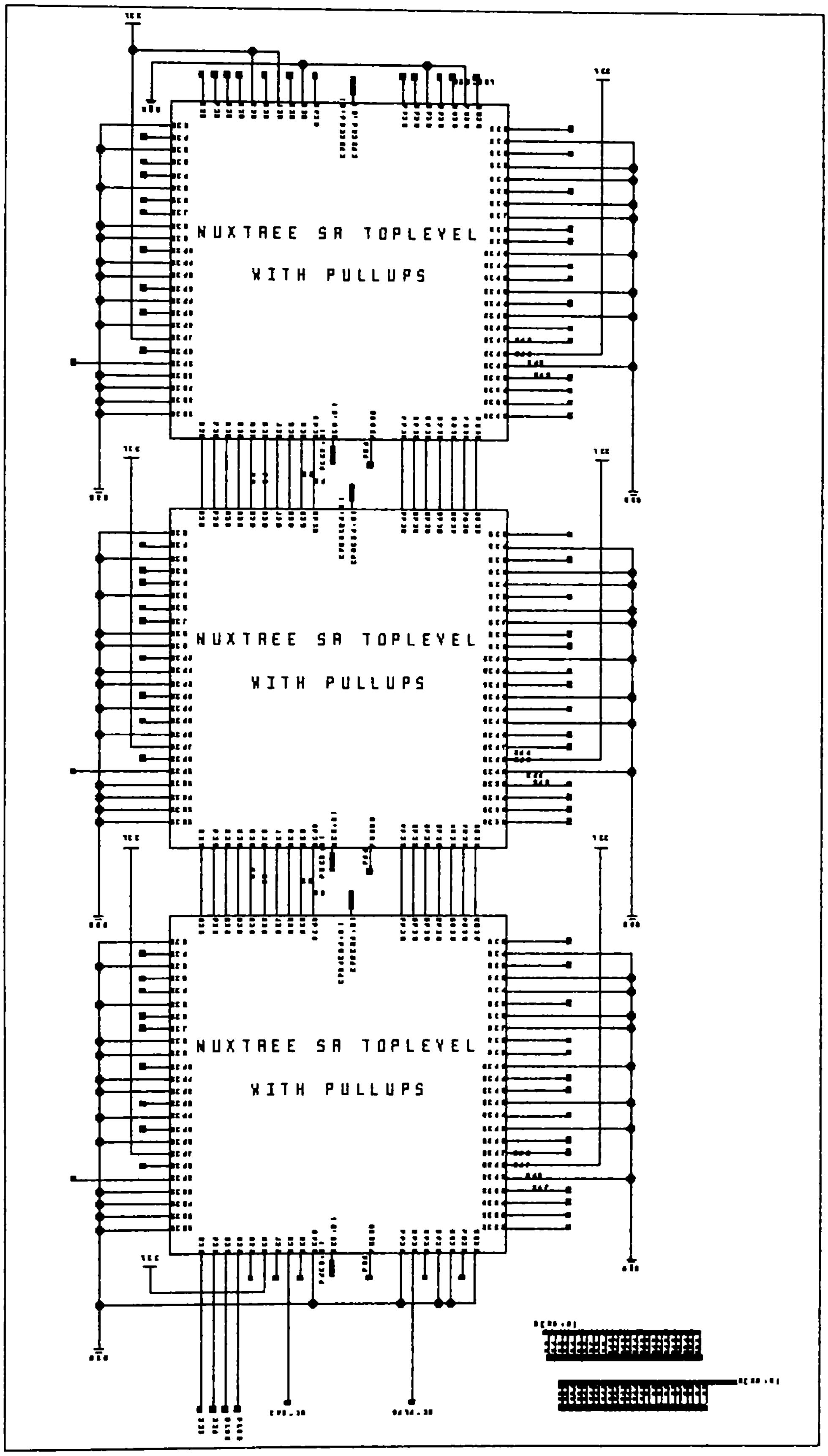


Figure A-11: Top view of a 3x1 memory column.

Figure A-12), and a 6x3 macro-molecular array.

The two essential processes for the Embryonics architecture, namely self-repair and self-replication were preserved in the new FTRAM-MuxTree design. While the self-repairing mechanisms required a significant change in the way

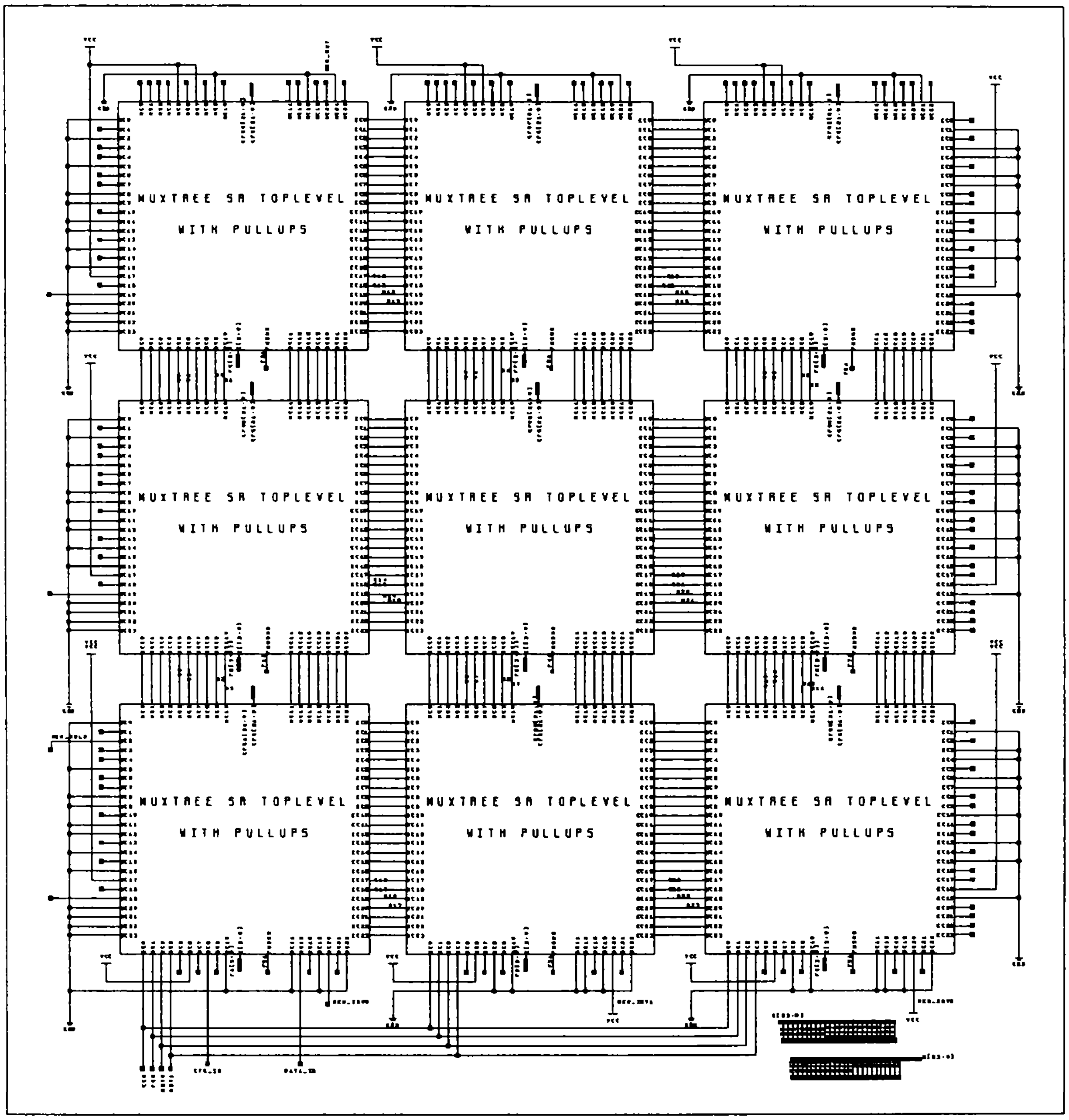


Figure A-12: Top view of a 3x3 macro-molecule.

they operated in order to ensure memory data protection, this was not the case when the self-replication mechanisms were concerned. Since the macro-molecules are configured by the MOLCODE (as are machines made of logic molecules), the self-replication process extends successfully upon the new memory structures.

At this moment, the design of FTRAM-MuxTree molecule may be considered as final: there are logic molecules for implementing combinational and sequential logic machines, and there are memory molecules required by micro-programmed machines. With self-repairing capabilities accommodating both molecular operating modes, it remains for the future to have complex computing systems implemented onto the Embryonics platform as extremely dependable machines.

# REFERENCES

[1] M. Abramovici, M. A. Breuer, A. D. Friedman. *Digital Systems Testing and Testable Design*. Wiley-IEEE Press, 1994.

[2] M. Abramovici, C. Stroud. "No-overhead BIST for FPGAs". In *Proc. 1st IEEE International On-Line Testing Workshop*, pp. 90-92, 1995.

[3] A. Avizienis, J.-C. Laprie, B. Randell, C. Landwehr. "Basic Concepts and Taxonomy of Dependable and Secure Computing". *IEEE Transactions on Dependable and Secure Computing*, 1(1), Jan-Mar 2004, pp.11-33.

[4] C. Bernard, D. Mange, A. Stauffer. Catalogue logidules. Laboratoire de systèmes logiques, Ecole Polytechnique Fédérale de Lausanne, Oct. 1994.

[5] M. Blaum, R. Goodman, R. McEliece. "The Reliability of Single-Error Protected Computer Memories". *IEEE Transactions on Computers*, vol.37, no.1, pp.114-119, January 1988.

[6] D.W. Bradley, A.M. Tyrrell. "Immunotronics: Hardware Fault Tolerance Inspired by the Immune System". *Proceedings 3rd International Conference on Evolvable Systems*, LNCS 1801, pp. 11-20, Springer-Verlag, April 2000.

[7] S.D. Brown, R.J. Francis, J. Rose, Z.G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publishers, Boston, 1992.

[8] A.W. Burks, ed. *Essays on Cellular Automata*. University of Illinois Press, Urbana, IL, 1970.

[9] C. Carmichael. *Triple Module Redundancy Design Techniques for Virtex FPGAs*. Xilinx Application Note XAPP197, June 2001.

[10] C. Carmichael, E. Fuller, J. Fabula, F. De Lima. "Proton Testing of SEU Mitigation Methods for the Virtex FPGA". *Military and Aerospace Programmable Logic Devices Conference (MAPLD)*, USA, 2001.

[11] F.H.C. Crick. "On Protein Synthesis". *Symposia of the Society for Experimental Biology*, 12, 1958, pp. 548-555.

[12] D.R. Croley, H.B. Garett, G.B. Murphy, T.L. Garrard. "Solar Particle Induced Upsets in TDRS-1 Attitude Control System RAM During the October 1989 Solar Particle Events". *IEEE Transactions on Nuclear Science*, NS-42, 1489 (1995).

[13] C. Dai. "Soft Errors in Present and Future. Semiconductor Technologies and Products". *MSD (Materials Structures and Devices) C2S2 (Center for Circuits, Systems, Software) Workshop*, Berkeley CA, 2002.

[14] C. Dai, N. Hakim, S. Hareland, J. Maiz, S.W. Lee. "Alpha-SER Modeling and Simulation for Sub-0.25um CMOS Technology". *Symposium on VLSI Technology Digest of Technical Papers*, 1999.

[15] B. Davari. CMOS Technology Scaling, 0.1um and Beyond. IEDM, 1996.

[16] S.Durand, A.Stauffer, D.Mange. *Biodule: An Introduction to Digital Biology*. Technical Report, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, 1994.

[17] F. Faccio, G. Anelli, M. Campbell, M. Delmastro, P. Jarron, K. Kloukinas, A. Marchioro, P. Moreira, E. Noah, W. Snoeys, T. Calin, J. Cosculluela, R.

Velazco, M. Nicolaidis, A. Giraldo. *Total Dose and Single Event Effects (SEE) in a 0.25m CMOS Technology*. CERN-LHCC RD49 Project, Geneva, Switzerland.

[18] B. Feingold, P. Layton. "Total Dose and Single Event Effects Testing of a Commercial 0.8um CMOS Gate Array Process". The *2nd Military and Aerospace Programmable Logic Device International Conference (MAPLD99)*, 1999.

[19] T.C. Fogarty, J.F. Miller, P. Thomson. "Evolving digital logic circuits on Xilinx 6000 family FPGAs". In P.K. Chawdhry, R. Roy, and E.K. Pant (eds.), *Soft Computing in Engineering Design and Manufacturing*, Springer-Verlag, 1998, pp. 299-305.

[20] T. Francke, M. Boezio, G. Barbiellini, G. Basini, R. Bellotti, U. Bravar, F. Cafagna, P. Carlson, M. Casolino, M. Castellano, M. Circella, C. De Marzo, M.P.De Pascale, N. Finetti, R.L.Golden, C. Grimani, M. Hof, W.Menn, J.W. Mitchell, A. Morselli, J.F. Ormes, P. Papini, S. Piccardi, P. Picozza, M. Ricci, P. Schiavon, M. Simon, R. Sparvoli, P. Spillantini, S.A. Stephens, S.J. Stochaj, R.E. Streitmatter, M. Suffert, A. Vacchi, N. Weber, N. Zamp. "A New Measurement of the Atmospheric Proton and Muon Fluxes". In *Proc. 26th Intl. Cosmic Ray Conference*, vol. 2, 1999, pp. 80-83.

[21] L.B. Freeman. "Critical Charge Calculations for a Bipolar SRAM Array". *IBM Journal of Research and Development*, vol.40, no.1, January 1996, pp. 119-129.

[22] J. Gaisler. "Evaluation of a 32-Bit Microprocessor with Built-In Concurrent Error Detection". In *27th Annual Intl. Symposium on Fault-Tolerant Computing (FTCS-27)*, 1997, pp. 42-46.

[23] J. Gaisler. *Concurrent Error Detection and Modular Fault-Tolerance in a 32-Bit Processing Core for Embedded Space Flight Applications*. On-board Data Division, European Space Research and Technology Centre, 2200 AG Noordwijk, The Netherlands.

[24] H. deGaris. "CAM-BRAIN: The Evolutionary Engineering of a Billion Neuron Artificial Brain by 2001 Which Grows/Evolves at Electronic Speeds Inside a Cellular Automata Machine (CAM)". *Towards Evolvable Hardware*, Springer Verlag, New York, 1996.

[25] S.F. Gilbert. *Developmental Biology*. Sinauer Associates, Inc., MA, 3rd ed., 1991.

[26] M. Goeke, M. Sipper, D. Mange, A. Stauffer, E. Sanchez, M. Tomassini. "Online Autonomous Hardware". In T. Higuchi, M. Iwata, W. Liu, eds. *Proceedings International Conference on Evolvable Systems: From Biology to Hardware* (*ICES96*), LNCS 1259, Springer-Verlag, Berlin, 1997, pp. 96-106.

[27] A.J. van de Goor. *Testing Semiconductor Memories. Theory and Practice*. John Wiley and Sons, 1991.

[28] C. S. Guenzer, R. G. Allas, A. B. Campbell, J. W. Kidd, E. L. Petersen, N. Seeman, E. A. Wolicki. "Single Event Upsets in RAM's Induced by Protons at 5.2 GeV and Protons and Neutrons below 100 MeV". *IEEE Trans. Nuclear Science*, vol. NS-27, pp. 1485–1489.

[29] C. S. Guenzer, E. A. Wolicki, R. G. Allas. "Single Event Upsets of Dynamic RAM's by Neutrons and Protons". *IEEE Trans. Nuclear Science*, vol. NS-26, 1979, pp. 5048.

[30] R. Gupta. "Addressing problems of the large". In *IEEE Design & Test of Computers*, July-August 2003, pp. 3.

[31] F. Hanchek, S. Dutt. "Methodologies for Tolerating Cell and Interconnect Faults in FPGAs". *IEEE Transactions on Computers*, vol. 47, no. 1, January 1998.

[32] J.P. Hayes. *Introduction to Digital Logic Design*. Addison-Wesley, Reading, MA, 1993.

[33] P. Hazucha, C. Svensson. "Impact of CMOS Technology Scaling on the Athmosferic Neutron Soft Error Rate". *IEEE Trans. on Nuclear Science*, vol.47, no.6, December 2000, pp. 2586-2594.

[34] B.G. Henson, P.T. McDonald, W.J. Stapor. *SDRAM Space Radiation Effects Measurements and Analysis*. Innovative Concepts Incorporated, McLean, VA 22102, www.innocon.com.

[35] J.H. Holland. *Adaptation in Natural and Artificial Systems*. The University of Michigan Press, Ann Arbor, 1975.

[36] J.W. Howard Jr., K.A. LaBel, M.A. Carts, R. Stattel, C.E. Rogers, T. Irwin. "Update on Total Dose and Single Event Effects Testing of the Intel Pentium III (P3) and AMD K7 Microprocessors". The *4th Military and Aerospace Programmable Logic Device International Conference (MAPLD01)*, 2001.

[37] M.Y. Hsiao. "A Class of Optimal Minimum Odd-Weight-Column SEC-DED Codes". *IBM Journal of Research and Development*, vol.14, pp.395-401, July 1970.

[38] M. Huhtinen, F. Faccio. "Computational method to estimate Single Event Upset rates in an accelerator environment". *Nucl.Instrum.Meth.*A450, 2000, pp.155-172.

[39] B.W. Johnson. *Design and Analysis of Fault-Tolerant Digital Systems*. Addison-Wesley, 1989.

[40] J.O. Kephart. "A Biologically Inspired Immune System for Computers". In R. Brooks, P. Maes, eds, *Artificial Life IV*, MIT Press, 1994.

[41] J.R. Koza. *Genetic Programming*. The MIT Press, Cambridge, MA, 1992.

[42] K. LaBel, A. Moran, D. Hawkins, A. Sanders, C. Seidleck, H. Kim, J. Forney, E.G. Stassinopoulos, P. Marshall, C. Dale, R. Barry. *Single Event Effect Proton And Heavy Ion Test Results in Support of Candidate NASA Programs*. Whitepaper, 1995.

[43] J. Lach, W.H. Mangione-Smith, M. Potkonjak. "Efficiently Supporting Fault-Tolerance in FPGAs". In *Proc. FPGA 98*, Monterey, CA, February 1998, pp.105-115.

[44] P.K. Lala. *Self-Checking and Fault-Tolerant Digital Design*. Morgan Kaufmann, 2001.

[45] P.K. Lala. *Digital Circuit Testing and Testability*. Academic Press, 1997.

[46] P.K. Lala. *Fault Tolerance and Fault Testable Hardware Design*. Prentice Hall, 1985.

[47] C.G. Langton. "Self-Reproduction in Cellular Automata". *Physica* 10D, pp.135-144, 1984.

[48] J.A. Lesniak, W.R. Weber. "The Charge Composition and Energy Spectra of Cosmic-Ray Nuclei". *Astrophysics Journal* 223, 1978.

[49] D. Levi, S. Guccione. "GeneticFPGA: A Java-Based Tool for Evolving Stable Circuits". In *Proc. 1st NASA/DoD Workshop Evolvable Hardware*, Pasadena CA, July 1999, pp. 12-17.

[50] H.R. Lewis, C.H. Papadimitriou. *Elements of the Theory of Computation.* Prentice-Hall, Englewood Cliffs, NJ, 1981.

[51] P. Liden, P. Dahlgren, R. Johansson, J. Karlsson. “On Latching Probability of Particle Induced Transients in Combinational Networks”. In *Proc. Intl. Symposium on Fault-Tolerant Computing (FTCS-24)*, 1994, pp.340-349.

[52] D. Mange. *Microprogrammed Systems: An introduction to Firmware Theory.* Chapman & Hall, London, 1992.

[53] D. Mange, S. Durand, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, C. Piguet. *A New Paradigm for Developing Digital Systems Based on a Multi-Cellular Organization.* Technical Report, Logic Systems Laboratory, EPFL, Lausanne, 1996.

[54] D. Mange, S. Durand, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, C. Piguet. ”A New Self-Reproducing Automaton Based on a Multi-Cellular Organization”. Technical Report No 95/114, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, April 1995.

[55] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, S. Durand. *Embryonics: A New Family of Coarse-Grained Field-Programmable Gate Arrays with Self-Repair and Self- Reproducing Properties.* Technical Report No 95/154, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, November 1995.

[56] D. Mange, D. Madon, A. Stauffer, G. Tempesti. “Von Neumann Revisited: A Turing Machine with Self-Repair and Self-Reproduction Properties”. *Robotics and Autonomous Systems*, Vol. 22, No. 1, 1997, pp. 35-58.

[57] D. Mange, P. Marchal, C. Piguet, E. Sanchez. *Circuit électronique organisé en réseau matriciel de cellules.* No CH 688 425, September 15, 1997.

[58] D. Mange, P. Marchal, C. Piguet, E. Sanchez. *Electronic System Organised as an Array of Cells.* US Patent No 5,508,636, April 16, 1996.

[59] D. Mange, P. Marchal, C. Piguet, E. Sanchez. *Electronic System Organised as an Array of Cells.* Swiss patent application (applicant: CSEM, Centre suisse d'électronique et de microtechnique S.A.), patented in 17 european countries (AT, BE, CH, DE, DK, ES, FR, GB, GR, IE, IT, LI, LU, MC, NL, PT, SE), 1994.

[60] D. Mange, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, C. Piguet. “Embryonics: A New Methodology for Designing Field Programmable Gate Arrays with Self-Repair and Self-Replicating Properties”. *IEEE Transactions on VLSI Systems*, 6 (3), 1998, pp. 387-399.

[61] D. Mange, M. Sipper. “Von Neumann’s Quintessential Message: Genotype + Ribotype = Phenotype”. In *Artificial Life*, vol.4, no.3, 1998, pp. 225-228.

[62] D. Mange, M. Sipper, P. Marchal. “Embryonic Electronics”. In *BioSystems* 51, 1999, pp. 145-152.

[63] D. Mange, M. Sipper, A. Stauffer, G. Tempesti. ”Toward Robust Integrated Circuits: The Embryonics Approach”. In *Proc. of the IEEE*, vol. 88, No. 4, April 2000, pp. 516-541.

[64] D. Mange, A. Stauffer. *Interpréteur du langage PICOPASCAL.* Logic Systems Laboratory, The Swiss Federal Institute of Technology (EPFL), Lausanne, 1998.

[65] D.Mange, A.Stauffer. "Introduction to Embryonics: Towards New Self-repairing and Self-reproducing Hardware Based on Biological-like Properties". In *Artificial Life and Virtual Reality*, John Wiley, 1994.

[66] D. Mange, A. Stauffer, G. Tempesti. *Self-Replicating and Self-Repairing FPPAs*. Technical Report No. 97/246, Logic Systems Laboratory, The Swiss Federal Institute of Technology, Lausanne, September 1997.

[67] D. Mange, M. Tomassini, eds. *Bio-Inspired Computing Machines: Towards Novel Computational Architectures*. Presses Polytechniques et Universitaires Romandes, Lausanne, Switzerland, 1998.

[68] P. Marchal. "Field Programmable Gate Arrays: State of the Art". *Communications of the ACM*, vol. 42, no. 4, 1999.

[69] P. Marchal. "John von Neumann: The Founding Father of Artificial Life." In *Artificial Life*, vol.4, no.3, 1998.

[70] P. Marchal, P. Nussbaum, C. Piguet, S. Durand, D. Mange, E. Sanchez, A. Stauffer, G. Tempesti. "Embryonics: The Birth of Synthetic Life". In E. Sanchez, M. Tomassini, eds., *Towards Evolvable Hardware*, LNCS, Springer Verlag, Berlin, 1996, pp. 166-197.

[71] P. Marchal, C. Piguet, D. Mange, A. Stauffer, S. Durand. "Embryological Development on Silicon". In R.A. Brooks, P. Maes, eds., *Artificial Life IV*, MIT Press, Cambridge, MA, pp. 365-370.

[72] P. Marchal, A. Stauffer. "Binary Decision Diagram Oriented FPGAs". In *Proc. FPGA'94, 2nd International ACM/SIGDA Workshop on Field-Programmable Gate Arrays*, Berkeley, CA, February 1994, pp. 1-10.

[73] D. Marston. *Memory System Reliability With ECC*. Intel Appl. Note, AP-73, Intel Corp., 1980.

[74] M. May. "Mother Nature's Menders. The Battle Against Aging". *Scientific American* Special Issue "The Quest to Beat Aging", June 2000.

[75] T.C. May, M.H. Woods. "Alpha-particle-induced soft errors in dynamic memories". *IEEE Trans. Electronic Devices*, vol. ED-26, 1979, pp. 2.

[76] J.M. Moreno, Y. Thoma, E. Sanchez, O. Torres, G. Tempesti. "Hardware Realization of a Bio-inspired POEtic tissue". In R. S. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. Lohn, and A. Stoica, eds. *Proceedings of the NASA/DoD Conference on Evolvable Hardware (EH'04)*, Los Alamitos, CA, 2004, pp. 237-244.

[77] R. Negrini, M. G. Sami, R. Stefanelli. *Fault Tolerance Through Reconguration in VLSI and WSI Arrays*. The MIT Press, Cambridge, MA, 1989.

[78] J. von Neumann. *The Computer and the Brain* (2nd edition). Physical Science, 2000.

[79] J. von Neumann. The Theory of Self-Reproducing Automata. A. W. Burks, ed. University of Illinois Press, Urbana, IL, 1966.

[80] J. von Neumann. *Collected Works* (6 volumes). Pergamon Press, 1961-63.

[81] J. von Neumann. "Probabilistic Logic and the Synthesis of Reliable Organisms from Unreliable Components". In C.E. Shannon, J. McCarthy (eds.) *Automata Studies*, Annals of Mathematical Studies 34, Princeton University Press, pp. 43-98, 1956.

[82] M.A. Nielsen, I.L. Chuang. *Quantum Computation and Quantum Information*. Cambridge University Press, 2000.

[83] T.J. O'Gorman, J.M. Ross, A.H. Taber, J.F. Ziegler, H.P. Muhlfeld, C.J. Montrose, H.W. Curtis, J.L. Walsh. "Field Testing for Cosmic Ray Soft

Errors in Semiconductor Memories". *IBM Journal of Research and Development*, Vol 40, No 1, January 1996, pp. 41–50.

[84] C. Ortega, A. Tyrrell. "Reliability Analysis in Self-Repairing Embryonic Systems". In *Proc. 1st NASA/DoD Workshop Evolvable Hardware*, Pasadena, CA, July 1999, pp. 120-128.

[85] C. Ortega, A. Tyrrell. "Self-Repairing Multicellular Hardware: A Reliability Analysis". In D. Floreano, J.-D. Nicoud, F. Mondada, eds, *Advances in Artificial Life*, Berlin, Germany: Springer-Verlag, vol. 1674, 1999, pp. 442-446.

[86] C. Ortega, A. Tyrrell. "MuxTree Revisited: Embryonics as a Reconfiguration Strategy in Fault-Tolerant Processor Arrays". In *Evolvable Systems: From Biology to Hardware*, M. Sipper, D. Mange, A. Perez-Uribe, eds, LNCS vol. 1478, Springer Verlag, Berlin, 1998, pp. 206-217.

[87] K. Parnell, N. Mehta. *Programmable Logic Design. Quick Start Hand Book*. Xilinx Inc, Jan. 2002.

[88] D.A. Patterson, J.L. Hennessy. *Computer Organization and Design: A Quantitative Approach*. 3rd Edition, Morgan Kaufmann, May 2002.

[89] A. Perez-Uribe. *Structure-Adaptable Digital Neural Networks*. PhD thesis 2052, Swiss Federal Institute of Technology-Lausanne, October, 1999.

[90] A. Perez-Uribe, E. Sanchez. "FPGA Implementation of an Adaptable-Size Neural Network". In *Proc. Intl. Conference on Artificial Neural Networks (ICANN96)*, Springer-Verlag, Heidelberg, 1996, pp. 383-388.

[91] M. Perkowski, A. Mishchenko, A. Chebotarev. "Evolvable Hardware or Learning Hardware? Induction of State Machines from Temporal Logic Constraints". Proc. 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, July 1999, pp. 129-138.

[92] J.Y. Perrier, M. Sipper, J. Zahnd. "Toward a Viable, Self-Reproducing Universal Computer". In *Physica 97D*, 1996, pp. 335-352.

[93] M. Pflanz, K. Walther, H.T. Vierhaus. "On-line Error Detection Techniques for Dependable Embedded Processors with High Complexity". In *Proc. 8th IEEE Intl. On-Line Testing Workshop (IOLTW'02)*, France, July 2002, pp. 69-73.

[94] J. Preskill. "Fault Tolerant Quantum Computation". In H.K. Lo, S. Popescu and T.P. Spiller, eds. *Introduction to Quantum Computation*, World Scientific Publishing Co., 1998.

[95] L. Prodan, M. Udrescu, M. Vladutiu. "Survivability of Embryonic Memories: Analysis and Design Principles". *Proceedings of the NASA/DoD Conference on Evolvable Hardware* (*EH'05*), IEEE Computer Society, Washington DC, 2005, pp. 280-289.

[96] Lucian Prodan, Mihai Udrescu, Mircea Vladutiu. "Multiple-Level Concatenated Coding in Embryonics: A Dependability Analysis". *GECCO* (*ACM-SIGEVO*), Washigton, DC, USA, June 25-29, 2005, pp. 941-948.

[97] L. Prodan, M. Udrescu, M. Vladutiu. "Reliability Assessment in Embryonics Inspired by Fault-Tolerant Quantum Computation". In *Proceedings 2nd ACM International Conference on Computing Frontiers (CF'05)*, Ischia, Italy, 2005, pp.323-333.

[98] L. Prodan, M. Udrescu, M. Vladutiu. "Self-Repairing Embryonic Memory Arrays". In R. S. Zebulum, D. Gwaltney, G. Hornby, D. Keymeulen, J. Lohn and A. Stoica, eds. *Proceedings of the NASA/DoD Conference on*

*Evolvable Hardware (EH'04)*, IEEE Computer Society, Seattle WA, 2004, pp. 130-137.

[99] L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Electronic Stem Cells". In R.K. Standish, M.A. Bedau, and H.A. Abbass, Eds., Artificial Life VIII, *Proceedings of the 8th International Conference on Artificial Life*, Bradford Book, The MIT Press, Cambridge MA, pp.101-105, 2003.

[100] L. Prodan, D. Mange, G. Tempesti. *The Embryonics Project: Specifications of the MUXTREE Field-Programmable Gate Array.* Technical Report No. IC/2002/03, School of Computer and Communication Sciences, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, Switzerland, January 2002.

[101] L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Artificial Cells Driven by Artificial DNA". In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, M. Yasunaga (eds.), Evolvable Systems: From Biology to Hardware, *Proc. 4th International Conference on Evolvable Systems (ICES2001)*, Tokyo, Japan, Lecture Notes in Computer Science vol. 2210, Springer-Verlag, Berlin, 2001, pp. 100-111.

[102] L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Biology Meets Electronics: The Path to a Bio-Inspired FPGA". In J. Miller, A. Thompson, P. Thomson, T.C. Fogarty (eds.), Evolvable Systems: From Biology to Hardware, *Proc. 3rd International Conference on Evolvable Systems (ICES2000)*, Edinburgh, Scotland, LNCS vol. 1801, Springer, Berlin, 2000, pp. 187-196.

[103] L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Artificial Cells Made of Artificial Molecules". In *Proc. Fourth International Conference on Technical Informatics (CONTI 2000)*, Universitatea Politehnica din Timisoara, Romania, 2000, pp. 99-104.

[104] T.R.N. Rao, E. Fujiwara. *Error-Control Coding for Computer Systems.* Prentice-Hall, 1989.

[105] J.A. Reggia, J.D. Lohn, H.H. Chou. "Self-Replicating Structures: Evolution, Emergence, and Computation". In *Artificial Life*, vol.4, no.3, 1998.

[106] R. B. Roberts, ed. *Microsomal Particles and Protein Synthesis: Papers Presented at the First Symposium of the Biophysical Society.* Washington, DC, Pergamon Press, 1958.

[107] E. Sanchez, D. Mange, M. Sipper, M. Tomassini, A. Perez-Uribe, A. Stauffer. "Phylogeny, Ontogeny, and Epigenesis: Three Sources of Biological Inspiration for Softening Hardware". In T. Higuchi, M. Iwata, W. Liu, eds., *Proc. 1st Int. Conference on Evolvable Systems: From Biology to Hardware (ICES96)*, LNCS vol. 1259, Springer-Verlag, Berlin, 1997, pp. 35-54.

[108] E. Sanchez, M. Sipper, J.-O. Haenni, J.-L. Beuchat, A. Stauffer, A. Perez-Uribe. "Static and Dynamic Configurable Systems". *IEEE Transactions on Computers*, vol. 48, no.6, June 1999, pp. 556-564.

[109] E. Sanchez, M. Tomassini (eds.). *Towards Evolvable Hardware.* LNCS, vol. 1062, Springer-Verlag, Heidelberg, 1996.

[110] N. Seifert, D. Moyer, N. Leland, R. Hokinson. "Historical Trend in Alpha-Particle Induced Soft Error Rates of the Alpha™ Microprocessor". In

*IEEE 39th Annual International Reliability Physics Symposium*, 2001, pp. 259–265.

[111] L. Sekanina. *Evolvable Components. From Theory to Hardware Implementations*. Springer-Verlag, 2004.

[112] C.E. Shannon. "Von Neumann's contributions to automata theory". In *Bulletin of the American Mathematical Society* 64, 1958, pp. 123-129.

[113] P. Shivakumar, M. Kistler, S.W. Keckler, D. Burger, L. Alvisi. "Modelling the Effect of Technology Trends on the Soft Error Rate of Combinational Logic". In Proc. Intl. Conference on Dependable Systems and Networks (DSN), June 2002.

[114] P. Shivakumar, M. Kistler, S. Keckler, D. Burger, L. Alvisi. *Modeling the Impact of Device and Pipeline Scaling on the Soft Error Rate of Processor Elements*. UT-Austin Computer Sciences Technical Report TR-02-19, April 2002.

[115] M. Sipper. "Fifty Years of Research on Self-Replication: An Overview". In *Artificial Life*, vol.4, no.3, 1998, pp. 237-258.

[116] Moshe Sipper. "If the Milieu is Reasonable: Lessons from Nature on Creating Life". *Journal of Transfigural Mathematics*, vol.3, no.1, 1997, pp. 7-22.

[117] M. Sipper, D. Mange, E. Sanchez. "Quo Vadis Evolvable Hardware?" *Communications of the ACM* 42 (4), 1999, pp. 50-56.

[118] M. Sipper, D. Mange. A. Stauffer. "Ontogenetic Hardware". In *BioSystems* 44 (3), 1997, pp. 193-207.

[119] M. Sipper, E. Sanchez, D. Mange, M. Tomassini, A. Perez-Uribe, A. Stauffer. "A Phylogenetic, Ontogenetic and Epigenetic View of Bio-Inspired Hardware Systems". In *IEEE Transactions on Evolutionary Computation*, Vol. 1, No. 1, April 1997.

[120] M. Sipper, G. Tempesti, D. Mange, E. Sanchez. Guest Editors' Introduction. "Von Neumann's Legacy": Special Issue on Self-Replication. In *Artificial Life*, vol.4, no.3, 1998.

[121] D.F. Smart, M.A. Shea. *Galactic Cosmic Radiation and Solar Energetic Particles*. Report No. ADA 167000, U.S. Air Force Geophysics Laboratory, 1985.

[122] E.H. Spafford. "Computer Viruses – A Form of Artificial Life?" In C.G. Langton, C. Taylor, J.D. Farmer, S. Rasmussen, eds., *Artificial Life II*, Addison-Wesley, Redwood City, CA, 1992, pp. 727-745.

[123] A. Stauffer. *Membrane Building and Binary Decision Machine Implementation*. Technical Report No. 247, Logic Systems Laboratory, The Swiss Federal Institute of Technology, Lausanne, September 1997.

[124] A. Stauffer, D. Mange, G. Tempesti, C. Teuscher. "A Self-Repairing and Self-Healing Electronic Watch: The BioWatch". In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, and M. Yasunaga, eds., Evolvable Systems: From Biology to Hardware. *Proceedings of the 4th International Conference on Evolvable Systems (ICES'2001)*, October 3-5, 2001, Tokyo. LNCS Vol. 2210, Springer-Verlag, Berlin, Heidelberg, 2001, pp. 112-127.

[125] A. Stauffer, D. Mange, G. Tempesti, C. Teuscher. "Biowatch: A Giant Electronic Bio-Inspired Watch". In Proc. 3rd NASA/DoD Workshop on Evolvable Hardware, July 12 - 14, 2001, Long Beach, California, pp. 185-192.

[126] C. Stroud, S. Konala, M. Abramovici. "Using ILA testing for BIST in FPGAs". In Proc. 2nd IEEE International On-Line Testing Workshop, Biarritz, July 1996.

[127] H.H.K. Tang. "Nuclear Physics of Cosmic Ray Interaction with Semiconductor Materials: Particle–Induced Soft Errors from a Physicist's Perspective". IBM Journal of Research and Development, Vol 40, No 1, January 1996, pp. 91–108.

[128] G. Tempesti. A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Processes. Ph.D. Thesis No. 1827, Logic Systems Laboratory, The Swiss Federal Institute of Technology, Lausanne, 1998.

[129] G. Tempesti. "A New Self-Reproducing Cellular Automaton Capable of Construction and Computation". In Proc. 3rd European Conference on Artificial Life, Granada, Spain, June 4-6, 1995, Lecture Notes in Artificial Intelligence, vol. 929, Springer Verlag, Berlin, 1995, pp. 555-563.

[130] G. Tempesti. "A New Self-Reproducing Cellular Automaton Capable of Construction and Computation". Technical Report No 95/116, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, June 1995.

[131] G. Tempesti, D. Mange, A. Stauffer. "A Robust Multiplexer-based FPGA Inspired by Biological Systems". Journal of Systems Architecture: Special Issue on Dependable Parallel Computer Systems, EUROMICRO, 43(10), 1997, pp. 719-733.

[132] G. Tempesti, D. Mange, A. Stauffer. "The Embryonics Project: A Machine Made of Artificial Cells". Rivista di Biologia-Biology Forum, Vol. 92, No 1, January-April 1999, pp. 143-188.

[133] G. Tempesti, D. Mange, A. Stauffer. "Self-Replicating and Self-Repairing Multicellular Automata". In Artificial Life, vol.4, no.3, 1998, pp. 259-282.

[134] G. Tempesti, D. Mange, A. Stauffer, C. Teuscher. "The BioWall: an Electronic Tissue for Prototyping Bio-Inspired Systems". In A. Stoica, J. Lohn, R. Katz, D. Keymeulen, and R. S. Zebulum, eds., *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'2002)*, IEEE Computer Society, Los Alamitos CA, pp. 221-230.

[135] G. Tempesti, C. Teuscher. "Biology Goes Digital: An array of 5,700 Spartan FPGAs brings the BioWall to "life"". *XCell Journal*, Fall 2003, pp.40-45.

[136] Y. Thoma, G. Tempesti, E. Sanchez, J.-M. Moreno Arostegui. "POEtic: An Electronic Tissue for Bio-Inspired Cellular Applications". *BioSystems*, 74 (1-3), 2004, pp. 191-200.

[137] J.A. Thomson, J. Itskovitz-Eldor, S.S. Shapiro, M.A. Waknitz, J.J. Swiergiel, V.S. Marshall, J.M. Jones. "Embryonic Stem Cell Lines Derived from Human Blastocysts". *Science 282*, Nov. 6, 1998, pp. 1145-1147.

[138] Y. Tosaka, S. Satoh, T. Itakura, H. Ehara, T. Ueda, G. Woffinden, S. Wender. "Measurement and Analysis of Neutron-Induced Soft Errors in Sub-Half-Micron Circuits". In *IEEE Transactions on Electron Devices*, Vol. 45, No. 7, July 1998.

[139] Y. Tosaka, S. Satoh, K. Suzuki, T. Sugii, H. Ehara, G.Woffinden, S.Wender. "Impact of cosmic ray neutron induced soft errors on advanced

submicron CMOS circuits". *Symposium on VLSI Technology Digest of Technical Papers*, 1996.

[140] S. Trimberger, ed. *Field-Programmable Gate Array Technology*. Kluwer Academic Publishers, Boston, 1994.

[141] A.J. Tylka, J.H.Adams Jr., P.R. Boberg, B. Brownstein, W.F. Dietrich, E.O. Flueckiger, M.A. Shea, D.F. Smart, E.C. Smith. "CREME96: A Revision of the Cosmic Ray Effects on Micro-Electronics Code". In *IEEE Trans. on Nuclear Science*, vol.44, no.6, December 1997, pp. 2150-2160.

[142] A.J. Tylka, W.F. Dietrich, P.R. Boberg. "Probability Distributions of High-Energy Solar-Heavy-Ion Fluxes from IMP-8: 1973-1996". In *IEEE Trans. on Nuclear Science*, vol.44, no.6, December 1997, pp. 2140-2149.

[143] A.J. Tylka, W.F. Dietrich, P.R. Boberg, E.C. Smith, J.H.Adams Jr. "Single Event Upsets Caused by Solar Energetic Heavy Ions". In *IEEE Trans. on Nuclear Science*, vol.43, no.6, December 1996, pp. 2758-2766.

[144] A. Tyrrell. "Computer Know Thy Self!? A Biological Way to Look at Fault-Tolerance". In *Proc. 2nd IEE/EuroMicro Workshop on Dependable Computing Systems*, Milan, September 1999.

[145] M. Udrescu, L. Prodan, M. Vladutiu. "Improving Quantum Circuit Dependability with Reconfigurable Quantum Gate Arrays". In *Proceedings 2nd ACM International Conference on Computing Frontiers (CF'05)*, Ischia, Italy, 2005, pp. 133-144.

[146] M. Udrescu, L. Prodan, M. Vladutiu. "Simulated Fault Injection in Quantum Circuits with the Bubble Bit Technique". In *Proceedings 7th International Conference on Adaptive and Natural Computing Algorithms (ICANNGA)*, Coimbra, Portugal, 2005, pp. 276-279.

[147] M. Udrescu, L. Prodan, M. Vladutiu. "The Bubble Bit Technique as Improvement of HDL-Based Quantum Circuits Simulation". In *Proceedings 38th IEEE Annual Simulation Symposium*, San Diego CA, USA, 2005, pp. 217-224.

[148] M. Udrescu, L. Prodan, M. Vladutiu. "Using HDLs for Describing Quantum Circuits: A Framework for Efficient Quantum Algorithm Simulation". In *Proceedings 1st ACM Conference on Computing Frontiers (CF'04)*, Ischia, Italy, 2004, pp. 96-110.

[149] M. Udrescu, L. Prodan, M. Vladutiu. "A New Perspective in Simulating Quantum Circuits". In *Proceedings GECCO*, Chicago IL, July 2003, pp. 283-290.

[150] C. Vaughan. *Stem cells' "Guardians" Found to Control Cell Specialization*. Stanford Report, October 18, 2000.

[151] J.D. Watson, N.H. Hopkins, J.W. Roberts, J. Argetsinger Steitz, A.M. Weiner. *Molecular Biology of the Gene*. Benjamin/Cummings, Menlo Park, CA, 4th edition, 1987.

[152] G.S.West, S.J. Wright, H.C. Euler. *Space and Planetary Environment Criteria Guidelines for Use in Space Vehicle Developments*. NASA-TM-78119, 1977.

[153] W. Wolf. *Computers as Components. Principles of Embedded Computing System Design*. Academic Press, 2001.

[154] S. Wolfram. *Cellular Automata and Complexity*. Addison-Wesley, Reading, MA, 1994.

[155] L. Wolpert. *The Triumph of the Embryo*. Oxford University Press, New York, 1991.

[156] R. C. Wyatt, P. J. McNulty, P. Toumbas, P. L. Rothwell, R. C. Filz. "Soft Errors Induced by Energetic Protons". In *IEEE Trans. Nuclear Science*, 26, 1979, pp. 4905.

[157] S. Xanthakis, S. Karapoulios, R. Pajot, A. Rozz. "Immune System and Fault Tolerant Computing". In J.M. Alliot, ed., *Artificial Evolution*, LNCS vol. 1063, Springer-Verlag, 1996, pp. 181-197.

[158] X. Yao. "Following the Path of Evolvable Hardware". *Communications of the ACM*, April 1999, 42(4), pp. 47-49.

[159] J. M Yarbrough. *Digital Logic. Applications and Design*. West Publishing Company, 1997.

[160] C. Zalka. Threshold Estimate for Fault Tolerant Quantum Computation. In *arXiv:quant-ph/9612028*, v2, July 28, 1997.

[161] J.F. Ziegler. "Terrestrial Cosmic Ray Intensities". *IBM Journal of Research and Development*, Vol 42, No 1, January 1998, pp. 117–139.

[162] J.F. Ziegler. "Terrestrial Cosmic Rays". *IBM Journal of Research and Development*, vol.40, no.1, January 1996, pp. 19-39.

[163] J.F. Ziegler, H.W. Curtis, H.P. Muhlfeld, C.J. Montrose, B. Chin, M. Nicewicz, C.A. Russell, W.Y. Wang, L.B. Freeman, P. Hosier, L.E. LaFave, J.L. Walsh, J.M. Orro, G.J. Unger, J.M. Ross, T.J. O'Gorman, B. Messina, T.D. Sullivan, A.J. Sykes, H. Yourke, T.A. Enger, V. Tolat, T.S. Scott, A.H. Taber, R.J. Sussman, W.A. Klein, C.W. Wahaus. "IBM Experiments in Soft Fails in Computer Electronics (1978-1994)". *IBM Journal of Research and Development*, Vol 40, No 1, January 1996, pp. 3–18.

[164] J. F. Ziegler, W. A. Lanford. "The Effect of Cosmic Rays on Computer Memories". *Science*, vol.206, 1979, pp. 776.

[165] J.F. Ziegler, H.P. Muhlfeld, C.J. Montrose, H.W. Curtis, T.J. O'Gorman, J.M. Ross. "Accelerated Testing for Cosmic Soft Error Rate". *IBM Journal of Research and Development*, Vol 40, No 1, January 1996, pp. 51–72.

[166] J.F. Ziegler, M.E. Nelson, J.D. Shell, R.J. Peterson, C.J. Gelderloos, H.P. Muhlfeld, C.J. Montrose. "Cosmic Ray Soft Error Rates of 16-Mb DRAM Memory Chips". *IEEE Journal of Solid-State Circuits*, vol.33, no.2, February 1998, pp. 246-252.

[167] ***. "Opportunities and Challenges: A Focus on Future Stem Cell Applications". In *Stem Cells: Scientific Progress and Future Research Directions*, National Institutes of Health, Department of Health and Human Services, June 2001.

[168] ***. *The BioCube*. http://lslwww.epfl.ch

[169] ***. ITRS – International Technology Roadmap for Semiconductors, Emerging Research Devices, 2004, http://www.itrs.net/Common/2004 Update/2004_05_ERD.pdf

[170] ***. *Stem Cells: A Primer*. National Institutes of Health, May 2000.

[171] ***. *Xilinx 6200 Preliminary Data Sheet*. Xilinx, San Jose, CA, 1996.

[172] ***. http://www2.slac.stanford.edu/vvc/cosmicrays/cratmos.html

[173] ***. http://www.sel.noaa.gov/

[174] ***. http://www.ngdc.noaa.gov/stp/GOES/scatter.pdf

[175] ***. http://presto.stsci.edu/vision/vision_se/ngstOpsConceptWG/ Communications Study/SolarProtonFlux/SolarProtonFlux.htm

# LIST OF PUBLICATIONS

## Conference Proceedings

L. Prodan, M. Udrescu, M. Vladutiu. "Survivability of Embryonic Memories: Analysis and Design Principles". *Proceedings of the NASA/DoD Conference on Evolvable Hardware* (*EH'05*), IEEE Computer Society, Washington DC, 2005, pp. 280-289.

L. Prodan, M. Udrescu, M. Vladutiu. "Multiple-Level Concatenated Coding in Embryonics: A Dependability Analysis". *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO),* ACM-SIGEVO, Washigton DC, 2005, pp. 941-948.

L. Prodan, M. Udrescu, M. Vladutiu. "Reliability Assessment in Embryonics Inspired by Fault-Tolerant Quantum Computation". *Proceedings of the 2nd International Conference on Computing Frontiers* (*CF'05*), ACM-SIGARCH, Ischia, Italy, 2005, pp.323-333.

M. Udrescu, L. Prodan, M. Vladutiu. "Improving Quantum Circuit Dependability with Reconfigurable Quantum Gate Arrays". *Proceedings of the 2nd International Conference on Computing Frontiers* (*CF'05*), ACM-SIGARCH Ischia, Italy, 2005, pp. 133-144.

M. Udrescu, L. Prodan, M. Vladutiu. "Simulated Fault Injection in Quantum Circuits with the Bubble Bit Technique". In Bernardete Ribeiro et al., eds., *Proceedings of the International Conference "Adaptive and Natural Computing Algorithms"*, Coimbra, Portugal, 2005, Springer-Verlag, Berlin, pp. 276-279.

M. Udrescu, L. Prodan, M. Vladutiu. "The Bubble Bit Technique as Improvement of HDL-Based Quantum Circuits Simulation". *Proceedings of the 38th Annual Simulation Symposium* (*ANSS*), IEEE Computer Society, San Diego CA, 2005, pp. 217-224.

L. Prodan, M. Udrescu, M. Vladutiu. "Self-Repairing Embryonic Memory Arrays". *Proceedings of the NASA/DoD Conference on Evolvable Hardware* (*EH'04*), IEEE Computer Society, Seattle WA, 2004, pp. 130-137.

M. Udrescu, L. Prodan, M. Vladutiu. "Using HDLs for Describing Quantum Circuits: A Framework for Efficient Quantum Algorithm Simulation". *Proceedings of the 1st Conference on Computing Frontiers* (*CF'04*), ACM-SIGARCH, Ischia, Italy, 2004, pp. 96-110.

M. Udrescu, L. Prodan, M. Vladutiu. "A New Perspective in Simulating Quantum Circuits". *Proceedings of the Genetic and Evolutionary Computation Conference (GECCO),* 2003, LBP pp. 283-290.

L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Electronic Stem Cells". In R.K. Standish, M.A. Bedau, and H.A. Abbass, Eds., Artificial Life VIII,

*Proceedings of the 8th International Conference on Artificial Life*, Bradford Book, The MIT Press, Cambridge MA, 2003, pp.101-105.

L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Artificial Cells Driven by Artificial DNA". In Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, M. Yasunaga (eds.), *Evolvable Systems: From Biology to Hardware, Proceedings of the 4th International Conference on Evolvable Systems* (*ICES2001*), Tokyo, Japan, LNCS 2210, Springer-Verlag, Berlin, 2001, pp. 100-111.

L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Biology Meets Electronics: The Path to a Bio-Inspired FPGA". In J. Miller, A. Thompson, P. Thomson, T.C. Fogarty (eds.), *Evolvable Systems: From Biology to Hardware, Proceedings of the 3rd International Conference on Evolvable Systems* (*ICES2000*), Edinburgh, Scotland, LNCS 1801, Springer-Verlag, Berlin, 2000, pp. 187-196.

L. Prodan, G. Tempesti, D. Mange, A. Stauffer. "Embryonics: Artificial Cells Made of Artificial Molecules". *Proceedings of the 4th International Conference on Technical Informatics* (*CONTI 2000*), "Politehnica" University of Timisoara, Romania, 2000, pp. 99-104.

## Technical Reports

L. Prodan, D. Mange, G. Tempesti. *The Embryonics Project: Specifications of the Muxtree Field Programmable Gate Array*. Technical Report 200203, School of Computer and Communication Sciences, Logic Systems Laboratory, Swiss Federal Institute of Technology (EPFL), Lausanne, January 2002.