

Biological Development model for the Design of Robust Digital System

Heng Liu

Doctor of Philosophy

Intelligent Systems Research Group

Electronics Department

University of York

September 2007

Abstract

This thesis presents a biologically-inspired developmental model for the design of digital circuits. Circuits have been evolved that exhibit the ability to self-repair and correct transient faults to recover correct functionality. The method devised gives no explicit coordinate information to the evolved cell circuits.

The method presented has been implemented fully in electronic hardware. This allowed developmental circuits to be evolved considerably more quickly than in software simulation.

The methods presented have been applied to produce a self-repairing two bit multiplier and an autonomous robot controller circuit. Results are presented that shows that after introduction of faults, both circuits can autonomously recover correct functionality.

Keywords

Fault-tolerance, Evolvable hardware, FPGA, Development principle, Multi-cellular organism, Evolutionary algorithm, Cartesian Genetic Programming, French Flag, Multiplier, Digital circuit, Autonomous Robot Controller

Acknowledgement

No matter how much to write, it is just impossible to express my deep and genuine appreciation to every person who supported and contributed to my research in terms of financial, technical and mental aspects. I will try my best to acknowledge those without whom it would have been less productive or even impossible.

Without the unconditional financial support from my parents, the study of my Britain degree would have been absolutely unfeasible at all. Furthermore, they have always had intuitive ideas, constructive suggestions and supportive encouragement; especially during I am in the most arduous situation of my research. I am proud of being born in this family and being your son.

This research was made possible under the excellent supervision of Prof. Andy Tyrrell and Dr. Julian Miller. I would like to thank them for their continuous support and guidance of the project as well as their suggestions and direction. In addition, I highly appreciate the efforts they exerted to find financial support for my further research.

To my wife, Mrs. Fei Xu, who is ready to give me unconditional backup and encourage me whenever I encounter problems. You know, it is hard for both of us in the year I am abroad. However we together make it! We love each other no less than, if not more than, one year ago. I am proud of being your companions. Thank you for all your valuable support and the peace of mind feeling you grant me.

My lab mates, Andy, Marcus, Dapeng Xu, Chao Wu, Rachael and Simon, also deserve my credit. Thanks for all your assistance and suggestions, as well as the laughs and times shared with me.

To my dear friends with whom I have had plenty of un-forgettable wonderful time, it has been a real fortune having you such good friends to communicate with, express my feelings and discuss general topics.

Contents

Abstract.....	i
Keywords	i
Acknowledgement	ii
Figure Index.....	ix
Table Index	xii
Chapter 1 Introduction.....	1
1.1 Research Motivation and Objective	3
1.2 Hypothesis.....	4
1.3 Achievements.....	4
1.4 Structure of the Thesis	5
Chapter 2 Evolutionary Design of Electronic Systems.....	7
2.1 Introduction	7
2.2 Principles of Evolutionary Design	9
2.3 Evolutionary Algorithms.....	13
2.3.1 <i>Common Features</i>	13
2.3.2 <i>Overview of EAs</i>	16
2.3.3 <i>Comparison of different EAs</i>	18
2.4 Evolvable Hardware.....	19
2.4.1 <i>Classification of Evolvable Hardware</i>	19
2.4.2 <i>Extrinsic Evolvable Hardware</i>	20
2.4.3 <i>Intrinsic Evolvable Hardware</i>	22
2.5 Limitation of EHW	24

Contents

2.6	Summary	24
Chapter 3 Development Principles and Development Systems		26
3.1	Biological Development Principles.....	26
3.1.1	<i>Basic Concepts</i>	26
3.1.1.1	<i>Chromosomes and Genes</i>	27
3.1.1.2	<i>Genotype and Phenotype</i>	27
3.1.2	<i>Biological Living Organism Cells</i>	28
3.1.3	<i>Stages of Development</i>	30
3.1.4	<i>Characteristics of Biological Development</i>	31
3.2	Mathematical Abstractions of Development.....	32
3.2.1	<i>Gene Regulatory Network Abstractions</i>	32
3.2.2	<i>L-Systems</i>	33
3.3	Developmental Systems	35
3.3.1	<i>Gene Regulatory Networks</i>	35
3.3.2	<i>L-Systems and other Rewriting Rule Based Systems</i>	37
3.3.3	<i>Cell Programs</i>	39
3.3.4	<i>Other Systems</i>	40
3.4	Summary	43
Chapter 4 Fault-tolerant Techniques		44
4.1	Fundamental Concepts and Definitions	44
4.1.1	<i>Classification of System Faults</i>	46
4.1.2	<i>Phases of Fault Tolerance</i>	48
4.1.2.1	<i>Error detection</i>	48
4.1.2.2	<i>Damage confinement and assessment</i>	49
4.1.2.3	<i>Error recovery</i>	49
4.1.2.4	<i>Fault treatment and continued system service</i>	50

Contents

4.2	History of Fault Tolerance Systems	50
4.3	Conventional Fault-tolerant Design	52
4.3.1	<i>Hardware Redundancy</i>	52
4.3.2	<i>Information Redundancy</i>	53
4.3.3	<i>Software Redundancy</i>	54
4.3.4	<i>Time Redundancy</i>	54
4.4	Conventional Transient Fault-tolerant Techniques.....	54
4.5	Bio-inspired Fault-tolerant Techniques	55
4.5.1	<i>Phylogeny</i>	56
4.5.2	<i>Ontogeny</i>	56
4.5.3	<i>Epigenesis</i>	57
4.6	Summary	57
Chapter 5 Development Cellular Model for Digital System		59
5.1	Transforming of Biological Principles to Digital Systems	59
5.2	Design Consideration	61
5.3	Digital Cell Structure and Inter-Cell Connections	63
5.4	Chemical Diffusion	67
5.5	Digital Organism Growth.....	68
5.6	Boundary Condition	70
5.7	Cartesian GP and Genotype Representation	70
5.8	Summary	72
Chapter 6 Software Simulation of a Pattern Problem.....		73
6.1	A Pattern Formation Problem: French Flag	73
6.2	Multi-Breed Evolution Algorithm	74
6.3	EO Evolutionary Computation Framework	78
6.4	Experiment	79

Contents

6.5	Results	81
6.6	Analysis	82
6.6.1	<i>Comparison with previous research</i>	83
6.6.2	<i>Robustness Analysis</i>	84
6.6.3	<i>Inputs and Molecules usages analysis</i>	87
6.7	Summary	89
Chapter 7 Implementation of Digital Logic System		91
7.1	Digital Organism Structure	91
7.2	Incorporating Execution Unit	91
7.3	Software Simulation	93
7.3.1	<i>Evolution Strategy</i>	93
7.3.2	<i>Parameters</i>	94
7.3.3	<i>Outcome of Simulation</i>	95
7.4	Hardware Implementation of the Digital Organism	96
7.4.1	<i>Top Level Structure</i>	96
7.4.2	<i>Sub-Circuits Evolved via Software Evolution</i>	97
7.5	Hardware Simulation and Verification	97
7.6	Summary	101
Chapter 8 IEWH Implementation of 2-bit multiplier		103
8.1	Design of the Intrinsic Evolvable Hardware	103
8.1.1	<i>Evolvable Molecule: the Basic Element in the EHW</i>	104
8.1.2	<i>Evolutionary Algorithm</i>	105
8.1.3	<i>Overview of the EHW</i>	106
8.1.4	<i>Top-level Hardware Modules in the IEHW</i>	106
8.1.5	<i>Execution Phase of the IEHW</i>	109
8.2	Hardware Platform	110

Contents

8.3	Develop Environment	110
8.4	Experiment and Results	111
8.4.1	<i>IEHW Parameters</i>	111
8.4.2	<i>Synthesis Report</i>	112
8.4.3	<i>Results of Experiment</i>	113
8.5	Summary	113
Chapter 9 Evolution of Sequential Digital System.....		115
9.1	Introduction	115
9.2	Adaptive Behaviour	116
9.3	Robust Random Access Memory	118
9.3.1	<i>1-Bit Primitive RAM Implementation</i>	119
9.3.1.1	<i>Input and Output Configuration</i>	119
9.3.1.2	<i>Experiments</i>	120
9.3.2	<i>Towards Larger Robust RAM</i>	121
9.3.2.1	<i>Modification to the Model</i>	121
9.3.2.2	<i>Memory Unit Interface</i>	122
9.3.2.3	<i>Experiment Result</i>	122
9.4	Discussion	123
9.5	Summary	124
Chapter 10 Autonomous Robot Controller		125
10.1	Introduction	125
10.1.1	<i>Robot and Application</i>	125
10.1.2	<i>Autonomous Robot Controller</i>	126
10.2	Robot and its Environment Setup	126
10.2.1	<i>Kiki Robot</i>	126
10.2.2	<i>Robotic Task and Environment</i>	127

Contents

10.3	Webots Simulation	128
10.3.1	<i>Introduction to Webots</i>	128
10.3.2	<i>Evolutionary Framework Deployed in Webots</i>	129
10.3.2.1	<i>Robot Controller</i>	129
10.3.2.2	<i>Supervisor Controller</i>	129
10.3.3	<i>Experiments and Results</i>	130
10.4	Discussion	133
10.5	Summary	135
Chapter 11 Conclusions		137
11.1	Summary and Achievement	137
11.2	Further Research Areas	139
Reference		141
Appendix		158
Appendix A Schematics of Circuits for multiplier		158
Appendix B RC1000 Board from Celoxica		160
Appendix C XCV1000 FPGA from Xilinx		162
Appendix D World setup in Webot		165
Appendix E Introduction to EO		166

Figure Index

Figure 2-1 Flow Chart of Evolutionary Design	9
Figure 2-2 Conventional Circuit Design Flow	10
Figure 2-3 Simplified PLD Structure	11
Figure 2-4 Typical Configurable Logic Block Architecture	12
Figure 2-5 Flow chart of common EA	14
Figure 2-6 Mutation in Binary Encoding EA	15
Figure 2-7 Crossover in Binary Encoding EA	15
Figure 2-8 GP Genotype Demonstration	17
Figure 2-9 Circuit Representation in Miller's work	21
Figure 2-10 Architecture of VRC [150]	22
Figure 3-1 Overview Biological Cell Structure	29
Figure 3-2 Differentiation of tissues	31
Figure 3-3 'Weeds' generated from a L-system in three dimensions [109]	34
Figure 3-4 Illustration of Cell Program Structure in Development CGP	40
Figure 4-1 Computer system dependability tree designed by Baron et al. [152]	45
Figure 4-2 Faults classification Categories [152]	46
Figure 5-1 Digital Organism Structure	63
Figure 5-2 Digital Cell Structure	64
Figure 5-3 Control Unit Structure	65
Figure 5-4 NSCG Structure	66
Figure 5-5 Execution Unit Structure	66
Figure 5-6 Flow chart of the digital organism growth procedure	69
Figure 5-7 A sample digital circuit	71
Figure 6-1 Perfect Cell State Pattern for French Flag problem	74

Figure Index

Figure 6-2 Flow Diagram of the Evolution Algorithm Proposed	75
Figure 6-3 Growth of one of the best 6x6 individuals	81
Figure 6-4 Recovery Process of the 6x6 robust individual	82
Figure 6-5 Best solution for FF formation	83
Figure 6-6 Recovery of the best FF solution.....	84
Figure 6-7 Growth of one of the robust 9x9 individuals.....	85
Figure 6-8 Recovery process from a random erasing of states/chemicals of the 9x9 individual	85
Figure 6-9 Recovery from a single cell of the 9x9 French flag	86
Figure 6-10 Input usage of the two Genes encoding a French flag	88
Figure 6-11 Chemical signal usage details	88
Figure 6-12 Normalized molecule usage in the best solutions	89
Figure 7-1 3x3 Cells Digital Organism.....	92
Figure 7-2 Distribution of states	95
Figure 7-3 Digital Organism External Interface	96
Figure 7-4 Digital Cell External Interface	97
Figure 7-5 Overview of the Simulation Waveform	98
Figure 7-6 Developmental Growth Procedure	98
Figure 7-7 Injection of the first set of faults and the recovery procedure.....	99
Figure 7-8 Injection of the second set of faults and the recovery procedure	100
Figure 8-1 Molecule Interface.....	104
Figure 8-2 The Architecture of Molecule	105
Figure 8-3 Top-level Overview of the Intrinsic Evolvable Hardware Platform	107
Figure 8-4 The distribution of the generations of the 100 experimental runs.....	114
Figure 9-1 Adaptive French flag grows to maturity	117
Figure 9-2 Adaptive French flag recovers from a set of transient faults	117

Figure Index

Figure 9-3 The flag transforms to “inverted” flag to adapt to the environmental signal	117
Figure 9-4 Inverted flag recovers from transient faults	118
Figure 9-5 Re-grow to French flag after resetting environmental signals	118
Figure 9-6 1-Bit RAM Growth	120
Figure 9-7 1-Bit RAM Recovery from faults with value 0 saved (default)	120
Figure 9-8 1-Bit RAM Stores 1 as its new value	121
Figure 9-9 Saving 0 to 1-Bit RAM when 1 is stored	121
Figure 10-1 Kiki robot	126
Figure 10-2 Distance measurements of the Kiki sensors.	127
Figure 10-3 Simple environment	128
Figure 10-4 Maze environment.....	128
Figure 10-5 3x3 Cells Digital Organism for robot controller	130
Figure 10-6 Kiki robot	130
Figure 10-7 Trajectory of Kiki for the simple world	131
Figure 10-8 Trajectory of Kiki for the maze world.....	131
Figure 10-9 Trajectory for the simple world with different starting point.....	132
Figure 10-10 Trajectory for the maze world with different starting point.....	132
Figure 10-11 Trajectory for the simple world with faults.....	132
Figure 10-12 Trajectory for the maze world with faults	132
Figure 10-13 A case where the simple controller fails	133
Figure 10-14 New maze world 1.....	133
Figure 10-15 New maze world 2.....	133
Figure 10-16 The maze controller fails when the wall is not detected	134

Table Index

Table 2-1 Differences between the four types of EA.....	18
Table 3-1 Biological Cell Components and their functions.....	28
Table 3-2 Definition of Lindenmayer's L-system for algae growth.....	33
Table 3-3 The Production of Lindenmayer's L-system.....	34
Table 4-1 some well known Fault Types and their classification.....	47
Table 4-2 Ratios of transient errors to permanent failures [66]	48
Table 5-1 A 3x3 Digital Organism.....	62
Table 5-2 Comparison of Biological Cell with Digital Cell	67
Table 6-1 Growth speed comparison of this proposed model (left) with Dr. Miller's model (right).....	84
Table 7-1 Available functions for Molecules.....	94
Table 7-2 Overview of states occurrence in first stage experiment.....	95
Table 7-3 Chosen Cell State Pattern	95
Table 7-4 Resource Consumption of the Digital Organism FPGA VHDL implementation.....	101
Table 9-1 General RAM Memory Interface.....	119
Table 9-2 2-bit and 3-bit RAM Unit Experiment Results.....	123
Table 10-1 Dimensions of Kiki Robot.....	127
Table 10-2 Wheel speed decoding schema	129

Chapter 1

Introduction

Electronic systems are so used to by modern people in our everyday life that little attention is paid to them. However, they are utilized in almost every aspects of our society: from normal telephone to high-definition television, from smart mobile phones to most advanced cars, electronics systems coordinate all the components to work for the purpose.

Electronics systems do not only improve the convenience, efficiency and effectiveness of our daily life, they also help us achieve “impossible missions”: they allow us to dive into hazards deep sea, to explore on the surface of Mars and to make a long journey to the deep outer space for extraterrestrial intelligent cultures.

With the development of electronic technology, more complex and larger systems are designed and this tendency shows no sign of decelerating.

However, in spite of the achievement of designing complex systems, the resulting systems may not be as efficient as expected in regards to general robustness to system faults and adaptiveness to unpredictable changes in operating environments. Several factors contribute to these limitations. One of the most significant points is that all these systems are designed explicitly to a preset environmental condition. If either the specification for the system is difficult or even impossible to define, or the entire combination of environment conditions is unpredictable, the conventional design methodology may be more prone to lead to defective behaviour. The failure of Beagle 2 project [78] was led to by the very issue: the estimation of the atmosphere in Mars was not accurate; the surface characteristics in the landing site were little known; all landing conditions can not be predicted, and Beagle 2 was stuck in one of the un-foreseen scenarios. [79]

One the other hand, the most complicated systems existing on this planet, humans, are *created* by nature: from the development of a single cell to an adult, the blueprint to build a human has enough error margins to cope with

unpredictable environments correctly. In addition, human can adapt to the changing environment efficiently.

As a matter of fact, most living multi-cellular biological organisms exhibit these intrinsic characteristics electronic engineers earnestly long for. Amphibians can regrow lost limbs [82]; Mice can repair their punched ear very quickly and without scars [80]; Human can regenerate their liver throughout their whole life time [81].

Multi-cellular living organisms achieved these traits through millions of years of evolution by means of cells which have identical genotypes. In most cases, all of them come from a single special cell (zygote): this process is called development. The entire process of development is controlled by the interaction of cells rather than by a centralized process.

Evolution is the approach of nature to *design* complex living creatures. Evolution refers to the process in which beneficial traits are inherited from generation to generation. Varieties of traits are generated by genetic drift, such as mutation, genetic recombination, sex and gene flow. Among these novel traits, only the ones increase the survival chance of the individual will be passed on to next generation. This is called the natural selection. In other words, it is determined by natural selection that whether a particular characteristic is beneficial or not [83][84][85]. Although new species occurrence normally takes long stretches of time, it is suggested that evolution rate can be seven orders of magnitude greater than rates implied by the paleontological records [86].

One approach to tackle the issues raised previous about designing electronics systems is to draw inspirations from the biological systems to design novel frameworks.

The most widely used methodology in bio-inspired research areas is evolutionary algorithm, which mimics the biological evolution. Evolutionary algorithm will be discussed in further details in Chapter 2.

In this chapter, basic background and motivation of this project is overviewed first; following that, the hypothesis is presented; achievements of this research is highlighted in section 1.3; finally the structure of this thesis is reviewed.

1.1 Research Motivation and Objective

In order to try to discover novel approaches to overcome the shortcomings of existing design methodology, the project draw inspiration from development principle to design novel frameworks.

Although development principle¹ has been applied to various electronics models to obtain emergent properties, most of them tend to rely on the explicit information about the positions of cells in the cell array: the position of a cell may be fed to each cell or configured within the genotype. In addition, most of work in this field only deals with applications with no real world functionalities, such as pattern formations.

In this research, we will provide a developmental model which does not depend on any knowledge about cell coordination. Evolution is deployed to generate the desired design for a particular application, thus we also deliberate the requirements for evolving with such a developmental model. We will also describe how to deploy this model so that it can be applied to evolve functionalities, such as multipliers and robot controllers

In traditional evolution encoding schema, the solution of a specific system is encoded entirely in the chromosome so that it can be mapped directly to the result. Despite its simplicity and wide acceptance, it is not particular suitable for multi-cellular model due to the fact that the data the chromosome needs to contain is proportional to the number of cells (components) in a system.

In addition, the conventional evolution algorithms does not provide any mechanisms to honor the complicated interaction between sub-components in the system, as the case in the biological development which is facilitated the gene regulation network: the dynamics of the development model can not be exploited which could lead to fault-tolerance features or dynamic modification of the organism, as they make use of static mapping between genotype and phenotype.

¹ Biological development principle will be reviewed in Chapter 3.

Therefore a novel evolutionary algorithm tailored for hardware implementation and the grow mechanism are developed. (discuss on multiple chromosomes in conclusion/future section)

To summarize, the objective of our work is to propose a general FPGA-based evolutionary developmental model, which has the emergent properties of adaptation and fault-recovery and can be deployed to incorporate various functionalities.

1.2 Hypothesis

Our hypothesis for this research is: a system model derived from developmental principles can lead to emergent properties of adaptation and fault-recovery which can achieve levels of reliability for hardware systems, and functionality can be incorporated to development systems to perform desired functions.

1.3 Achievements

This section provides an overview about the achievements of this research, including references to the relevant chapters with more details.

We propose a developmental model² inspired on its biological counterpart³, in which an indirect genotype and phenotype mapping in the form of a simplified development process is deployed. This model is designed with hardware in mind so that it can be accomplished and executes efficiently in hardware implementation. In order to demonstrate some of the emergent features of this model, we first present the evolution and development of a French Flag assumed by the model (Chapter 6), which is a pattern formation problem with a French Flag pattern as the goal. Notably extremely high capacity of transient fault tolerance is observed.

After introducing the characteristics of the model, we then let it to carry some functions: the model is employed to implement a simple digital logic circuit (2 bit multiplier), and realized it in an off-shelf FPGA. We show that this model is

² The model is discussed in detail in Chapter 5.

³ In section 3.1, the biological principle of development is review.

indeed hardware friendly and the high performance of hardware implementation compared to software simulation (Chapter 7 and Chapter 8).

As the model equipped with memory units, sequential circuits are investigated in Chapter 9, where the model is enhanced and fine tuned. We demonstrate that the model can be used not only for combinational digital systems, but also for circuits with memory.

Finally in Chapter 10 we evolve autonomous robot controllers to control the navigation of a robot, which is a more challenging problem than multiplier. We demonstrate the capacity of the model in solving non-trivial applications with the bonus of implicit fault-tolerance and self-repair.

1.4 Structure of the Thesis

This chapter presents the necessary introduction information for this work. In the following chapters, we will first review the evolution methodology in electronic system design in Chapter 2. We introduce the definition of evolutionary algorithms and evolvable hardware. We discuss the latter one in further details with digital and analogue system evolution and the clarification of evolvable hardware.

Since this research draws the inspiration from biological development principles, we brief the biological background in Chapter 3. Some existing computational abstractions of the principles are examined as well. At the last of this chapter, we present some development models proposed to work with electronics systems and we also review their applications.

Next we introduce the definition of fault-tolerant techniques in Chapter 4. We begin with basics concepts and definitions. Then the evolution of fault tolerance systems is described and conventional techniques are examined. We also present fault tolerant techniques inspired by different principles from biology.

In Chapter 5, we present the developmental cellular model proposed in this work. We show the process of the transforming from biological principles to digital circuits and systems. The internal structures of the model and the chemical diffusion mechanism are described first. The growth strategy deployed and the genotype representation of the digital organism are the highlighted. One

of the novel contributions of this thesis is to introduce an execution unit which brings functionality to developmental systems.

We reveal the built-in fault tolerant features of the simplified developmental model in the French Flag software simulation in Chapter 6. An evolutionary algorithm employed in this research is first presented which specifically fine tuned for hardware implementation. Then we demonstrate the setup of the experiment and the findings about the recovery characteristic of this model in the ending part of this chapter.

After incorporating all the components in the model, we present the implementation of a simple digital logic circuits assumes by this framework in Chapter 7. Details and results about both software simulation and intrinsic hardware implementation on FPGA are described.

In Chapter 9, the model faces a new challenge to deal with a sequential system, which is the fundamental infrastructure for learning. We first introduce adaptive behaviours and attempt to adapt the model for adaptive traits. Then we move on to evolve memory with more capacities.

A more complex application is evolved utilizing the model in Chapter 10, autonomous robot controller. The definition and background about robot controller are reviewed and then the configuration of the experiments is presented and results are discussed.

Finally, in Chapter 11 we conclude this thesis with emphasizing on contributions of this project, and make suggestions about further potential research directions.

Chapter 2

Evolutionary Design of Electronic Systems

As the basis of the evolution theory in biology, natural selection was first set out by Charles Darwin and Alfred Russel Wallace [133], the idea of which is that it is more likely for individuals who possess inheritable advantages to survive and more importantly to replicate. Inheritable advantages refer to capacities of tackling challenges posed by their own biological body and external ecological and physical environment. In modern evolution theory in biology, called the modern evolutionary synthesis, it is known that these advantages are encoded in the form of genes in DNA and these genes can be passed on from generation to generation. [87]

Some search and optimization techniques are inspired by natural evolution theories. These techniques are normally called evolutionary algorithms which can be deployed to design electronics systems. This chapter gives an overview of the evolutionary design approaches employing evolutionary algorithms. It presents the principles of evolution of electronics systems and introduces the evolutionary algorithms. Finally, this chapter highlights the discipline of evolvable hardware and discusses its applications in both digital and analogue systems.

2.1 Introduction

Inspired by the design approach of nature, evolution, similar methodologies were proposed to create artificial evolution for computational and electronics applications to explore novel solutions or optimize existing implementation. A common trait among these approaches is the utilization of evolutionary algorithm [23] [39], which refers to a generic search and optimization paradigm inspired by biological evolution in the form of simulating natural evolution over populations of candidate solutions. Specifically, the employment of evolutionary algorithms to design hardware electronics systems, such as circuits and controller structures, is called Evolvable Hardware (EHW).

The earliest citation of the EHW idea may come from Turing in 1940s. He proposed a network of unorganized randomly inter-connected NAND logic gates [88]. This primitive machine may be the first “neural network” system. He also suggested that it would be possible to make use of evolutionary search to train the network. However he did not actually investigate it further.

The concept of EHW was first conceived by H. De Garis in 1992 to accelerate trainings of neural network. The common definition of EHW was then introduced by Higuchi in 1993 after hearing H. De Garis’s talk on the EHW idea [89]. Higuchi accurately foresaw EHW as an emerging new branch of Artificial Intelligence and “the basis for a radically new approach to electronic and computer design” [90].

From the authors’ point of view, Evolvable hardware is the deployment of evolution design approach, specifically evolutionary algorithms, to meet all the requirements a predefined objective purpose. The key points in this definition are that the targeted application has to be implemented in hardware finally and evolutionary algorithms are utilized to obtain the desired functionality, the internal structures of the system and the inter-connection of the building blocks.

The final hardware can be assembled from discrete components, such as logical gates, shifters and latches. Alternatively, nowadays there are wide ranges of reconfigurable devices which are capable of implementing complex applications, so EHW may take the form of reconfigurable devices as well.

One of the advantages of the evolvable hardware approach over traditional design method is that it can be applied to application with only high level specifications: identifying the objective is easier than specifying the implementation particulars. Taking robot controller for an example, it is difficult to determine whether a particular output from the system is superior to another directly without feeding the output to the activator of the robot. On the other hand, we can distinguish how well a controller is doing when it is manipulating a robot in an environment. In addition, the design of the controller itself is automated by the employment of evolutionary process.

We present the principles of evolutionary design methodology first in section 2.2. Various well established evolutionary algorithms are overviewed in the next

section 2.3. More details about evolvable hardware are discussed in section 2.4, which includes highlights of existing evolved digital systems, as well as analogue counterparts. In the last section 2.5, we conclude this chapter with summarizing the approach we were followed in this research.

2.2 Principles of Evolutionary Design

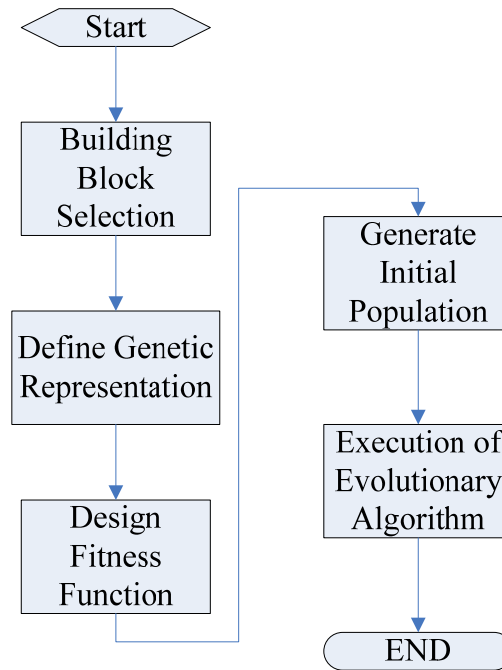


Figure 2-1 Flow Chart of Evolutionary Design⁴

Applying the evolutionary design methodology is quite different from the conventional approach. First of all, it normally requires selecting the building blocks or components available to the evolutionary algorithm to explore, and the interconnection structures between these parts. Second, a genetic representation of the solutions should be defined which is capable of describing what building blocks to deploy and how to connect them. Afterwards, a *fitness* function should be designed to evaluate how well a solution encoded in the genetic representation is doing at accomplishing a specific task. Eventually, the evolutionary algorithm generates a pool of potential solutions encoded in their genetic representation and operates on them with the objective of augmenting the best and overall fitness of

⁴ The procedure “Execution of Evolutionary Algorithm” is demonstrated in Figure 2-5

the population. When a system is created by an evolutionary algorithm, we call the system is *evolved* (see Figure 2-1), rather than *designed* specifically as in the traditional approach.

On the contrary, conventional hardware design procedure generally consists of writing functionality specification, synthesising an abstract implementation, implementing each components or sub-systems and testing of the final design.

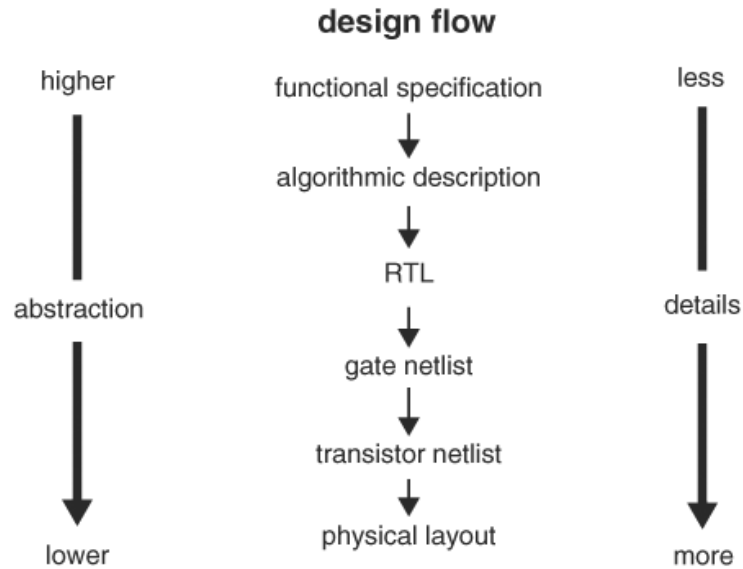


Figure 2-2 Conventional Circuit Design Flow

In Figure 2-2, a typical traditional design flow of circuits is depicted. It demonstrates that it is necessary to specify the lowest level of details of the implementation in this design methodology.

There are several methods of describing the abstract implementation. The most historical representation is schematics. An electronics schematic is a brief diagram of a circuit. It includes simplified and standardized pictogram of components used in the circuit and the connections between these components and power supply. The components could be logic gates or integrated chips, or a block representing another schematic.

Graphic representation of abstract implementation of Finite State Machines (FSM) includes state diagrams and state transition tables. A FSM or Finite Automation is a model of a behaviour defining possible states of a system, transitions between states and actions which trigger transitions. State diagrams represent these three key elements in a graph, while state transition tables

describe what state to move to given a combination of input and current state in tables.

With advancement in electronic technology, other representations of abstract implementation are introduced. For instance, Hardware Description Language (HDL) is essential for modern digital system design with explosive complexities. HDL is a textual representation of a system. HDL describes the operation and design of hardware as programming languages describe a piece of software. HDL may describe structures composed of basic logic elements, such as logic gates, and hierarchical custom modules.

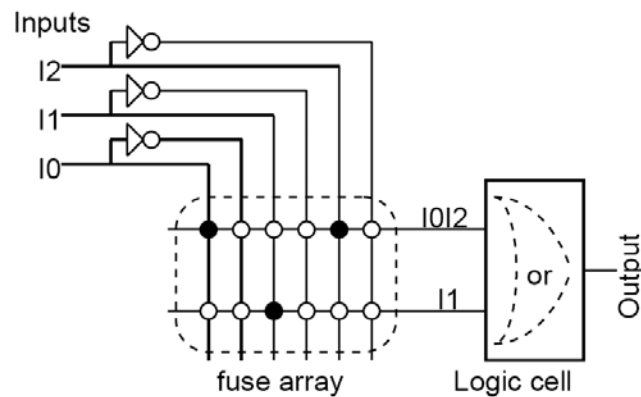


Figure 2-3 Simplified PLD Structure

The structure for one output pin [146]

Recently, reconfigurable devices are becoming more and more popular due to its fast prototyping nature. These devices can be reconfigured after manufacture and tailored for specific tasks using HDL or schematic design. The first device of reconfigurable device introduced was Programmable Logic Array (PLA) which can achieve combinational circuits (see Figure 2-3). Any Boolean logic function can be denoted by a sum of product terms (sum-of-product), the structure of PLD is sum-of-product oriented which makes it efficient to implement such circuits. Each output of PLD is from an OR gate which acts as the sum operation. The inputs to the OR gate are determined by configuration. Each horizontal line connected to the OR gate is an input, and represents an AND gate. The inputs to each AND gate are programmable, and include input signals, their inverse or remain unconnected. In Figure 2-3, a black dot in the horizontal line represents a connection between the input signal and the AND gate, while a white dot means

no connection of the two crossover lines. In the figure, the upper horizontal line (AND gate) connects to inputs I0 and I2 ($I0 \cdot I2$), while the lower AND gate connects to input I1 only. As a result, the *output* = $I0 \cdot I2 + I1$. The connection in the configurable AND gates are one-time programmable fuses or reconfigurable memory based element. These configurations can be considered as the genetic string for the circuits.

Programmable Array Logic (PAL), a form of improved PLA, was then announced which is capable of implementing simple sequential circuits. Afterwards, Complex Programmable Logic Device (CPLD) based on macro cells was introduced which provides more capacities in terms of available logic units.

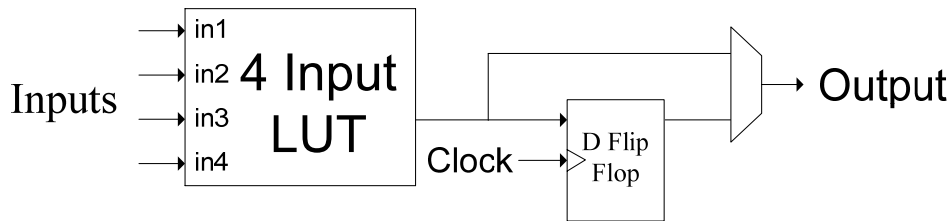


Figure 2-4 Typical Configurable Logic Block Architecture

With the announcement of Field Programmable Gate Array or FPGA, reconfigurable devices are brought to a new high level. FPGA features not only configurable components, but also customizable inter-connections. FPGA includes a larger number of programmable logic elements so it can even incorporate more complex systems, such as CPU cores. Although FPGAs from different vendors are different from each other substantially, they share the typical basic structures. The classic architecture comprises an array of configurable logic blocks (CLBs) and routing resources (routing channels). Each row or column of the CLB array may have more than one Input/Output pad fitted. The typical CLB structure is demonstrated in Figure 2-4. It consists of a 4 input look-up table (LUT) and a flip flop. Output from a CLB is only one bit: it can be either the output of the LUT or the flip flop which is controlled by a configuration bit. The content of LUT is also specified by the configuration string. The output can connect to routing channel below and the channel to the right. The routing channels are normally segmented: routing wires are connected

by configurable switch boxes. By default, switch boxes are not connected, while configuration string can program switch box to connect adjacent routing wires.

Reconfigurable devices for analogue systems are also available, called Field Programmable Analog Array (FPAA), which typically are composed of operational amplifiers, programmable capacitor arrays and programmable resistor arrays or configurable switches for switched-capacitor circuits.

In evolvable hardware, genetic encoding can take all of the above forms, such as HDL or an index graph of schematics. However, these representations are tending to be executed in software simulation rather than in hardware directly, due to the fact that the transformation from abstract implementation to hardware is computational intensive. Alternatively, the generic encoding can be the native form as the configuration format (normally a string of binary) of a device, such as FPGA, so that it can be downloaded to the device to assess the fitness directly.

FPGA is particularly popular in evolvable hardware community due to its highly reconfigurable architecture. In addition, the various configuration approaches to customize the device contribute to the popularity, including HDL, schematic design and native configuration string. The emergence of FPGA makes it possible to execute complex algorithms online with fast implementation methods, thus FPGA is one of the most ideal platform for evolvable hardware.

2.3 Evolutionary Algorithms

Evolutionary algorithm is one of the most significant components in the evolutionary design methodology which mimics the natural evolution with selection and differential replication procedures. Various forms of evolutionary algorithms have been proposed and investigated. Some of the most representative classes of evolutionary algorithm are considered to be *genetic algorithms*, *evolutionary programming*, *evolution strategies* and *genetic programming* [22] [25] [40].

2.3.1 Common Features

Although these four varieties of EA are proposed for different purposes, they all share several common properties.

One common feature of these algorithms is the use of population. A pool of candidate solutions to the specific task is usually referred to as a population of individuals.

Another characteristic shared by all the EAs is that it requires a scalar *fitness* measurement to evaluate the performance of a particular individual in the population to satisfy the targeted problem. In each iteration the fitness of all individuals is calculated. If the predefined stop criteria are met (normally it is in regards to the best fitness), then the EA stops. Otherwise, it selects some (in some cases all) of the individuals to replicate. Just as in nature, no two copies are exactly the same; the duplication procedure in EA also applies genetic variation operators. The common flow chart for EAs is shown in Figure 2-5.

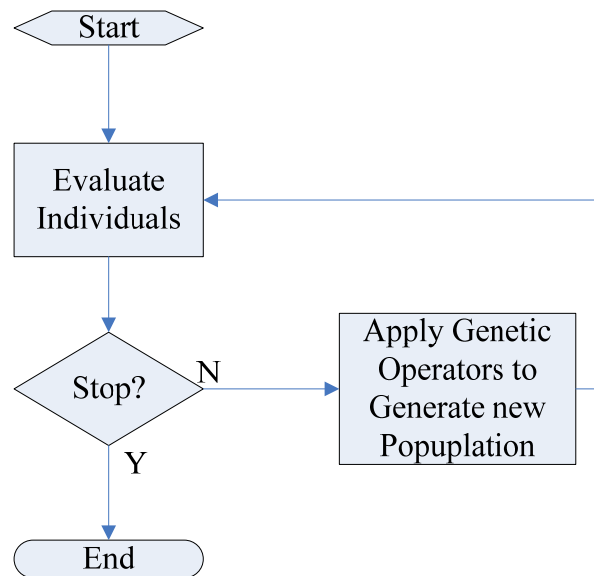


Figure 2-5 Flow chart of common EA

Before start of EA, an initial population is generated (see Figure 2-1). The fitness for all the individuals in the population is evaluated first. If stop criteria are not met (stop criteria normally are set as target fitness or number of generation), new population is generated by applying genetic operators and it loops back to evaluating individuals. Otherwise, if stop criteria are satisfied, EA exits.

Mutation and crossover are the most popular genetic operators in the community. Mutation involves modifying a part of the chromosome to generate a new offspring, while crossover generates new generation by combining parts of chromosome from (normally) two parental individuals. These reproduce

offspring forms the next generation and are exposed to the same selection procedure if stop criteria are not satisfied.

In evolvable hardware context, binary encoding and its derivate are widely used. In Figure 2-6, the mutation normally deployed in binary encoding EA is demonstrated. In the resulting chromosome (on the right hand side of “=>”), the bold digits are the modified bits by the mutation operator. Mutation only requires one and only one individual to generate one offspring.

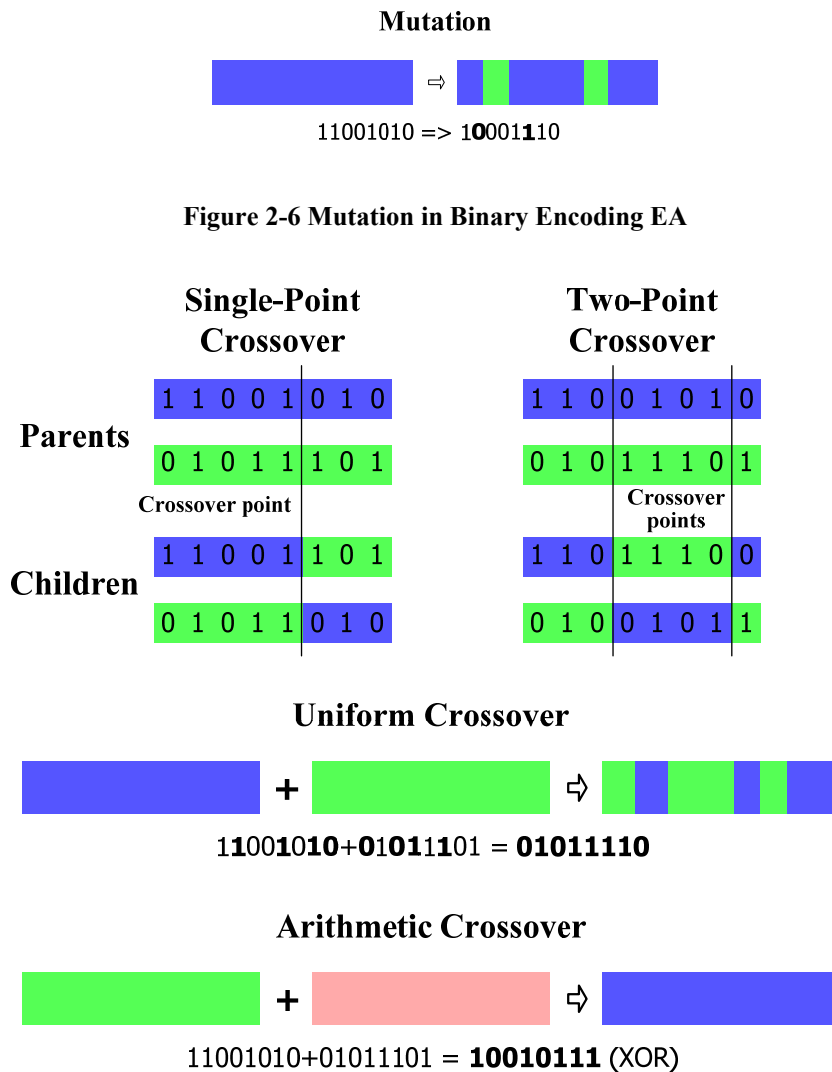


Figure 2-7 Crossover in Binary Encoding EA

On the other hand, crossover can have several forms. In Figure 2-7, four types of crossover operators are illustrated. Crossover requires at least two individuals (parents) to generate one or two offspring. In single point crossover, one point in the parents is selected, and one part is exchanged between the two parents to

generate to two new offspring, while in two-point crossover, two points are selected and the part between the points are swapped between the two parents. Other two types of crossover are also demonstrated in the figure. In uniform crossover, each bit in the offspring is selected randomly from the two parents. The bold digits are the selected ones. Another possible crossover operator is arithmetic based: the resulting offspring is generated by an arithmetic operator, such as XOR, AND, performed on the two parents.

Spears compared the relative importance of crossover and mutation in [134]: “Mutation serves to create random diversity in the population, while crossover serves as an accelerator that promotes emergent behaviour from components. The meta-issue, then, is the relative importance of diversity and construction. For the GA community, this is also related to the balance between exploration and exploitation.” It is suggested that mutation plays more important role than crossover in population with less diversity. It is also argued that the difference between mutation and crossover is not as obvious as it seems, and they can be considered as two extreme forms of a general genetic operator.

As for evolvable hardware, due to the resource constraints, population is limited and as a consequence the diversity in the population is normally relatively low. Thus mutation should have more impact in the EA than crossover. In addition, mutation is easier to implement than crossover. In this thesis, only mutation operator is employed.

The word *mapping* is often used in EHW to depict the decoding procedure from chromosome to the actual presentation (from genotype to phenotype).

2.3.2 Overview of EAs

Genetic Algorithms (GAs) were originally proposed to explore the adaptive systems principles [26] - [28]. It encodes the parameters of a system as bit strings which consist of genes. A gene is considered to be several bits representing a single parameter. As each gene is the direct binary representation of a parameter, it is straightforward to apply the chromosome to hardware. Generally, GAs make use of both mutation and crossover. Random bit flip is deployed as mutation, while exchanging sub-strings is used as crossover mechanism. The split of genes

is arbitrary and it can encode any parameters, which makes it a universal optimization technique for complex systems.

On the other hand, Evolutionary Programming (EP) was intended to carry out intelligent behaviour simulation using Finite State Machines (FSM) [29] [30]. They normally take the form of two functions, first of which represents the transition subject to a given combination of current state and inputs. The other function defines the outputs in a particular state with a set of inputs. Genetic variation operators can modify the available states and the transition between them. Outputs of the FSM can also be changed.

Similar to GA, Evolution Strategy (ES) was originally designed to be a parameter optimization method for aero technology devices [31] - [34]. However, rather than encoding the parameters in their binary forms, ES makes use of the parameter directly. It maintains a list of value pairs: the first one is the object parameter, which is the target to be optimized, while the second one is the strategy parameter which controls the mutation rate of the corresponding first value. The mutation of the first value in each pair is performed by adding a normal distributed number with the deviation given by the second value in the pair. The mutation of the first value in each pair is performed by adding a normal distributed number with the deviation given by the second value in the pair. The strategy parameter is mutated by randomly increasing or decreasing the value. Crossover is carried out by selecting each pair from its two parents with 50% probability. When the presentation of a task is a list of numbers, ESs are sometimes used in EHW.

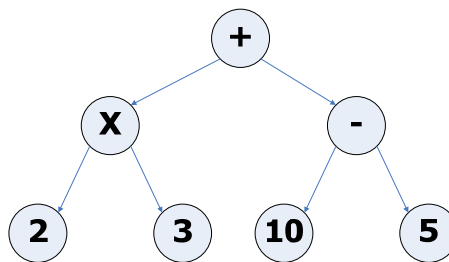


Figure 2-8 GP Genotype Demonstration

Genetic Programming (GP) was developed as an automated method of finding a “program” to solve specific problems [35] [36]. The original GP is a multi-branches tree (see Figure 2-8). The leaves of the tree are the input parameters, while each node is a function which operates on the branches it connects to. The result of the node is passed further on the branch to its parent as an input. The

result of the root node is used as the output of the system. Mutation and crossover operators specific to GP are designed. Mutation can modify the function of nodes and the value of leaves. Crossover can chop a branch and clone one from another parent.

2.3.3 Comparison of different EAs

The essential difference between these four types of EAs relies on the different chromosome representation. Due to the diversity of the encoding mechanisms, the genetic operators are also specialized for each kind of EA (see Table 2-1).

In general, EAs belong to the population based family of search and optimization. There are some other related techniques, such as simulated annealing (SA) [91][92]. Rather than inspired by biology, it is modelled after annealing in metallurgy. It normally only has one individual in a population. In each step, the current individual is replaced by a random solution, determined with a probability that relies on the current fitness and a global parameter (called temperature T). T is gradually decreased, so that at the beginning the new solution is generated almost completely randomly when T is large, while it converges to the global optimum as T goes to zero.

EA	Representation	Genetic Operators		
		Mutation	Crossover	Other
GA	Bit Strings	Bit flips	Substring exchange	
EP	Functions	Changing transition rules/output		Add or remove stats
ES	List of value pairs	Adding a random number	Uniform Crossover	
GP	Tree-like structure	Change the value or function of a node	Subtree exchange	

Table 2-1 Differences between the four types of EA

In [93], EP, ES and GA were compared against a set of different parameter optimization problems. It is suggested that, as their main design purpose, EP and ES are more suited for parameter optimization. GA is more oriented for general problem evolution. It is also found that because of its self adaptation of mutation rates, ES can converge more quickly than GA.

In this project, genetic programming was employed to search for solutions of the problems presented. The deployed GP will be discussed in further details in section 5.6.

2.4 Evolvable Hardware

When evolutionary algorithms are applied to the implementation and/or design of hardware, such as electronic circuits [24], we can realize evolvable hardware (EHW) [43].

The first possibility of EHW was brought to us by the Programmable Logic Array (PLA) devices. Nowadays, more complicated and higher density Field Programmable Gate Arrays (FPGA) are available in reasonable prices, which provide an ideal platform for EHW.

In this section, we introduce several existing classification of EHW first. Afterwards, we focus on a particular type of classification and present previous research projects and their achievements.

2.4.1 Classification of Evolvable Hardware

Based on different criteria, different classification systems are conceived. In general, two types of classification are used.

In regarding to where the evolutionary algorithm and the evaluation of individuals are implemented, there are two methodologies of EHW [44][45]. One is Extrinsic Evolvable Hardware (EEHW) [57] - [60], the other is called Intrinsic Evolvable Hardware (IEHW) [42] [47]- [55]. As the name indicating, in the former type of EHW the EA is realized in a computer (i.e. software) as well as the evaluation of each individual and the elite solution is then implemented on the hardware, while in the IEHW case the EA is carried out at least partially (possible completely) in hardware. In this thesis, two sub types of IEHW are defined: full-IEHW and semi-IEHW. The former term refers to systems in which evolutionary process and the evaluation of population are completely independent from the host PC, meanwhile the latter one refers to these which only parts of the system (the evaluation component for instance) are realized on the hardware.

EEHW is relatively straightforward to implement comparing with IEHW, because the EA is implemented in software program. However, IEHW has an advantage over EEHW in that in general hardware is much faster than software simulation, especially with regard to evaluating digital circuits. Also it allows physical characteristics of the system to be part of the evolutionary process. An EEHW will be presented in Chapter 7 and an IEHW will be discussed in Chapter 8.

Another popular system classifying EHW depending on whether any constraints are imposed in the possible hardware configuration. In intrinsic evolvable hardware, if the encoding of the problem can access all possible configurations of the underlying hardware without any limitations, then this is *unconstrained* EHW. This technique can explore unknown characteristics of the fundamental hardware and materials (normally silicon) of the hardware.

There is another not so popular classification system, which divides EHW depending on whether the evolutionary algorithm is realized in a chip or in the computer, and termed them as *on-chip* and *off-chip* EHW [94].

2.4.2 Extrinsic Evolvable Hardware

Considerable efforts have been devoted to the EHW research field. Due to the relative simplicity of EEHW, most of the research carried out is in the extrinsic area.

EHW can result in novel implementation of well known functionality. Miller successfully evolved 6 tap digital finite impulse response (FIR) filters with only multiplexers as building blocks [143]. Cartesian Genetic Programming (CGP) (see section 5.6) was utilized to encode circuits [4]. An array with 7x7 logic elements was employed to achieve the target FIR filter. The evolved circuits are fed with sample signals and the fitness is evaluated by the output similarity to the desired filtered signal. Individuals that displayed the targeted band-pass filtering behaviour are evolved successfully. It was observed that no explicit of multiplications or additions were in the evolved circuits. It was emphasised that no known mathematical way to implement a FIR filter with only multiplexers. This approach can be referred to as schematic evolution, in which a representation of circuits is evolved.

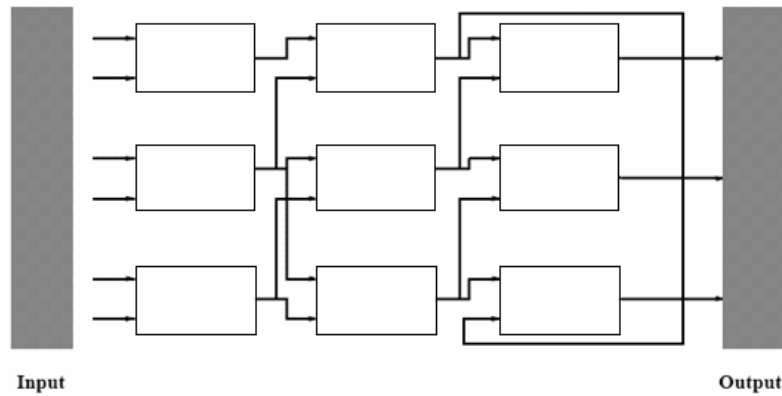


Figure 2-9 Circuit Representation in Miller's work

The two inputs of each gate (square block in the figure) can only connect to the outputs of its left columns of gates [144]

Schematic evolution can also lead to novel implementations with fewer resources than human designed solutions. Miller et al. evidenced this by evolving 1x1, 2x2 and 3x3 bit multipliers using CGP which require less two-input gates than the best known conventional design [144]: a 3x3 multiplier implemented with 21 gates (14 two-input gates and 7 MUX) was evolved, while the most efficient conventional design requires 26 gates (24 two-input gates and 2 MUX). (see Figure 2-9)

In [150], Kumar et al. proposed a *virtual reconfigurable circuit* (VRC) on top of normal FPGAs. The VRC is composed of *processing elements* (PEs), each of which can be configured to perform a certain binary functions. PEs are interconnected as shown in 10: there are 25 PEs, with 24 are laid out in a 4x6 matrix and the remaining one as the output PE. All PEs in a column can only connect to outputs of PEs located on the 2 columns on the left. A PE has 2 8-bit inputs and one 8-bit output, and it may have one of 16 possible binary functions. They successfully evolved an image filter which outperforms conventional human designs: the evolved filter can obtain better images from distorted original images than human designed filters. The VRC can be translated to a CGP encoding straightforward: with a 4x7 molecules of CGP, specifying one of the molecule on the right most column as the output.

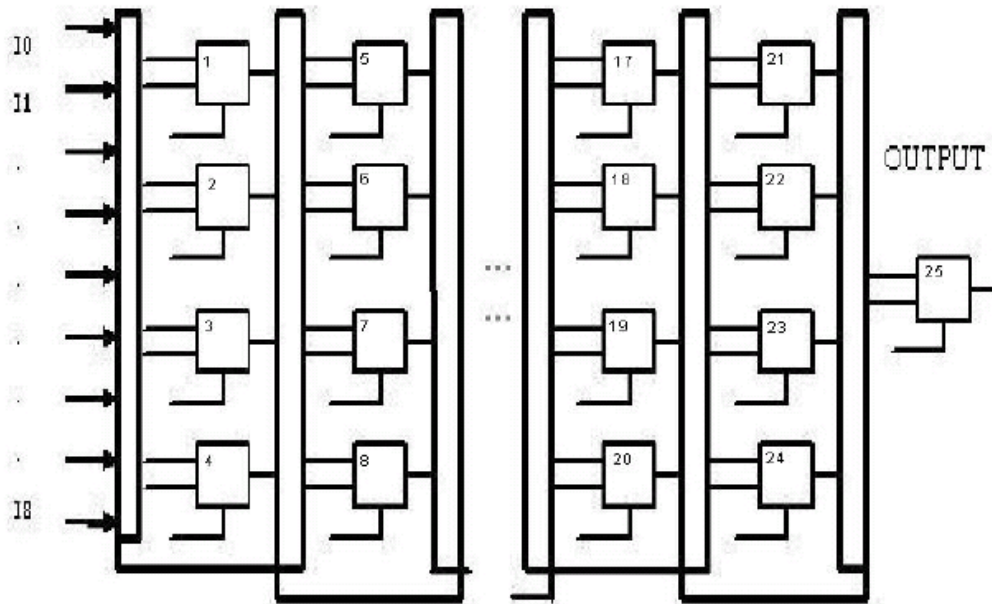


Figure 2-10 Architecture of VRC [150]

More recently, Masiero et al. looked at how to deploy EHW to solve designing issues in a merging field: molecular circuit. Rather than trying to build smaller components of circuit, this novel approach is to use individual or a small collection of molecules to achieve computational functions, such as AND gate [140]. They designed a SPICE model to simulate molecule level logic gates and employed EHW to integrate basic gates to form diodes. They demonstrated that, with the help of EHW, building blocks of molecule circuits can be achieved. This demonstrates that how EHW can be helpful to achieve novel designs.

All these researches mentioned above demonstrated that, with the help of EHW, it is possible to obtain novel designs which can be superior or more resource efficient than human designed solutions. Thus EHW was chosen as an ideal methodology to explore novel designs in this thesis.

2.4.3 Intrinsic Evolvable Hardware

Despite of being more sophisticated than EEHW, more and more resources and endeavours have been being invested into IEHW systems to obtain more superior performance.

FPGA, the ideal platform for IEHW, makes use of binary strings to configure its functionality and intra-connections. However, with some FPGAs, not all possible combinations of configuration binary strings will yield a working circuit.

In Fact, some of these configurations can even permanently damage the hardware [46]. One way to avoid that is to only modify a valid design [46]: a 8-bit counter was successfully evolved using this method. In [47] Smith et al. followed a similar route: a secure intrinsic abstract layer was presented for Virtex FPGA which basically mimics a XC6200 FPGA that can not have harmful configurations.

Another approach to workaround the issue is to only evolve logic function of the target application, not including routing. This route is followed by Tyrrell et al.: a custom-off-the-shelf hardware friendly evolutionary system was proposed [48] and a robot controller was implemented to verify the system. This approach is normally more efficient in terms of resource consumption. In addition, it can feature faster re-configuration. In this work, a Virtex FPGA will be targeted. In order to support safe intrinsic evolvability, only logic of the circuits will be evolved. The logic is built using evolvable molecule, which will be discussed in 8.1.1.

A real-world application, compression of image data with very high resolution, was considered in [49] and the compression method and hardware chip architecture were devised as well. They deployed FPGA to implement their IEHW platform. The results suggested that when implemented in chip it can achieve a speed 2 magnitude faster than its software counterpart.

All of the projects about IEHW referenced above are of type semi-IEHW. However full-IEHW was increasingly attracting researchers.

Kajihara et al. investigated two chips in [50] and [51]: the first one was a gate-level evolvable chip which integrated all the necessary components of a full-IEHW, including evaluation of individuals, GA and control logic, and it was found that this chip could dramatically improve the evolution process; a DSP reconfiguration structure for neural network application was composed in the latter paper and it exhibited two orders of magnitude performance improvement than a Sun Ultra 2. Recently, full-IEHW platforms were designed and implemented for two industry applications [54] and [56] to improve performance so that they are more suitable to be employed in commercial products. An online adaptive and IEHW was realized to carry out the packet switching tasks in

network [54], while Yang and others proposed and implemented an IEHW platform to evolve novel image filter digital circuits employing CGP [56].

More recently, Rajan et al. conceived a dynamically reconfigurable circuit (DRC) based on FPGA to implement image filters [145]. The implementation in FPGA can react to the noise in the input image: if the noise increases, the FPGA will be dynamically reconfigured to deploy a more accurate (but slower) DRC filter. When the noise level drops down, a faster filter will be *swapped* back. They demonstrated that, with the FPGA implementation, 2 magnitude of speedup can be obtained compared to running it in software with a Pentium 4 2.8GHz processor.

Based on these researches, an IEHW platform will be investigated in Chapter 8 to obtain a better performance.

2.5 Limitation of EHW

Although EHW alone is quite competitive to discover solutions for some applications, it does have several limitations, one of which is that evolution can not provide desirable integrated fault tolerance and adaptation, widely processed by multi-cellular living organism.

One way to tackle the constraint of EHW is to draw inspirations from biological principles. In this thesis, development principles will be discussed and considered in order to overcome the issue.

2.6 Summary

In this chapter, we gave an overview of the principles of evolutionary design and highlighted the key components in evolutionary design, the Evolutionary Algorithms. Afterwards, we discussed in further detail about a specific area of evolutionary design, evolvable hardware, which is the focus of this thesis. We presented the classification of EHW and reviewed some of the previous related projects in extrinsic evolvable hardware and intrinsic sub areas, which show the potentials of evolvable hardware as a design approach.

Chapter 2 Evolutionary Design of Electronic Systems

In this project, a special type of GP evolutionary algorithm will be used and we will deal with digital systems with extrinsic evolvable hardware first, and then implement an intrinsic platform to improve performance.

To deal with the limitations of EHW, developmental principles of biology will be considered. In the following chapter, the principle is reviewed and developmental electronic/mathematic systems which are the inspiration sources of this work.

Chapter 3

Development Principles and Development Systems

We overviewed the evolutionary design methodology and evolvable hardware principles in the previous chapter. Different types of EHW projects were also reviewed to demonstrate the characteristics of evolvable hardware. As development is the evolutionary selected approach to build complex systems, such as multi-cellular living creatures, developmental principles will be the sources of inspiration for this project.

The focus of this chapter is to present the basic principles of biological development and discuss some of the mathematical models proposed for development. Finally we highlight some existing development models specifically designed to be oriented towards electronic systems.

3.1 Biological Development Principles

Modern developmental biology⁵ focuses on biological development⁶, cell growth and cell differentiation. Biological development studies the mechanisms that control the spatial distribution of different specialized types of cells and which lead to the forms of tissues, organs, organisms and the body anatomy.

Specifically, this sub section briefly reviews the principles of embryo development, which is the direct inspiration source of this research work.

3.1.1 Basic Concepts

Biology introduces a large number of glossaries, some of which have been borrowed by electronics to represent similar but fundamentally different

⁵ Developmental biology is a science branch in which the growth and development of organisms are studied.

⁶ Biological development is also known as morphogenesis.

principles. We discuss some of these glossaries and their corresponding meanings in computation and electronic systems.

3.1.1.1 Chromosomes and Genes

Deoxyribonucleic Acid (DNA) is a type of acid found in nucleus of living cellular organism cells, which conveys genetic information that directs how to carry out the functions of the cell and the process of replication. The overall DNAs in an organism are normally organised in the form of chromosomes. DNA directs development of cells via the synthesis of proteins which perform the activities and replication of the cell.

The minimal heredity units encoded in the chromosomes are called genes. They determine the physics characteristics, development and behaviour of the organism and encode the information necessary for proteins synthesis.

In the evolvable hardware context, chromosome is borrowed from biology to represent the aggregate of heredity information. Chromosomes in EHW may take the form of strings of integers, trees or indexed graphs.

3.1.1.2 Genotype and Phenotype

Before introducing genotype and phenotype, another definition is required to define them: alleles. In most living beings, one gene can be present in more than one viable chromosome. An allele refers to any one of them.

The aggregation of the alleles for a specific gene is termed the genotype for the gene. The physical outcome or appearance of the gene is called phenotype of the gene. In the perspective of a entire organism, rather than a particular gene, genotype can also be used to denote all the inheritable information, while phenotype in this context refers to the appearance and or performance of the individual.

Although genotype is the most influencing factor, the phenotype is not exclusively determined by genotype. Environmental factors also play an important role in the finalization of the phenotype from the genotype. On the other hand, a particular phenotype may result from a wide variety of slightly different genotypes. [18]

Normally, in EHW context, each gene only has one allele, so genotype is used to refer to the entire inheritable information (including all the genes). Phenotype in EHW also refers to the net outcome of the genotype in the form of final solutions.

3.1.2 Biological Living Organism Cells

Component		Function
Nuclear region	DNA	<ul style="list-style-type: none">• Carrier of hereditary information
	Enzymatic systems	<ul style="list-style-type: none">• Duplicating the hereditary information
Cytoplasm		<ul style="list-style-type: none">• Synthesizing proteins and most of the molecules• Generation of chemical energy• Conduction of stimulus signals• Transport of materials
Membrane		<ul style="list-style-type: none">• A continuous outer boundary• Selection of water-soluble molecules• Recognition and binding of molecules

Table 3-1 Biological Cell Components and their functions

Cells are the elementary structural and functional units of all living cellular organisms: the sub-structures of the cell are unable to retain the basic qualities of life, such as growing, reproducing or responding to outside stimuli in a coordinated, but potentially independent fashion. Thanks to the extremely organized molecular and biochemical systems, cells can store information in the form of heredity material DNA, utilizing it to synthesize cellular molecules and generate the necessary power to support all these functions from sources of chemical energy. In addition, cells are capable of motility and can tolerant environmental fluctuations by changing their internal biochemical systems. One of the most significant functions of cells is the ability to duplicate themselves by passing on their inherited information and their major biochemical and molecular systems to their offspring in the cellular reproduction process. [1]

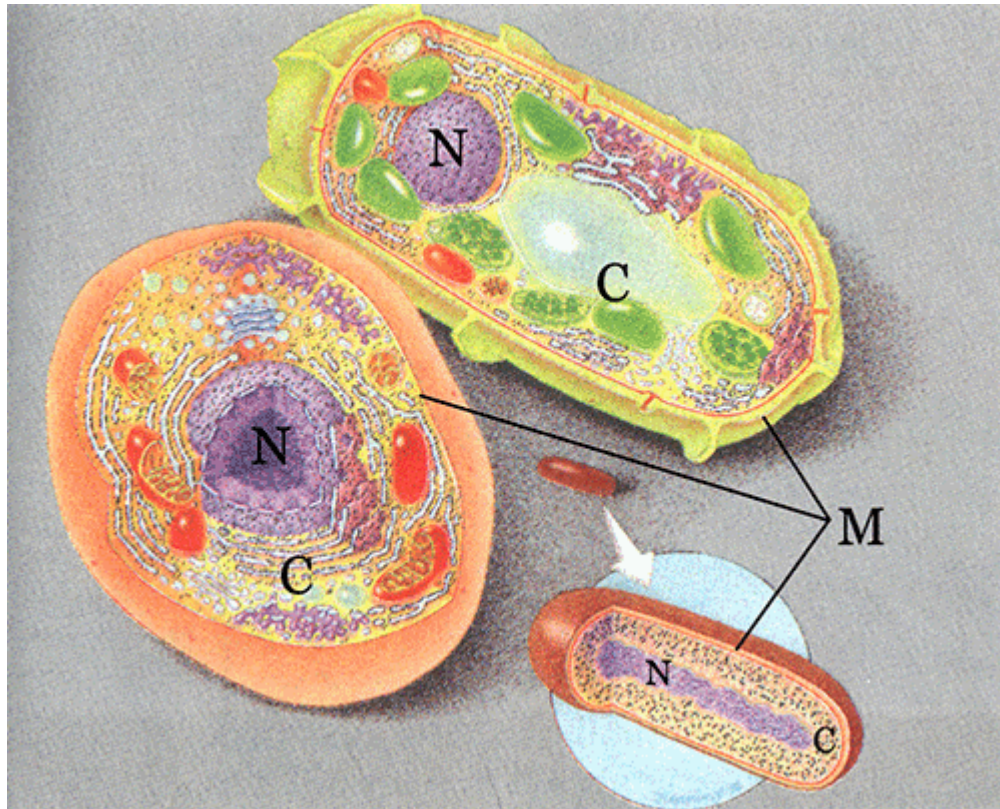


Figure 3-1 Overview Biological Cell Structure⁷

The nuclear region (N), the cytoplasm (C) and the membrane (M)

In spite of cells having an immense number of diversified forms in multi-cellular living organisms in terms of size, shape and function (see Figure 3-1), all cells have two common major internal regions divided by their nature of function (See Table 3-1): The first core part is *nuclear region* containing DNA molecules and enzymatic systems. The DNA molecules are the carriers of the hereditary information which directs cell growth and duplication in the cell. The enzymatic systems perform the task of duplicating the hereditary information for both cell reproduction and instructing the synthesis of various proteins. The other core region, the *cytoplasm*, synthesizes proteins and most of the other molecules necessary for the proper functioning of a living cell. This region also conducts other vital tasks, among the most significant of which are the generation of chemical energy usable by the cell, the conduction of stimuli from outside to interior, the transport of materials to and from the cell, and cell movement. [1]

⁷ The original picture was published in [1]

Another common feature of all cells is a continuous outer boundary, called *membrane*, that separates the cell contents from the exterior: membranes maintain cells as distinct environments and collections of matter. The most important task of membranes is to control what material can pass from one side to another and the amount of the particular material in a cell. [1]

3.1.3 Stages of Development

The development of an embryo is determined by genes, which control where, when and how many proteins are synthesized [1]. Complex interactions between various proteins and between proteins and genes within cells and hence interactions between cells are set up by activities of genes. It is these interactions that control how the embryo develops. These interactions are normally referred to as Gene Regulatory Networks (GRN).

Embryo development involves five stages: cell division, the emergence of pattern, change in form, cell differentiation and growth. In the beginning, the zygote undertakes a rapid cell division⁸, called cleavage. Next, a spatial and temporal pattern of cell activities is organized within the embryo so that a well-ordered structure can develop, the process of which is called pattern formation. Then the form is changed to lay the fundamental for further development. Afterwards, cell differentiation (see Figure 3-2) is triggered in which cells become structurally and functionally different from each other, ending up as distinct cell types. Finally, they grow to increase their sizes in their determined location in order to divide into daughter cells..

The model proposed in this paper contains only two of these aspects, cell division and differentiation. Since in hardware, no new resources can be created as cells are pre-formatted and their number can not be increased, “growth” is used in this thesis to refer to “cell division”.

⁸ Cell division is the process a cell replicates its internal materials and split into two offspring cells, called daughter cells.

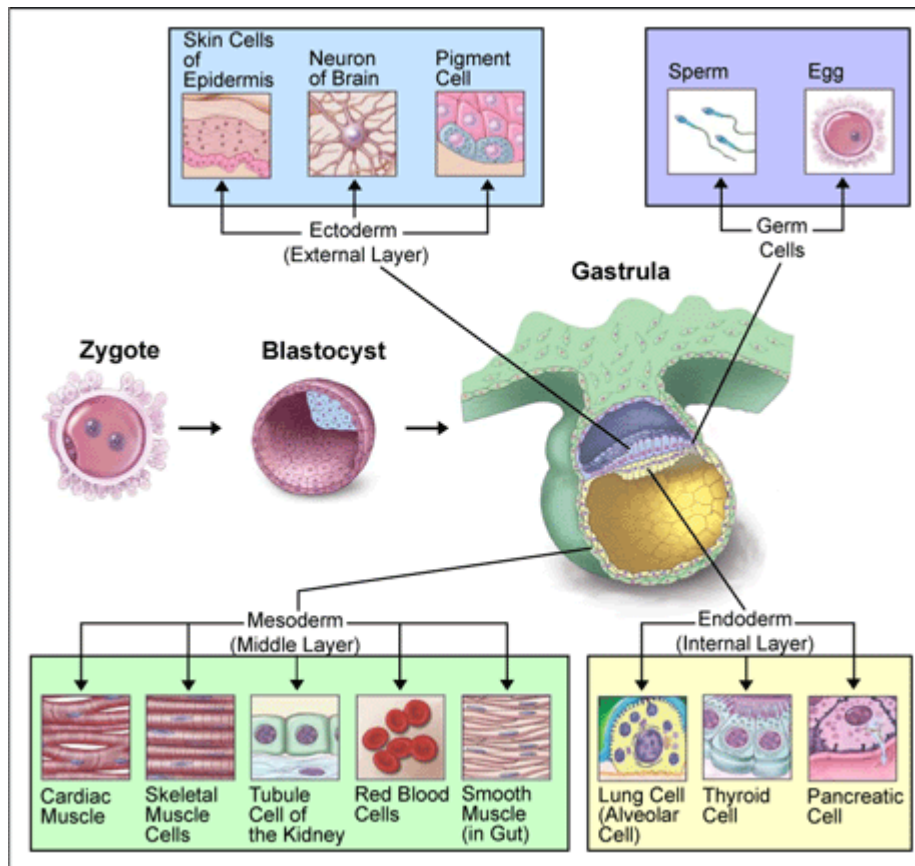


Figure 3-2 Differentiation of tissues⁹

The *inner cell mass* (with a line to the up-right corner) will construct the germ cells (eggs and sperm) as well as cells derived from all three germ layers (*ectoderm*, *mesoderm*, and *endoderm*), which include nerve cells, muscle cells, skin cells, blood cells, bone cells, and cartilage (demonstrated in the bottom panel).

3.1.4 Characteristics of Biological Development

Although all the cells in the embryo derive themselves from the same zygote and contain identical chromosomes¹⁰, eventually they develop into diverse types of cells. The cell differentiation emerges as a result of differences in gene activities which lead to the synthesis of different proteins that in turn perform distinct functions and behave in different manner.

One of the decisive matters for the differentiation is inductive interactions by means of chemicals or proteins between cells: these inter-signal mechanisms can

⁹ Picture is obtained from [95].

¹⁰ In most cases these cells contain the same genetic information, with rare exceptions.

make cells different from each other. In addition, the response to these inductive signals depends on the state of each cell which can lead to further diverge.

The different concentration of a particular chemical or protein can convey positional information: the concentration may be low at one end of axis, and gradually rise when going towards the other end. Patterning formation in embryo involves the interpretation of positional information, as well as lateral inhibition.

Another intrinsic characteristic of development is that the process is progressive: The fate of cells becomes determined gradually along with the progress. [1]

A similar mechanism in the proposed model will be introduced to provide the basis for cells to differentiate from each other and perform different functions.

3.2 Mathematical Abstractions of Development

In order to make use of development principles efficiently in other areas, several mathematical abstract models are proposed to demonstrate some of the characteristics of development in a well defined structural approach.

3.2.1 Gene Regulatory Network Abstractions

Stuart Kauffman is among the pioneers in modelling gene regulatory network of biological development [96]. He proposed the use Random Boolean Networks (RBN) to represent the activity of genes: either activated (expressed, represented by binary 1) or repressed (deactivated, represented by binary 0).

Each gene is denoted by a node in a directed graph. Arrows are introduced in the graph to represent relations between causal nodes. Each node in the graph is a Boolean function (AND, OR, XOR or an arbitrary one specified by a truth table) of the outputs from the nodes with arrows pointing towards it. Time is discrete in this model: in each time step, each node is evaluated based on the prior states of its input nodes.

Random in the name RBN refers to the fact that the connections between nodes are randomized, so each node can have any of the other nodes as its inputs.

It is demonstrated that the RBN can exhibit attractors, which are stable cycles of states, including one (point attractors) or more states (cycle attractors). These

attractors can be considered as different types of cells. This may be used to estimate the number of possible cell types in an organism with similar sized gene regulatory networks. RBN is also observed to have paths to change from one attractor to another, as biological cells differentiate. In [110], a classification of random Boolean networks is discussed in details.

Since the inception of RBN, other abstractions of GRN have been conceived, including Petri nets [97][98], Bayesian networks [99][100][101], graphical Gaussian models [102][103][104], Stochastic Process Calculi [105][106] and differential equations [107]. These more complex models tend to incorporate different degrees of gene expression, rather than simply either on or off.

3.2.2 L-Systems

Lindenmayer System (L-System) is a set of rules applying to a collection of symbols. It was initially designed to model the plant development by Aristid Lindenmayer in 1968 [108].

Variables	A B
Constants	none
Start	A
Rules	A → AB B → A

Table 3-2 Definition of Lindenmayer's L-system for algae growth

L-System consists of four ordered lists:

- **V** (the *alphabet*) is the set of symbols which can be replaced by other symbols. These symbols are termed *variables*.
- **S** is the set of symbols which can not be replaced by other symbols. They are called *constant*.
- **ω** (the *start*) is the initial state of the system. It only contains symbols defined in **V**.

- **P** defines a set of rules which specify the way a variable can be replaced by symbols in **V** and **S**. A rule contains two components: *predecessor* (or left-hand side) and the *successor* (or right-hand side).

The rules are applied to the start state iteratively. Lindenmayer's original L-system for growing algae is shown in Table 3-2. The first four production of this L-system is demonstrated in Table 3-3.

Step	Production
0	A
1	AB
2	ABA
3	ABAAB
4	ABAABABA

Table 3-3 The Production of Lindenmayer's L-system

The predecessors of the rules in the example only refer to a single symbol, so this L-system is *context-free*. Otherwise, if the left-hand side of rules contain more than one symbol, it is termed a *context-sensitive* L-system. Nowadays, L-systems are known as *parametric* L-systems, in which symbols can have associated parameters and these parameters can occur in the rewriting rules.



Figure 3-3 'Weeds' generated from a L-system in three dimensions [109]

The recursive characteristic of the rules makes L-system well adapted to generate self-similarity and fractal like forms, including plant and other natural

patterns. A verisimilitude of weeds is shown in Figure 3-3 which is generated from a 3-D L-system.

3.3 Developmental Systems

In electronic systems, development principles have inspired several novel approaches to implement applications. These approaches are normally based on biological multi-cellular organisms: Identical cells can differentiate under the supervision of a development mechanism.

As evolutionary algorithms are generally utilized in these design approaches, a fitness function should be conceived first. In most cases, the fitness is related to the *grown* or *developed* organism. As a result, the fitness is evaluated after the development is complete (or partially complete). If the development is not deterministic, it may be necessary to assess the fitness more than once for each individual in a population. For instance, the development depends on a probability based parameter.

In the evolvable hardware context, development systems employed can be classified into three major categories: general cellular program, gene regulatory network based model and those using rewriting rules (such as L-system). In addition, other models not covered in the previous three categories will be reviewed as well.

3.3.1 Gene Regulatory Networks

In [111], Gordon and Bentley proposed a development model for the evolution of evolvable systems, in which a minimalistic gene regulatory network with binary protein concentrations achieved in a 2D cellular array was deployed. The structure of cells contains four inputs, a 4-input look-up table (LUT) acting as the functional part and one output. Development is controlled by rules which are composed of *preconditions* and *postconditions*. A precondition defines what proteins must be present for this rule to be activated. The postcondition of a rule is applied if and only if the rule is activated. Postcondition may generate proteins, modify the LUT contents or the input and output connectivity. As only one bit concentration for each protein, diffusion and decay of proteins can not be modelled in this system. The system was implemented in Virtex FPGA and each

individual was evaluated in the FPGA, thus this was a semi-IEHW platform. In this model each cell merely occupied one CLB in Virtex FPGA. A two bit adder with carry was the target function to evolve. It was found that although the evolvable capability is relatively inferior compared to direct mapping systems with regards to either mean fitness or best solutions found given the allocated resources (cells), this system demonstrated significant potential: the created patterns by the evolved rules were the same as human designed traditional ripple-carry adders. Considering this fact, the authors argued that, although no correct 2-bit adder was found, if more resources were made available to the EHW, perfect solutions for a 2-bit adder would be evolvable. Afterwards, the authors tried to reduce the number of rules in the hope of improving evolvability by reducing the search space. This modification did not improve results which suggested that reducing the search space does not necessarily lead to better evolved solutions. What is important is the evolvable potential of the overall search space imposed by the biases we design. In [112], the authors improved the model to incorporate diffusion and a better mechanism of detecting protein concentrations around a cell. They also replaced the evolutionary algorithm with a hill-climbing algorithm. With these modifications, a 2 bit adder can be evolved successfully. Finally, a simpler architecture layer was adopted and larger adders and even parity circuits could be evolved [129].

A more realistic model of the gene regulatory network was considered by Koopman and Roggen to explore the dynamics of continuous protein concentrations [113]. The continuous protein concentration is represented in fixed-point bit-serial arithmetic and encoded with minimal number of bits tailored for compact hardware implementation. Protein diffusion and decomposition are embraced in this model. A gene regulatory network contained in each cell interprets the genome to activate or repress genes. This model was used to evolve predefined sized organisms. It was demonstrated that the dynamics of the developmental systems gave rise to tolerance of diverse kinds of faults in the cellular organism.

Another direction of more realistic gene regulatory network model is to have structural proteins. This approach was followed by Bentley when he proposed a gene regulatory network making use of fractal proteins [114]. Fractal proteins are

defined as a finite square subset of the Mandelbrot set [116], which potentially can provide as much complexity (or more) as biological proteins folding. An integer representation of protein concentrations was deployed. The concentration is subject to the production and diffusion of the protein in question. Proteins in the environment of the cells are also included in the model which can affect the outcome of development. An evolutionary algorithm was used to design the gene interaction network to produce a specific gene activation pattern which can be employed to carry out certain tasks. This model was applied to a robot controller the task of which is to guide the robot through an environment avoiding obstacles. In addition to diverse solutions, the fractal gene regulatory network demonstrated the ability to identify reusable patterns and modules, which is a significant property of development.

Hardware implementation of gene regulatory networks was considered in [185] where a custom FPGA design was presented to improve the performance of the evaluation of GRN.

3.3.2 L-Systems and other Rewriting Rule Based Systems

Deploying a simplified mathematical abstraction of development is easier than modelling the real gene regulatory networks. L-system is one of the popular and well studied development abstracts and a qualified candidate for investigation in evolvable hardware.

In [115], Haddow et al. proposed to apply L-System to the genotype to tackle the scalability issue in evolvable hardware by reducing the length of chromosome. In order to achieve unconstrained evolution, they implemented a layer on top of a Xilinx Virtex FPGA so that all available configurations are safe and can be downloaded to the FPGA [117]. Two types of writing rules were introduced in their work: change rules and growth rules. The former type replaces a portion of the genotype with another part of equivalent length. On the other hand, the growth rule can allocate new resources (termed *s-block* in the system) on unused spaces around the *s-block* which the rule is applied to. The starting state and the writing rules of the L-System are obtained by evolution using a standard genetic algorithm. They tried to use this model to find circuits

with specific distribution of s-blocks on a 16x16 cell array with moderate success. Afterwards, they reduced the number of s-block types to direct evolution to find specific kinds of systems, and introduced contextual rules in the L-System [119]. In addition, cell death can be specified in writing rules. Tasks explored in the work included growing into a specific sized structure within a certain developmental steps, differentiating into several types of cells within a limited steps and pattern formation in which a symmetrical pattern of s-blocks has to be evolved in a certain number of developmental steps. The development system was executed by a dedicated hardware processor and the evolution part was implemented in software on a PC [118]. In 2005, Haddow et al. presented an intrinsic EHW platform to implement a cellular computing machine (CCM) [148]. The platform features a centralized genome repository and development process. More recently, in 2007, Haddow reported a development model integrated with environmental signals and adaptive behaviours [149]: an evolved solution has to be able to response to present environment and survive in it, which was called phenotypic plasticity.

Another rewriting rule based development system is discussed in [121]. The model represents circuits in Hardware Description Language (HDL) and evolves HDL according to a HDL grammar via developmental rewriting rules. A tree like genotype is employed which controls the rewriting: each node in the chromosome tree represents a production rule which is iteratively applied to an initial symbol, as in the normal L-System. Besides common genetic operators, mutation and crossover, several specific operators tailored for this system are introduced, including duplication, insertion and deletion, all of which manipulate functional blocks in the model. Fitness is evaluated by simulating the resulting HDL in an emulator. It is suggested that due to the possibility of denoting high level building blocks, such as latches and shifters in HDL, encoding circuits with HDL usually leads to more compact representation than a direct mapping using basic logic gates. Controllers for artificial ants were evolved to guide the ants to follow a food trail which may be corrupted. In following research [122], it is further suggested that this approach might be more scalable for more complex tasks since this model showed the potential to exploit phenotype regularities.

3.3.3 Cell Programs

Another more general approach to development implementation is cell program, which are not limited to gene and proteins interaction, or rewriting rules. A cell program refers to a function running in a cell to determine the output of the cell based on its current state and current inputs from environmental (including signals from neighbour cells).

Miller extended Cartesian Genetic Programming (CGP) (section 5.6 for details) to a developmental system, called Developmental Cartesian Genetic Programming (Developmental CGP) [120], in which each cell contains a cell program to determine its functionality and development. As digital circuits were evolved, the functional parts are logic gates. The developmental part of the cell program in each cell takes the current position, connectivity and functionality of this cell as inputs, and generates the new connectivity and functionality. In addition, the developmental program also specify whether this cell *divide* or not in the next developmental step. As in nature, the development starts from a single initial cell, the zygote and all the cells share the same cell program encoded in CGP. The cell program is evolved to find solutions for binary adders and even-parity functions. It is demonstrated that the DCGP can provide generalization to some extent: by simply increasing the developmental steps, the evolved solution can implement the same logic function with one additional input.

In subsequent work [12], a method for evolving a developmental program inside a cell to create multi-cellular organisms of arbitrary size is described and its characteristics are analyzed. This time, the cell program maps the input conditions to output behavior, which includes chemical production, change of cell type and whether to divide and if so where to grow (see Figure 3-4 for the cell structure). Miller found that the artificial organism is able to organize itself into well defined patterns of differentiated cell types, such as the French Flag. Emergent properties, self-repair and adaptation for instance, were observed in the solutions found using evolutionary strategies: one of the matured organisms was subjected to harsh damage, and it reconstructed itself in several steps of growth (with some scarring); another organism responded appropriately to global environmental signals which lead to the metamorphosis of the whole organism.

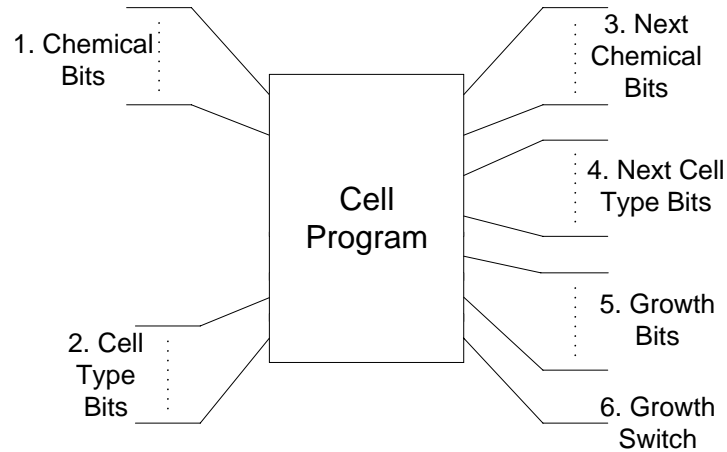


Figure 3-4 Illustration of Cell Program Structure in Development CGP.

The cell program in each cell determines the chemical concentration (3 in the graph) and cell type (4) for the next time step based on the local information available to this cell, including the current chemical concentration (1) and types (2) of the neighbour cells and itself. In addition, the cell program also decides whether this cell should grow (6) and if so, to which direction¹¹ (5).

In this thesis, the author follows a similar direction and further investigates the cell program potentials. We consider a CGP based development technique that makes use of a chemical signal which gives the system beneficial emergent properties.

CGP was selected as the genotype representation because it does not seem to suffer from program length bloat and introns (unconnected nodes) in the genotype, which can improve the effectiveness of the evolution with the benefit of not requiring any processing power until it is connected to the actual program.[189]

3.3.4 Other Systems

Some other models are also used in developmental approaches for evolvable hardware.

¹¹ One cell can grow to its eight Moore neighbour cells, so 8 bits are required to represent all directions.

In [123], it was demonstrated by Koza et al. that it is possible to evolve electronic systems with genetic programming. Although development was not explicitly explored, the decoding of the tree-structured chromosome to circuits could be considered as a primitive developmental process. It was further pointed out in [59] and [124] that with automatically defined functions, the model could be improved using modularity and potential reuse of structures. Low-pass filters and two-band crossover filters, among others, were successfully evolved using this model [125]. Later, Koza et al. focused on evolving solutions identical (or even better) to filed patent for various applications. In [95], six examples of GP duplicating functions of patented electrical circuits are presented. More recently, GP was deployed to evolve 6 optical lens systems [37]. One of the evolved solutions infringed an issued patent, while 2 others discovered novel designs duplicate or (improve upon) existing patents. All these experiments made that point that, genetic programming has the potential to deliver human-competitive machine intelligence.

In 2000, Ohntishi and Takagi proposed a hierarchical feedback model borrowing ideas from biological development principles [69] and the model was applied to applications to verify its feasibility for complex systems: The model could successfully use a simple expression to describe an image.

A multi-cellular organization was designed based on MUXTREE [71] cell which was implemented on off-shelf FPGAs and RAM and this model exhibited self-reproduction and self-repair capabilities.

There are some other projects which involve special decoding mechanisms to map the genotype to phenotype. As the decoding systems generate the final circuits step by step, they could be considered as *simplified development model* as well.

One example is a linear representation of analog circuits, conceived by Lohn and others [126][127]. The genotype contains a sequence of *instructions* which are executed by a *circuit composing robot*. In each instruction, the type of element (such as resistor, capacitor etc.), the parameter or value of the element and the connectivity with other elements are specified. Some unfeasible circuits were evolved, however the representation could generate some well known

topologies in manually designed circuits. In addition, analog filters and transistor based amplifiers were successfully evolved.

Another simplified development model was proposed by Mattiussi [128]. The genotype employed in this model is in the form of strings and substring matching is utilized for decoding. Components are separated in the genotype by predefined special strings (or start tags). The details of each component, including the parameters (if any) and connectivity are delimited by other predefined tags (sub tags). The decoding process starts from the beginning of the genotype. Whenever it encounters a start tag, it scans from this point to extract the details of the element. If necessary information for this element (e.g. parameter or connectivity) is missing, then this element is not expressed in the phenotype (the resulting circuit). With this design, even after substantial modification in the genotype by genetic operators, the encoding can still be decoded into a circuit. In addition, this mechanism prevents the phenotype changing significantly under the pressure of genetic operators. The authors evolved voltage reference sources successfully with this presentation and suggested that this model may be used by other similar analog systems, such as gene regulatory network and neural network.

In contrast to most of those previously mentioned, a direct genotype-phenotype mapping developmental system was proposed by Tempesti and colleagues, which is called Embryonics [130][131][132]. Another noteworthy difference of Embryonics is that no evolutionary algorithm is involved: the system is completely designed manually.

The system is implemented in a 2D array of identical cells. Each cell contains the same program, considered as genotype of the organism. The differentiation of cells in the organism is achieved by different coordinate: a part of the cell program determines the coordinate of the current cell and propagate signals to surrounding ones. After the coordination is initialized for all the cells involved, the organism may begin to perform the desired task. Fault-tolerance is provided in the system with spare cells. When a faulty cell is detected in the organism, the coordination is re-initialized, and the faulty one is ignored and spare cells are activated to replace it. Essentially, after re-establish the organism, the system can always come back to its original functionality, given spare resources (cells) are available.

3.4 Summary

In this chapter the background of biological developmental principles is reviewed. Development in multi-cellular organism is one of the fundamental mechanisms for living creatures to adopt the environment and tolerate errors in its internal, and it is a viable inspiration source for novel electronic systems design approaches.

In addition we described some of the mathematical abstractions of development and reviewed development models for electronic systems. As a more general approach, Cell Program is used in this thesis for the representation of the development system. In addition, efforts will be put to introduce functionality to development system, which normally only deals with pattern formation problems.

As described in this chapter, most of the systems reviewed are extrinsic or semi-intrinsic. In this thesis, we will also focus on an intrinsic developmental evolvable platform.

This work is to provide a novel approach for fault-tolerant system design, so in the next chapter, the definition and background on this technique will be reviewed and bio-inspired fault-tolerant systems will be discussed.

Chapter 4

Fault-tolerant Techniques

Fault-tolerance is a technique applied to the implementation of systems to ensure their reliability. With the complexity of systems nowadays increasing dramatically, fault tolerant technique becomes more and more critical, especially in crucial applications, such as aircraft control.

This chapter presents the background of fault tolerant techniques in the electronic system design context. First some basic principles and terms widely used in fault tolerant field are reviewed. In section 4.2, the developing history of fault tolerance dating back to 1940's is briefly overviewed. In the next two sections, 4.3 and 4.4, the conventional fault tolerant techniques and their bio-inspired counterparts are presented.

4.1 Fundamental Concepts and Definitions

One of the most fundamental concepts in fault tolerance field is the notion *reliability*, which is defined as the probability that a system can operate and fulfil its required function under specified environment for a given period of time. *Availability* of a system is the probability that the system operates correctly at a given instant time. [151]

The term *dependability* is used to refer to a concept which treats reliability and availability as distinct features of a system. Dependability can be described by either reliability if the system is non-repairable, or availability given the system is repairable. [151]

Another set of nomenclature is introduced in [152] specifically designed for computer systems by Baron et al.

“*Dependability* is defined as the trustworthiness of a computer system such that reliance can justifiably be placed on the service it delivers. Dependable has several aspects:

- with respect to the readiness for usage, dependable means *available*;

- with respect to the continuity of service, dependable means *reliable*;
- with respect to the avoidance of catastrophic consequences on the environment, dependable means *safe*;
- with respect to the prevention of unauthorized access and/or handling of information, dependable means *secure*. ” [152]

A *fault* in a system is the cause of an *error* which is liable to trigger a system failure. If a system can not perform its designed function and/or deliver the expected service, a *failure* of the system is occurred.

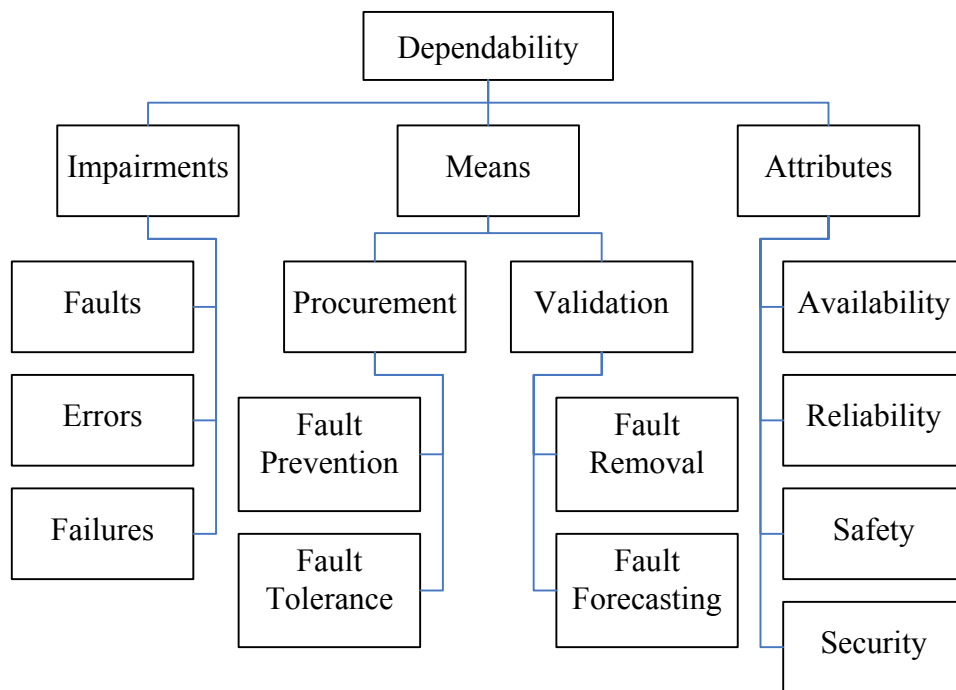


Figure 4-1 Computer system dependability tree designed by Baron et al. [152]

In order to achieve the desired level of dependability to minimize system failures, a set of approaches could be deployed which can be classified into four types [152]:

- Fault prevention: attempt to preclude faults from happening;
- Fault tolerance: try to provide the expected service even if faults occur in the system;
- Fault removal: attempt to reduce the number of faults, and/or their significance;

- Fault forecasting: try to assess the consequence and number of present faults, and future potential fault occurrences.

The first two categories of approaches can be considered as dependability procurement which ensure the obtaining of dependability. On the other hand, the last two types can be treated as validation means of dependability, which estimate how to obtain a certain level of confidence of dependability.

Figure 4-1 shows the various aspects of dependability.

4.1.1 Classification of System Faults

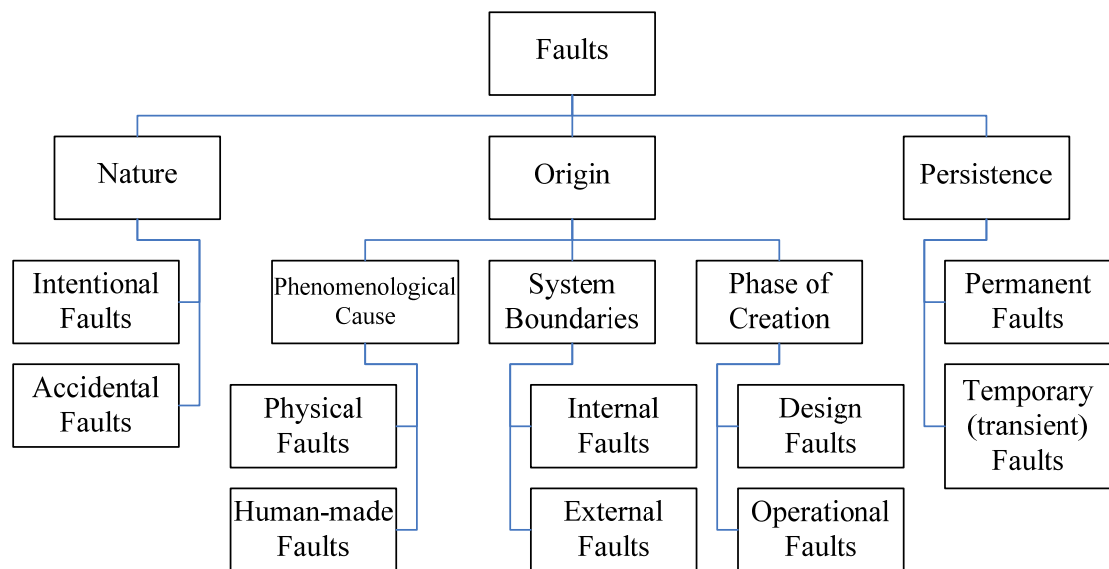


Figure 4-2 Faults classification Categories [152]

System faults have extremely diverse sources. Based on different view of points, three major aspects can be identified: nature, origin and persistence feature of a fault [152]. In Figure 4-2, the points of view on faults are summarized.

According to the nature of the introduction of faults, they can be classified into intentional (faults are created deliberately) and accidental faults (those are introduced inadvertently).

Faults can also be categorized according to their origin. The origin category can be further divided into 3 sub aspects: There are two types of phenomenological causes, physical and human-made; based on whether the faults occur within the system boundaries, there are internal faults and external

ones; two sub categories can be identified with regards to when the fault is introduced, design phase or operational phase.

With regards to persistence of faults, there are permanent faults and temporary ones. Permanent faults are those that damage the system (normally this refers to the hardware implementation) permanently, longer than the execution time of the system or leading to more serious failures which the entire system can not recover from without substitution of one or more components. On the contrast, the operating system can restore to working order from temporary faults without any replacement and no hardware damages.

Nature	Origin			Persistence	Usual Name
	Phenomenological cause	System boundaries	Phase of creation		
Accidental	Physical	-	Opera.	Perm.	Physical Faults
Accidental	Physical	External	Opera.	Temp.	Transient Faults
Accidental	-	Internal	-	Temp.	Intermittent Faults
Accidental	Human-made	Internal	Design	Perm.	Design Faults
Accidental	Human-made	External	Opera.	Temp.	Interaction faults
Intentional	Human-made	Internal	Design	-	Malicious logic
Intentional	Human-made	External	Opera.	-	Intrusions

Table 4-1 some well known Fault Types and their classification.

(“-“in a cell denotes that the corresponding column does not matter)

In Table 4-1, some commonly used fault names and their classification are listed. Most types of faults are of accidental nature, with only two exceptions: malicious logic and intrusions (hackers in the context of internet applications), both of which are intentional actions of person involved. Among accidental faults, two are caused by human, either those who designing it or operating it. Physical environment can also lead to faults in a system, which can be further categorized into two types: physical faults and transient faults. The former one lead to permanent damage to a system, while the latter one is temporary, caused by “external faults originated from the physical environment” [152].

Table 4-2 lists ratio of transient faults to permanent faults in some known systems. On each row of this table, a digital system is listed. For example, the last row shows a 37 units of 1M RAM using MOS manufactory technology and deployed parity checking to detect faults. MTTF of transient faults is 7% of MTTF of permanent faults (as shown in the last column), which means transient faults are 14 times ($1/0.07$) more likely to happen than permanent faults. A

similar transient to permanent faults MTTF ratio is observed for all the listed systems: only small portion of system failures are caused by permanent faults [17][66][67][68]. Thus this work concentrates on transient fault-tolerance.

System	Technol-ogy	Detection Mechanism	MTTF (transient)	MTTF (permanent)	transient / permanent
CMUA PDP-10	ECL	Parity	44 hrs.	800-1600 hrs.	0.03-0.06
Cm LSI-11	NMOS	Diagnostics	128 hrs.	4200 hrs.	0:03
C.vmp	TMR LSI-11	Crash	97-328 hrs.	4900 hrs.	0.02-0.07
Telettra	TTL	Mismatch	80-170 hrs.	1300 hrs.	0.06-0.13
1 M x 37 RAM	MOS	(Parity)	106 hrs.	1450 hrs.	0.07

Table 4-2 Ratios of transient errors to permanent failures [66]
(Data from [175][176][177][178][179][180])

This kind of fault is mainly prompted by environment variation, such as “power jitter, electromagnetic interference, ionization due to cosmic rays, or alpha particles from packaging materials” [65]. Some other factors may also contribute to the introduction of transient faults, taking physics faults of the materials of the devices for example, which will cause some component oscillate between different states repeatedly [65].

4.1.2 Phases of Fault Tolerance

A four-phase view of fault tolerance was conceived by Anderson and Lee in 1981 [153], which includes *Error detection*, *Damage confinement and assessment*, *Error recovery* and *Fault treatment and continued system service*.

These four phases are not necessarily present in all real world fault tolerance systems and normally the distinction between them is not so obvious for they may have considerable interactions in some particular systems. Besides Error detection, the other three phases may have other order in which a system can deploy.

4.1.2.1 Error detection

In general, Error detection is the first phase for fault tolerance. This phase is intended to detect presence of errors in the system resulted from faults, so that some actions can be taken to prevent errors from provoking system failures. When an error is detected, the system can either handle it by using other

available resources or mask the error and carry on (which normally leads to some kind of deterioration of service it can provide) [153].

Theoretically, if every possible error in the system can be identified, no system failure would arise. In practice, due to system complexity, project budget and other limitations, far from close to this theoretical level of error manipulation can be realized.

4.1.2.2 Damage confinement and assessment

Once an error happens in a system, it should be confined at the lowest level to replaceable or repairable modules to preclude the error from propagating to other modules of the system, which would minimize the lost from the error and recovery time. In typical hierarchical error-containment boundaries, voters are extensively deployed to establish the boundaries.

Besides damage confinement, once an error occurs, damage assessment would be performed to estimate which components of the system should be replace or repaired. As it is highly dependent on the structure of the operational system, damage assessment normally involves subjective decisions and assumptions, and the assessment is normally application specific.

4.1.2.3 Error recovery

Error recovery techniques may be performed once an error is detected. The seriousness of damage an error can cause could be predictable or unpredictable. *Forward error recovery* is the technique normally deployed for situation when the damage can be anticipated, while *backward error recovery* is employed in unanticipated damage.

The success of forward error recovery techniques relies on the accuracy of predicted damage, thus this is an application specific approach. Error correcting code, such as Hamming code, Hsiao code, Reed-Solomon code¹², is a typical kind of forward error recovery techniques: by introducing redundant information in the code, errors in the code can be not only pinpointed but also corrected.

¹² See [151] for more details about these error correcting codes.

On the other hand, for more complex systems, it is impossible to predict all possible errors, so backward error recovery techniques were conceived. Rather than trying to correct the errors after they are detected as in forward error recovery, when backward error recovery is utilized, the entire state of a system is restored to a prior state which is known to be free of errors. In essence, that mimics the reversal of time: given the fault is a temporary one, then the restoring of a correct prior state should remove all errors which resulted from that fault. The restoration of the state is called a reset of the system. There are two commonly used terms to describe two types of resets, *cold start* and *warm reset*. The former one refers to restoring to the initial state of the system, while restoring to other operational states is called warm reset.

4.1.2.4 Fault treatment and continued system service

When applicable, only recovering from errors is not sufficient, eliminating the faults which provoked those errors is more desirable, so the same errors would not manifest over and over again.

Fault treatment is the process of eradicating faults when errors are detected. This process normally contains two sub phases: *fault location* and *system repair*. After system repair, system service can be re-established.

In some cases, it is too complicated to design and/or implement fault treatment in a system, so this phase is ignored [153]. Given the system meets the following characteristics, it could be efficacious:

1. Error recovery deployed can successfully deal with recurring faults;
2. Future operation of the system avoids the faults fortuitously;
3. The faults are of transient nature.

4.2 History of Fault Tolerance Systems

One of the first appearances of fault tolerance was employed in the first digital computer to surmount the low reliability of their fundamental building components, transistors. Taking Bell Relay Computers (BRC) for instance, in some of the early BRCs, two central processing units were deployed so that when one unit encounters an error, the other can continue with the execution of

the next instruction. In later versions of BRC, a retry mechanism was employed in which an instruction is re-executed if errors occur. Errors may also arise when transferring data, thus the IBM 650, UNIVAC, and the Whirlwind-I computers¹³, among others, utilized parity check to verify the received data. Full redundancy of the Arithmetic Logic Unit (ALU) in computers is considered to be first introduced in EDVAC computer delivered for production operation in 1949. EDVAC won't execute the next instruction unless the outputs from the two ALUs agree. [154]

John von Neumann is generally considered to be the first pioneer of theoretical work in fault tolerant computing. In the 50s, he published an article, "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", in which the majority voting mechanism was presented, and its impact on the reliability of a system was analysed. [159]

As the reliability of transistors was improved significantly over the years, the importance of fault tolerance suffered a decrease of attention, while the major concern was shifted to improve performance and capacity of systems and reliability of system mostly relied on the reliability of components, such as the enhanced transistors. However, when the time came that computers were utilized in performing more essential tasks, such as space exploration and aircraft control, fault tolerant techniques emerged as a critical issue again. [158]

There has been rapid development in the field of fault tolerance since 1970s. Many famed journals, such as IEEE Micro, the Proceedings of the IEEE, the Journal of Design Automation and Fault Tolerant Computing, and the IEEE Transactions on Computers, published special issues on the topic of fault tolerance, dependability and reliability in regular bases. Several conferences were being held for fault tolerant techniques as well. One of the earliest events is Symposium on Fault-Tolerant Computing (FTCS), which has been held annually since 1971. The proceeding of the conference covers a wide range of topics on

¹³ For more details about these early (50s or 60s) computers, please consult [155], [156] and [157] respectively.

fault tolerant techniques contributed by researchers and engineers from academic institutions, government laboratories and industries.

Parallel systems are used more and more widely due to more demanding target applications. Some research focused on one type of parallel systems, massively parallel processing networks, such as artificial neural networks. Normally, a distribution of the available information (knowledge) throughout the elements ensures the reliability of the system when one or a relatively small portion of the processors (elements) fail.

On the other hand, some research draw inspiration from biological principles, which will be covered in section 4.4, after briefing of conventional techniques.

4.3 Conventional Fault-tolerant Design

Over the years, several fault tolerant techniques were conceived. In order to counteract the faults occur in the system, at least one type of *redundancy* is incorporated. Redundancy in a system refers to additional resources, which can be excluded given no faults arise without affecting the desired function. The redundant resources may take several forms, additional hardware (*hardware redundancy*), replicate information (*information redundancy*), additional software (*software redundancy*), extra time (*time redundancy*) or a combination of two or more of these. [151]

Although the fault tolerant techniques listed above can enhanced the reliability of a system considerably, it should be highlighted that fault tolerance alone can not achieve the desired system reliability level without prerequisite of other traditional reliability objectives, such as making use of reliable components and refined interconnections. “Thus, while fault tolerance can be used to increase significantly the reliability of an already reliable system, it is of little use if the original system is unreliable”. [151]

4.3.1 Hardware Redundancy

Making additional hardware available in a system is the most common way to improve reliability. Hardware redundancy can have two approaches, static and dynamic redundancy. [151]

In the static variant, fault tolerance is realized without actually detecting which component fails. This technique is also called *masking redundancy* which masks the effect of the faulty component instantaneously. One typical static redundancy technique is Triple Modular Redundancy (TMR), which was first suggested by von Neumann [159]. In TMR, a voting mechanism is deployed to deliver the majority vote as output. Error correcting code is another example of static redundancy. [151]

On the other hand, dynamic redundancy can identify faulty components and recover from the consequence of a fault. A system with this kind of redundancy normally consists of several modules but at any given time, only one instance of them is operating. In addition, there is a fault detection and recovery module which constantly monitors the operating module. When a fault is detected in the active module, one of the spare ones replaces it to maintain the proper functioning of the system. [151]

Dynamic hardware redundancy can also take other forms, such as spare resources (processing units). An embryonic array model with spare cells was proposed in [21] and [37]. More recently, the model was extended to introduce not only spare molecules in each cell, but also spare cells [64]: if a molecule in a cell is faulty, a spare one can be used. If it is impossible to make use of any remaining spare molecules in a cell, the cell will be killed and spare cells will need to be activated to replace the faulty cell. The model relies on explicit global coordinates of each cells to configure what each cell should carry out.

Mixture of static and dynamic redundancy is also possible, which leads to *hybrid redundancy*, which embraces several TMRs. When the active TMR fails, a spare one is used to replace it to resume the operation. [151]

4.3.2 Information Redundancy

Extra information can be made available in order to recover from a fault. For instance, in the case of error detecting and error correcting codes, extra check bits are appended to the original data bits. Information redundancy is usually adopted in state machines and memory systems. [151]

4.3.3 Software Redundancy

Even the most thoroughly tested software is not free from faults, so fault tolerance in software is also required. However, the techniques used in hardware can not be applied to software directly, as the quantification of the expected enhancement in reliability that can be realized by using additional software is impossible.

An *N-version programming* mechanism was proposed for software redundancy in [160], the concept of which is similar to the principle of hardware TMR. Instead of identical modules in TMR, independently implemented versions of programs are used in N-version programming. A voter is also utilized in this approach to obtain the final output from several implementation of the program. With the expense of more complicated voter implementation, “significantly similar” output from each program could be regarded as equivalent [161]. It was also suggested that the N-version programming exhibits tolerance against some categories of temporary hardware faults, besides enhanced reliability with regards to design faults. [162]

4.3.4 Time Redundancy

When dealing with temporary faults, time redundancy is commonly used, that involves the repetition, or rollback, of instructions [163], segments of programs [164] or even entire program [165], instantaneously after the detection of a fault [151]. Time redundancy is a type of backward error recovery techniques (discussed in section 4.1.2.3).

4.4 Conventional Transient Fault-tolerant Techniques

One of the most popular conventional technique deployed to tackle transient faults is backward error recovery technique [70][190][191][192].

In [70], Gomaa et al. proposed Chip-level Redundantly Threaded multiprocessor with Recovery (CRTR), which is specifically designed for multiple processors on a single chip. It makes use of two copies of an application, called leading and trailing threads, to check whether there is a fault. The result of

the two threads are compared and saved as checkpoint. When a fault is detected, the application will be reverted back to last correct checkpoint.

Melhem et al. incorporated transient fault tolerance to schedule algorithm in real-time systems. A dedicated task is used to save checkpoints of the system to enable recovery. They demonstrated that by non-uniformly distribute checkpoints taking into accounts of power consumption and reliability factors, up to 68% power can be reduced, while the system can still deliver desired reliability [190]. Another algorithm for optimal checkpoint insertion was proposed by Kwak et al. which uses reliability as a performance measure [191].

While all of the mentioned researches are software based technique to achieve transient fault tolerance, there are others who considered hardware to improve the performance. In [192], a transient fault tolerant architect for shared-memory multiprocessor environment was presented, which deploy parity protection, logging and checkpoints. It uses hardware to do logging and parity updates, while more complex operations, saving checkpoints and recovery, are still left to software.

4.5 Bio-inspired Fault-tolerant Techniques

The conventional fault tolerant techniques are widely used and serve their area well for many years. However, they have some intrinsic issues. Static hardware redundancy is prone to single point failure, as the correct output solely relies on reliability execution of the majority voting element. On the other hand, dynamic hardware redundancy suffers another problem: it demands error handling capability which will incontrovertibly increase the complexity of the system and its design cost. What's more, it is difficult to design such an error handling circuit which stores adequate information for recovery so that it can discover most transient faults, especially in those more complicated applications.

As the adoption of more and more complex systems for more challenging tasks shows no signs of deceleration, new techniques have to be conceived to overcome the issues associated with conventional ones.

A promising compensation of the drawbacks of the conventional techniques is to take inspiration from biology. Besides evolution (reviewed in Chapter 2), also

known as Phylogeny, development or Ontogeny (discussed in Chapter 3), another major biological principle can be motivated as an inspiration source, which is known as Epigenesis, or learning.

4.5.1 Phylogeny

Phylogeny is the study of the origin and evolution of a set of living organisms (species). It is commonly cited to refer to Evolution [166] (see also Chapter 2) in evolvable hardware context.

Phylogeny is the most studied area of the three principles in search for novel fault tolerant techniques. A fine-grained reconfigurable hardware architecture built on top of Field Programmable Transistor Array (FPTA) was presented in [167] by Stoica et al. With this architecture, they successfully demonstrated evolvable hardware can achieve fault tolerance: “evolution can recover functionality lost due to an increase in temperature” [167]. In their later work [168], they evolved fault tolerant digital and analog circuits using two approaches: fitness is evaluated against potential faults the circuit may encounter; the other approach is population based, where implicit information of the population statistics accumulated by evolution over generations are used.

Shanthi and Parthasarathi investigated structures of several available FPGAs and proposed a three-tier model for evolving fault tolerant digital systems on FPGA [169]. The model makes use of GA and structure redundancy built into FPGA to achieve fault tolerance. Required number of solutions to provide desired fault coverage was also explored.

4.5.2 Ontogeny

In ontogeny field, also known as ontogenesis or morphogenesis, the origin and the development of an organism from the fertilized egg to its maturity is studied. In evolvable hardware field, ontogeny is interchangeable with development.

Besides the researches reviewed in section 3.3, another developmental principle oriented fault tolerant model was conceived in [131] and a hardware implementation was realized, called BioWall [170]. A watch, named BioWatch, capable of self-repair and self-healing was successfully implemented with the

BioWall. Although in this work, human designed approach was utilized, in most developmental work, evolutionary approach is commonly used.

In a series of papers [19] [20] [21] written by Tyrrell etc., a bio-inspired asynchronous embryonic array was proposed and implemented, which exhibits fault-tolerance provided by a reconfiguration strategy. Cells in the embryonic array store recovery information, so that when faults are detected, reconfiguration can be carried out to replace the faulty cell with a spare one. The fault recovery procedure is transparently performed while the array is functioning.

4.5.3 Epigenesis

Epigenesis is normally used to refer the learning process after born, by which the individuals adapt the environment and obtain skills necessary for surviving.

Learning was utilized to design novel online categorization systems [171]. With the help of learning capacity, robot controller can improve their reaction to past events, such as to adapt to a new operating environments or to new robots [172] [173].

Learning is not so widely used in fault tolerant field as the other two biological principles. However Tyrrell and others suggested that in the future, learning may be used to discriminate correct from incorrect behaviours in electronic systems [174].

4.6 Summary

Fault tolerance is as critical an issue as the correct functioning of a system in modern society. In this chapter, the basic concepts of fault tolerance were introduced and the history of the technique, from middle of the last century to present day, was reviewed.

We briefly reviewed the classification of the conventional fault tolerant techniques, including their characteristics and typical implementation. Most conventional fault tolerance requires extra design in the system which requires the designers either to predict all faults may occur or to always save the entire states of a system so that if faults do happen it can rollback, both of which are

Chapter 4 Fault-tolerant Techniques

complicated to design or even impossible to achieve. In order to tackle these issues, several bio-inspired novel approaches were presented.

In order to achieve intrinsic fault tolerant, developmental principles is considered as in previous chapter. In the next chapter, the framework of the development inspired cellular module proposed will be discussed in detail.

Chapter 5

Development Cellular Model for Digital System

Considering the intrinsic superior characteristics of the development process explained in previous chapters, it is apparent that this biological principle provides an ideal candidate as a source inspiration which may lead to novel design of highly fault-tolerant systems. One of the most fundamental features of the development principle is the universal cell structure: each of the cells in a multi-cellular organism contains the entire genetic material, the genome. [1]

In order to apply this biological principle to digital systems, a practical mapping of a ‘digital cell’ must be established first. In section 5.1 the structure and functionality of biological cells are reviewed; the difficulties inevitably encountered in the transforming phase will be discussed in the first section as well; the various aspects of the development model proposed will be described in the following five sections: section 5.2 and 5.3 will give the overall structure of the proposed model, which features an integrated Execution Unit for each cell to carry out desired functionality; then chemical diffusion mechanism and the growth procedure of the digital organism will be demonstrated in section 5.4, 5.5 respectively; section 5.6 will discuss special cases for some cells in this model; In the next section, the genotype representation employed in this work will be presented; finally a summary, which briefs the most important aspects of the model that need to be taken forward, will be given in section 5.8.

5.1 Transforming of Biological Principles to Digital Systems

Some of the features of real biological cells described in previous chapters, which could result in highly fault-tolerant systems, are also eminently desirable in digital systems. However they are beyond the possibilities of current state-of-the-art microelectronic techniques in terms of manufacturing techniques and

electronics principles. Motility and real reproduction are among these kinds of features. Some of the other functions are not compulsory in digital systems, such as the conversion of chemical energy¹⁴ and the boundary maintenance via membrane, for real movement of cells is impossible in digital systems, so all components of a cell are fixed.

As stated in previous section, due to hardware limitation, only cell “growth” and differentiation are borrowed from biological development principles. However, the transforming of biological principles to electronic system is not straightforward in bio-inspired research fields.

The *State* of cells in this work is used to distinguish the types of cells. At least, two states should be classified to support the growth of the artificial organism: *living* and *dead* cells. A dead cell¹⁵ has not been specialized to a specific type and can not trigger their neighbours to alter their states, while a living cell can generate output via its circuits based on its inputs (from the environment) and is capable of signalling to its adjacent cells to modify their states. Dead cells are transparent to other live cells in that they propagate its input signals without any alteration to its outputs.

Cell growth in the digital organism refers to the process that a living cell signals one or more of its neighbours to transform into another state other than dead.

Differentiation of cells is achieved in this model by introducing multiple types of living states. So after receiving signals from one or more neighbours, a cell will transfer to one of the available living states.

In addition to the two simplifications described it is found that in order to map the development principles, some further issues must be considered to conceive a practical structure both for the digital cell and for the whole artificial organism.

¹⁴ Energy is possible to be introduced, but as it is not a core part of the development principle, so no energy will be considered in this work.

¹⁵ Dead cells are not faulty ones.

Proteins in real cells perform most of the activities of living organisms, such as stabilizing and controlling the activity of DNA, providing the transport and recognition functions of cellular membranes [2] and proteins themselves are synthesized directed by heredity information. However, in a hardware implementation, allowing a kind of intermediary layer such as protein would introduce much more complexity to the model. As a result, a more “direct” mapping from genotype to phenotype¹⁶ is employed in this work. Details will follow in section 5.7.

Proteins and other complex molecules act as messengers both in intercellular environment and the interior of cells while all kinds of activities of life rely on the correctness of the transmission of these signals. However, it is extremely difficult, if not impossible, to incorporate a large number of such signals in digital organisms because of the confined resources in the target hardware. As a result, only one chemical is introduced to transfer information between the digital cells. The chemical is indispensable for live cells: with zero chemicals, a cell stays in the dead state and it can not trigger adjacent cells to grow, which is achieved via a diffusion procedure as in natural world. Details will follow in section 5.4.

Another simplification imposed by the nature of silicon manufacture techniques is that the organism resides on a 2-dimension surface and each cell is considered to be square in shape.

5.2 Design Consideration

In the Miller’s previous work [3], each cell has dedicated outputs to indicate whether it will grow into a new position. All the cells are updated in a specific order (row-wise then column-wise), and later cells in the scan path will overwrite grow signals from earlier ones. For example, considering a 3x3 cell organism as shown in Table 5-1 (the number for each cell shown in this table is the order in which it is updated for each step), if cell 2 decides to grow to 3 while cell 6 also wants to do the same, the output from cell 6 will overwrite cell 2.

¹⁶ Phenotype includes cell structure and function.

The overwrite mechanism is quite arbitrary and depends on the scan path when updating, which inevitably introduces a bias to the organism: it always favours these cells encountered latter in the update path. Overwriting is only necessary when two or more cells signals to grow into the same position. If the cell does not output a grow signal, then there is no need to implement a overwrite mechanism at all. Thus, when designing this model, it was decided to get rid of any grow signals from each cell so that no bias will be introduced in the model.

1	2	3
4	5	6
7	8	9

Table 5-1 A 3x3 Digital Organism

To compensate the lacking of grow signals, each cell in the proposed model will determine its own next state based on locally available information (from its immediate neighbour cells). The assumption here is a bit different than those in [3]. In Miller's work, until a cell is grown into, it is considered dead or not existing. While in this model, all the cells in the digital organisms are considered as stem cells, which can determine how it differentiates on its own.

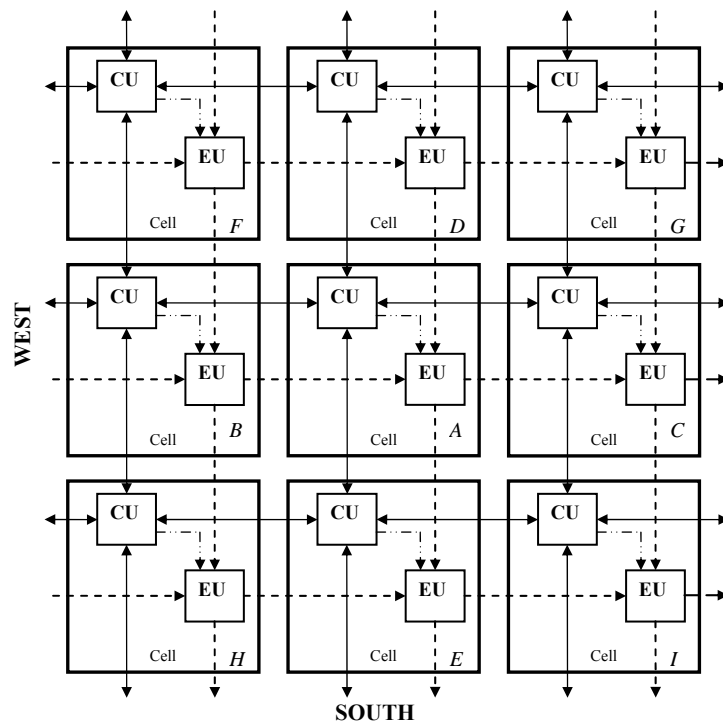
In order to bring combinational functionality to the digital organism, a dedicated sub component will be introduced for each cell, called Execution Unit (EU). The state of a cell will be fed to the EU within it as input, so that differentiated cell can carry on operations according to what type the cell is. EUs in adjacent cells will be interconnected to form EU network so that all the cells can collaboratively implement a specific functionality. The EU network is connected in a way to ensure feed-forward nature which guarantees that it implements a combinational circuit.

Similar to [3], the cells in the digital organism will only have access to local information, without any long range direct connections. However, unlike the model in [3], where each cell has 8 immediate neighbours, in the proposed model, there will be only 4, because of hardware resource consumption concerns.

5.3 Digital Cell Structure and Inter-Cell Connections

A “snap shot” of the digital organism is illustrated in Figure 5-1¹⁷. Every cell only has direct access to the information of its four adjacent cells: No direct long term interaction between non-adjoining cells is permitted in this model.

The internal structure of digital cells is shown in Figure 5-2. A digital cell is composed of three main components: Control Unit (CU), Execution Unit (EU) and Chemical Diffusion module (CD).



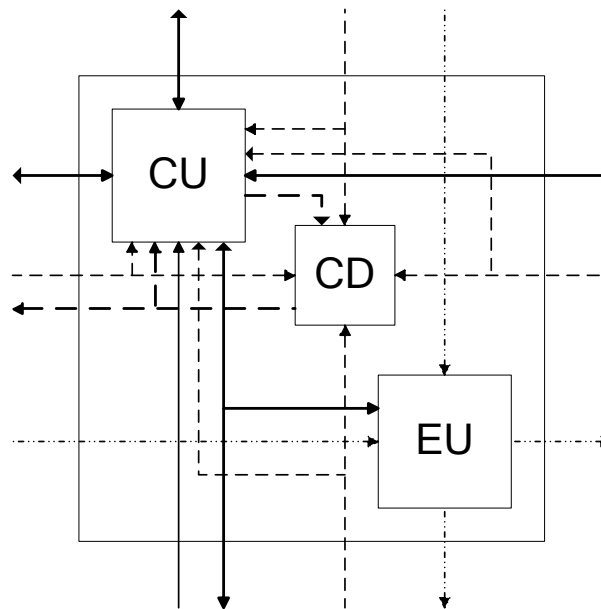
LEGENDS:

CU	Control Unit	EU	Execution Unit
- - - - ->	EU Function Selection	← - - - -	States & Chemical Signals
—	Cell border	- - - - ->	Executing Signals

Figure 5-1 Digital Organism Structure

¹⁷ Chemical Diffusion modules (CD) is omitted in this graph due to limited space and all chemical signals in this graph refers to diffused ones.

The Control Unit (CU) has a States Register (SR in Figure 5-3), which stores the internal states of the cell, including the cell state (type) and chemicals. Each CU connects to its 4 immediate neighbors (shown in Figure 5-1) and a Next States & Chemical Generator (NSCG in Figure 5-3) determines its own next state/chemicals according to the current states and chemicals of the neighbors, its own state and its own chemical (illustrated in Figure 5-2). The NSCG contains two components as shown in Figure 5-4: Next States Generator (NSG) and Next Chemical Generator (NCG), both of which are built from combinational circuits which are evolved. The CU is analogous to the nuclear region in the aspect that it stores the information and instructs the function of the other region (refer to Table 5-2).



LEGENDS:

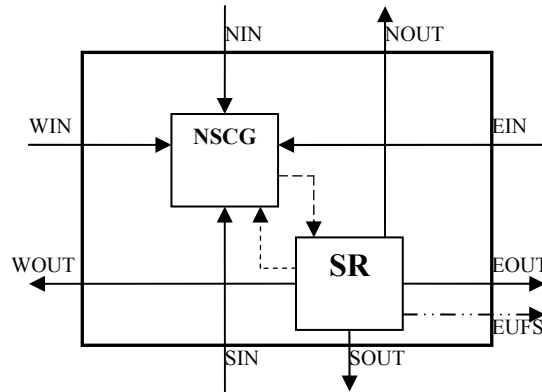
CU	Control Unit	CD	Chemical Diffusion Module
EU	Execution Unit	↔	State Signal
- - →	Chemical Signal→	Executing Signal

Figure 5-2 Digital Cell Structure¹⁸

The Execution Unit (EU) is the circuit incorporated to do the real calculation of the target functionality. The inputs to each EU come from its immediate west

¹⁸ The lines in bold denote the self signals (including chemical and state) of this cell.

and north neighbors, and the state of this cell¹⁹ (refer to Figure 5-2). Every EU also propagates its output (Executing Signals) to its immediate south and east neighbors (Figure 5-1). The Execution Unit Core (EUC) is the evolvable core logic circuit, which determines how to process the input signals in the EU. EUC will be obtained through evolution. The EU component in this model can be considered as cytoplasm in biological cells which carries out the real task of the cell.



LEGENDS:




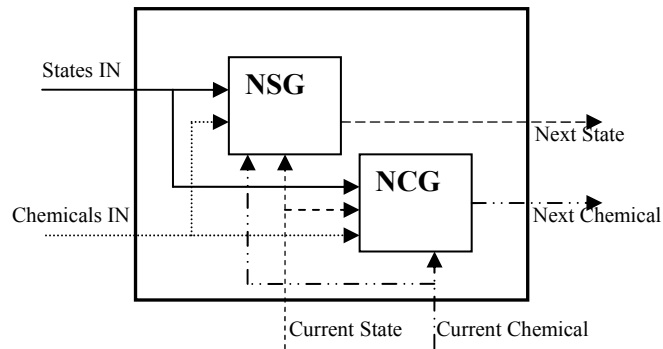
NSCG	Next States & Chemical Generator	SR	Cell States & Chemical Register
	States & Chemical Signals from neighbours		States & Chemical Signals of this cell
	Intra-cell Signals	EUFS	EU Function Selection

Figure 5-3 Control Unit Structure

At present only combinational applications are considered, hence the EUs are purely combinational circuits. (It would be an easy extension to consider sequential systems, as there are already registers in this model, which will be investigated in Chapter 9.) The state signal is 2-bit wide, while the width of Executing Signal is 3-bit.²⁰ When a cell is of type 0 (dead or stem cell), the EU in that cell will simply propagate its west (left) inputs to its south and east neighbours, otherwise this is a living cell), and the EU will execute and propagate its calculated output to the south (below) and east (right).

¹⁹ The state is stored in SR in the CU.

²⁰ Unless otherwise stated, this is used in all the following sample application.

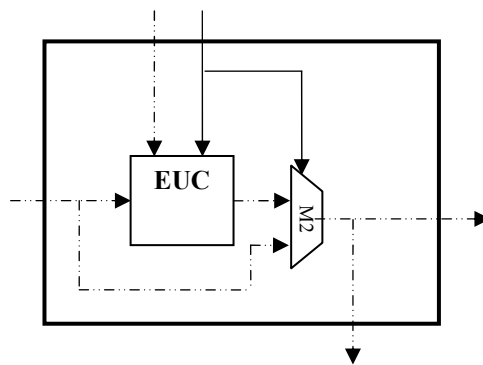


LEGENDS:

NSG	Next States Generator	NCG	Next Chemical Generator
—→	States Signals from neighbours→	Chemical Signals from neighbours
- - - - -→	Chemical of this cell	- . - . -→	State of this cell

Figure 5-4 NSCG Structure

Both the internal core logical structures of EU (EUC) and CU (including NSG and NCG) are evolved. As a result, a genotype encodes the EU and CU internal structures. The representation of the internal structure of EU and CU will be described in section 5.7.



LEGENDS:

EUC	Execution Unit Core→	Executing Signals
▤	Multiplexer	- . - . -→	EU Function Selection (The state of this cell)
—	EU border		

Figure 5-5 Execution Unit Structure

The similarity of components between biological cells and digital cells is summarised in Table 5-2.

Biological Cell	Digital Cell	Common Functions
Nuclear region	Control Unit	Saving the information
		instructing the function of the other region
cytoplasm	Execution Unit	Conducting the real functions

Table 5-2 Comparison of Biological Cell with Digital Cell

The Chemical Diffusion module (CD) mimics aspects of the real environment where biological cells inhabit. In principle, the CD module should reside outside of the cells. However, this design decision makes it more convenient practically, so it is merged into the cell internal structure. More details on CD module will be presented in next section.

5.4 Chemical Diffusion

As discussed in section 5.1, the chemical signal is introduced to transmit information between cells. Another function of the chemical is to serve as a resource²¹ which is required for a dead cell to transform to a living one.

Previous experiments [3] and [12] suggest that chemicals are indispensable in order to achieve a robust solution: without chemicals, evolved individuals have degraded stability and much lower fitness. The chemical diffusion regulation is the key mechanism which makes it such a significant aspect of this model: cells have a means to send long-distance messages.

The chemical diffusion rule employed in this work is similar to that in [3], except that there are only 4 immediate neighbors in this case as described in section 5.1. As a result the rule is:

$$(C_{ij})_{i+1} = \frac{1}{2}(C_{ij})_t + \frac{1}{8} \sum_{(k,l) \in N} (C_{kl})_t \quad (1)$$

Let N denote all the immediate neighbors of a cell at (i, j) with neighboring position (k, l) , the chemical at this position at the next time step is given by (1). The meaning of this equation is that each cell retains half of its previous chemical and distributes the other half equally to its four adjacent cells and receives the diffused chemical from them. It is evident the rule makes sure that

²¹ This resource can be treated as a kind of energy.

chemicals are conserved (apart from the unavoidable loss when the level falls below one) in the diffusion procedure.

Calculating the diffused chemical in each grow step based on the chemicals from the four immediate neighbors and the cell's own chemical value is the main task of the Chemical Diffusion module (CD) (in Figure 5-2). The CD also propagates the calculated value to its four adjacent cells.

5.5 Digital Organism Growth

Given a genotype, the inner-structure of the cells is determined and a zygote (which is located at the centre of an 'artificial environment' with x rows and y columns of cells) can be initiated and start to "replicate" (grow). The position of the zygote was selected to speed up the growth: it takes least time for the digital organism to "cover" the entire area if the zygote is arranged in the centre. The inputs to the cells on the border of this environment are fixed to 0 (see section 5.6 for more details). Without chemicals no cells can live. This means that initially some chemicals must be injected at the position of the zygote to trigger the start of the development.

Given a genotype, the growth procedure is demonstrated in Figure 5-6 and can be described as follows:

1. Initialize chemical and the zygote;
2. Chemical diffusion;
3. All cells update their state simultaneously. For each cell, if the current position has chemical and itself is alive, go to 5; otherwise go to 4;
4. Change the state of the cell to 0 if it is not so already and skip this cell (go to 3)
5. The program in current cell is executed to generate next time chemical and state based on current states and chemicals;
6. If next state generated is alive, go to 7, otherwise go to 3;
7. Overwrite chemical at this position with its own generated chemical, go to 3;
8. After all cells are updated once, one growth step is completed; if stop criteria is not satisfied (such as how many steps to grow), go to 2, otherwise go to the end;

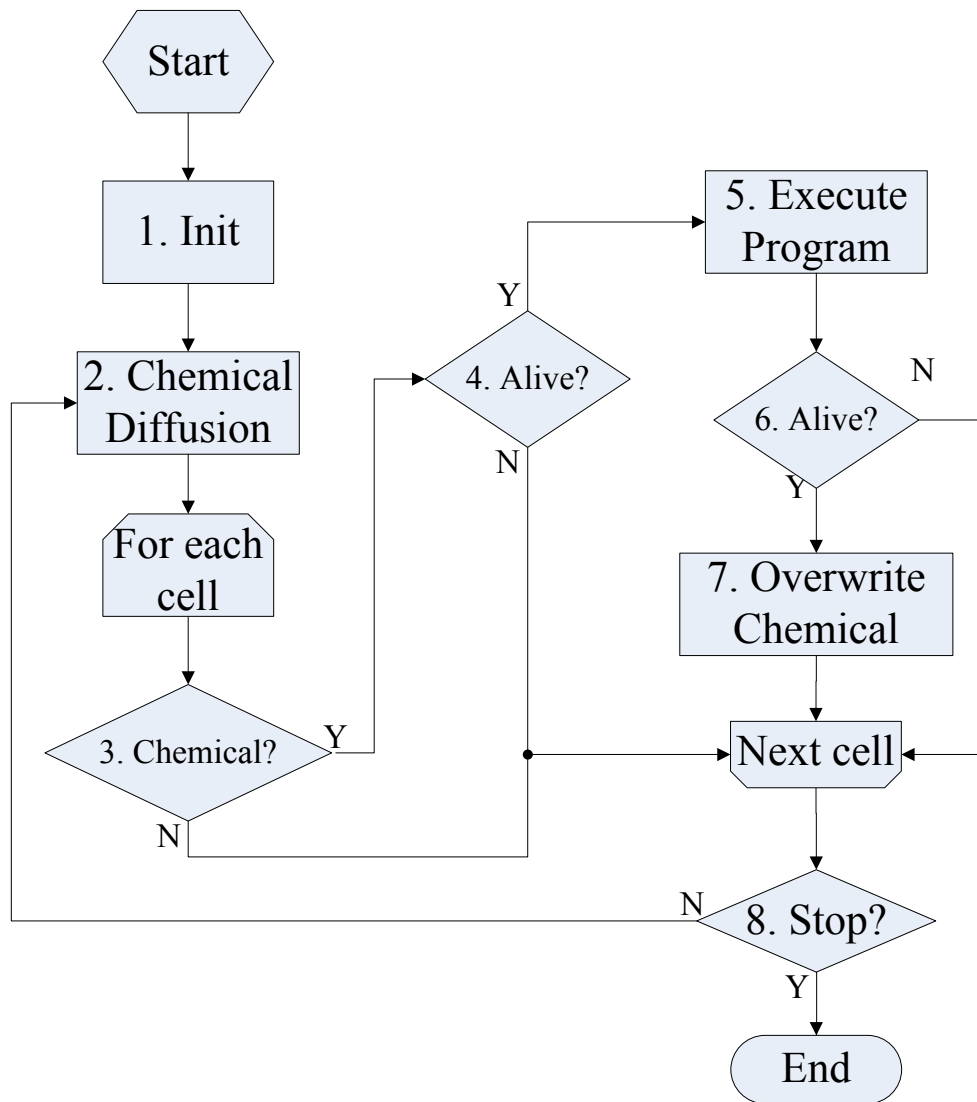


Figure 5-6 Flow chart of the digital organism growth procedure

This growth procedure ensures that each cell calculates its own next chemical/state based on locally available information when some criterions are satisfied: there is some chemical in this position and at least one of its four neighbors is alive. Thus all the dead cells in the initial step can be classified as stem cells which have the potential capability to transform into any kinds of cells.

It should be noted that, the growth procedure of the digital organism is slightly modified in Chapter 9, where the step 3 is changed: for each cell, if the current position has chemical and one of the cells' four immediate neighbors are alive or itself is alive, go to 5; otherwise go to 4. Thus even if a cell is dead, the program in that cell will still be executed in the work done in Chapter 9.

5.6 Boundary Condition

The available amount of cells in a certain implementation of this model is obviously not infinite, so when cells reach the boundaries of the matrix, no further growth is possible. For those cells adjacent to one or more borders of the digital organism, boundary condition has to be defined.

In most cases, if not stated otherwise, 0 is the value of boundary condition in this work. For instance, for a cell in the left border of the available matrix, the chemical inputs and state input from its left is always 0: as no cell is present to its left, thus neither chemical or state available.

One of the exceptions to this rule is when incorporating of Execution Unit: some of the boundary condition will be changed so that they reflect the current inputs from outside of the system (external environment).

5.7 Cartesian GP and Genotype Representation

Cartesian Genetic Programming [4] was selected as the genotype representation as in previous work carried out by Dr. Miller[3]. A program is expressed as an indexed graph which is encoded in a linear string of integers. So the genotype just contains a list of node connections and functions.

One instance of a function with 3 inputs is called a molecule. A digital circuit is encoded as n rows, m columns molecules: each molecule's input can only connect to the outputs of those molecules in its left columns, no connection within the same column is allowed. These regulations ensure the feed-forward nature of the circuit. This characteristic guarantees only combinational circuits will be evolved.

The molecules in the left most column can only receive data from the inputs to the system. A molecule's output can connect to inputs of molecules no further than p columns away on the right. So p is termed "Back Level Depth".

As 32-bit (which is the data width of IA32 series CPU) is not practical to be implemented in hardware (e.g. FPGA), a smaller data width is chosen. w is used as the data width for a given circuit.

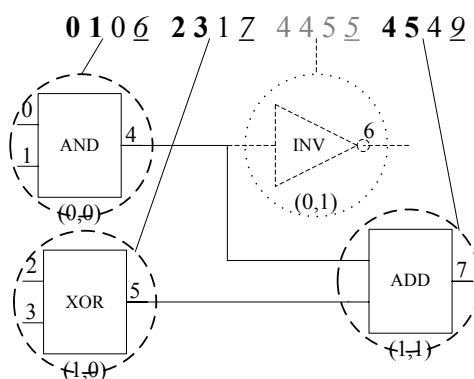


Figure 5-7 A sample digital circuit

A digital circuit is encoded as $n \times m$ molecules, with data width w , and back level depth p . From now on, this is abbreviated as “the setting for a gene is $n \times m @ w - p$ ” (In this work, one gene encodes one circuit). In order to encode EUC, NSG and NCG in the proposed model, 3 separate circuits are needed (although they are connected by some supporting logics). It was suggested that, with a one row of molecule ($m=1$) and back level depth equals to number of molecules ($p=n$), it can promote more reuse of a collection of nodes, when unrestricted connectivity of molecules and inputs can be achieved. Thus, in this work, m is always 1 while p is always identical as n .

A sample circuit with gene setting $2 \times 2 @ 8 - 1$ is shown in Figure 5-7. This circuit which is composed of a maximum of 4 molecules has 4 inputs and one output. The underlined integers in italics represent functions: INV, AND, XOR and ADD are function 5, 6, 7, 9 respectively. The other 3 integers in each group encode the inputs to this molecule. This circuit has a maximum of 6 inputs, connecting to the two molecules in column 1 (left two), although in this sample circuit only 4 inputs are actually utilized. The output is 7, so the INV gate (drawn in dotted line) encoded by the 4 grey digits is inactivated (this INV gate is called an intron in the genotype). Bold integers are actually used inputs to the molecules and the remaining digits are just ignored. It should be noted that, to encode one molecule, 4 integers are used in this example: the first 3 integers represent the input to this gate (molecule), while the last one denotes the type of the gate. However, as only two or one input logic gates are used in this example, so all the third integers are ignored.

Recently, two variation of tradition CGP are emerging [61] [62]. Oltean etc. proposed “Multi Expression Programming (MEP)” [61] in which the output nodes from the circuit were not fixed as in the case of conventional CGP, and they allowed an algorithm to determine which nodes should be selected as the desired outputs. On the other hand, a partition approach for CGP was conceived and verified by Shanthi and his colleagues [62]. They spit the truth table of the desired function into several partitions and evolved solution separately for each of them. Finally, they merged the sub-solution into one using their compaction algorithm. However, due to complexity concern, this work will utilize the original CGP.

5.8 Summary

In this chapter, the real biological cell structure is transformed to electronic context and the details of the proposed biological development model for digital systems are presented.

The Control Unit (CU) and the Execution Unit (EU) are the two primary components of this model, while the latter one is one of the novel contributions of this model, which brings functionality to a developmenal system, rather than purely pattern formation capacities. Their internal logic is determined through evolution. The inter-cell connection pattern is fixed to simplify the implementation. Another indispensable component of this model is the Chemical Diffusion module (CD) where the spreading of chemicals is implemented. The growth procedure of the digital organism is also illustrated. Cartesian Genetic Programming (CGP) was chosen as representation of the evolvable circuits.

In the following chapter, the model will be applied to a pattern formation software simulation application to examine its feasibility and flexibility, and to study its characteristics.

Chapter 6

Software Simulation of a Pattern Problem

Due to the flexibility and easy-to-debug peculiarity of software implementation compared to hardware counterpart, a software simulation was chosen as the first means to verify the applicability and feasibility of the model proposed in the previous chapter.

To start with, a pattern formation problem was selected (in this case, a French flag [3]) as the target application the model would be applied to, so that the characteristics of this model can be compared to what was proposed in [3].

This chapter is organized as follows: first, the French Flag problem is defined in section 6.1; next, a evolution algorithm tailored for FPGA is presented which is employed in this work; then the software framework utilized will be illustrated; the remainder of this chapter will detail the experiments carried out, the outcomes and analysis of the results will be presented.

6.1 A Pattern Formation Problem: French Flag

Cell differentiation, as discussed in section 3.1.4, is the key procedure which leads to all kinds of function and shapes of cells merging from a single zygote with identical inherited information. One of the mechanisms commonly found in biological cells to achieve that is position dependent differentiation. Concentration of a chemical (protein or enzyme) provides a gradient which can guide the cells to transform into different types.

An analogy of a French flag formation problem was presented in [194] by Lewis Wolpert to demonstrate how chemical gradient can act as positional information: let's assume a high concentration of chemical diffuses from one side (left side) of the flag to the other, then the chemical level in a given position in the flag is a measurement of how far away from the left border of the flag. Each cell can compare the current chemical level with given thresholds to determine its colour pattern.

French flag problem is widely used to demonstrate self-organizing behaviours of cellular systems. Due to the simplicity and intuitiveness of this problem, it was selected as the first application for this developmental model to solve.

Let state 0 denote a dead cell and the EU will simply propagate its west (left) inputs to its south and east neighbours, while state 1, 2 and 3 represent blue, white and red cells respectively, all of which are living cells, so the EU will execute and propagate their calculated outputs to the south and east. The intended perfect final state pattern of an $n \times n$ digital organism is demonstrated in Figure 6-1, which is a square French flag.

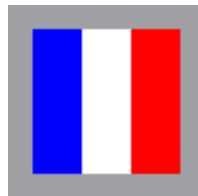


Figure 6-1 Perfect Cell State Pattern for French Flag problem

The fitness for a given genotype is the number of cells which exhibit correct states compared with a French flag at growth step s or the total number of cells exhibiting the correct states from step $s1$ to $s2$.

6.2 Multi-Breed Evolution Algorithm

Delon Levi conceived a hardware friendly EA, called Hereboy [5] and it was used in hardware evolution successfully. To some extent the work of the thesis was not addressed at finding the best (if it exists) EA for hardware, thus it was chosen as the basis for the EA in this work.

HereBoy is a combination of the characteristics from both Genetic Algorithms and Simulated Annealing. The binary chromosome (a string of 1s and 0s), one of the most important features of Genetic Algorithm, is utilized as the data structure in HereBoy, for in principle this kind of chromosome can be mapped to any problem domain. The population only contains one individual, and mutation is the only variation operator employed, which are the cases in Simulated Annealing. [5]

In each step of iteration, the chromosome is mutated by flipping bits and then evaluated. If the mutation leads to a better individual than its parent, the offspring will be kept; otherwise the algorithm discards this mutation based on a possibility test. That means sometimes a worse chromosome will be retained. This mechanism allows the system to search for better solutions. [5]

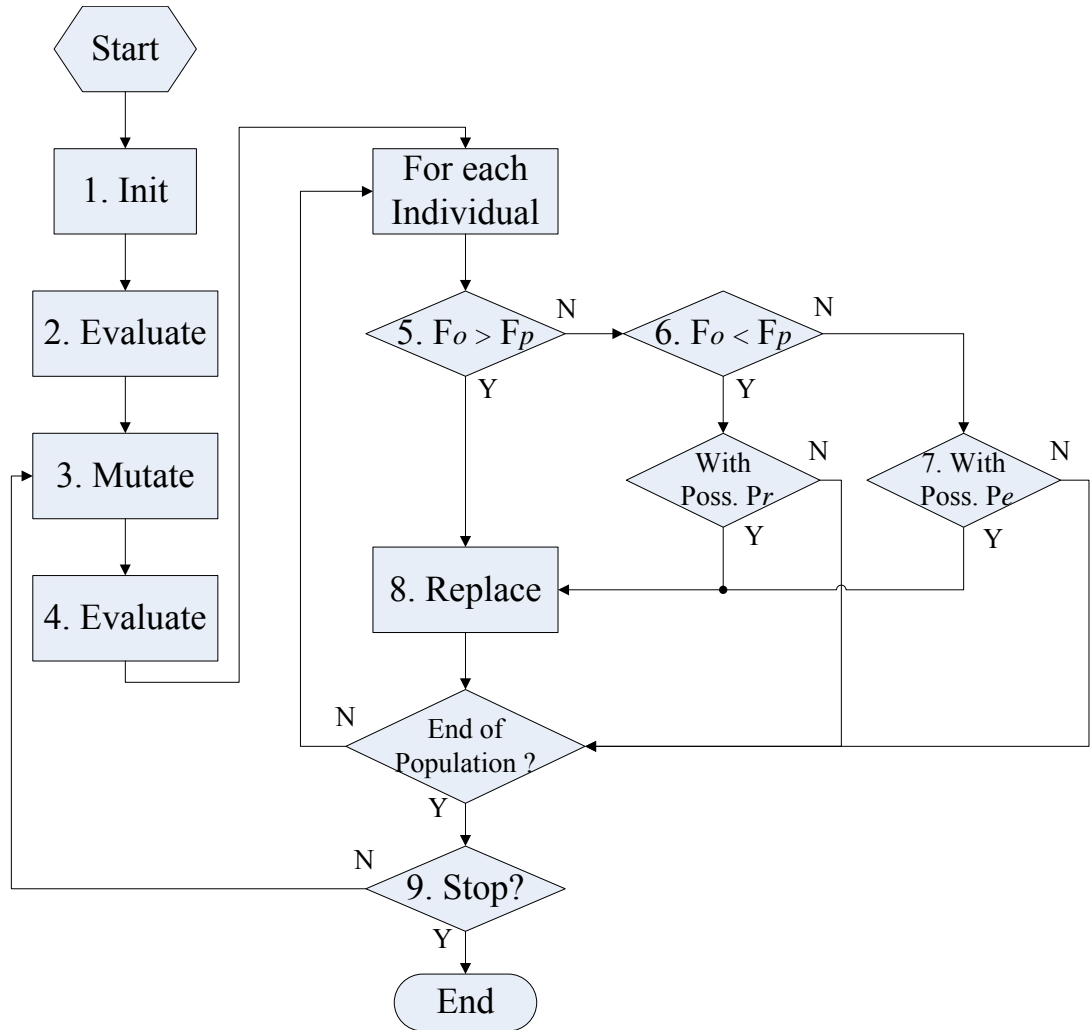


Figure 6-2 Flow Diagram of the Evolution Algorithm Proposed

It may be argued that HereBoy Algorithm is the same as the random mutation GA or the 1+1-ES. The most significant variation of HereBoy is that it honours a better individual most of the time, but does not favour a better one all the time: inferior offspring has a limited opportunity to overwrite its superior parent individual.

Inspired by this algorithm, an EA Modified from HereBoy as follows was conceived, which is named multi-breed evolutionary algorithm by the author:

1. Randomly initiate n individuals;
2. Evaluate all of them;
3. Mutate every individual once to generate n offspring; (Mutation rate p_m is fitness related, described below)
4. Evaluate all offspring;
5. If an offspring (F_o) is better than its parent (F_p), go to 8
6. Else if an offspring is worse than its parent, the offspring has a probability p_r to jump to 8;
7. Else (when an offspring's fitness is the same as its parent), the offspring has a constant probability p_e (which is determined as an input parameter beforehand) to go to 8;
8. Replace the parent with this offspring;
9. Unless stopping criterion reached return to 3.

The flow chart of this algorithm is illustrated in Figure 6-2. The main difference from the original HereBoy is that a population with more than one individual is possible in the proposed algorithm: HereBoy is a special case of this algorithm which has only one individual in its population.

It was suggested in several research that the evolution would benefit from adaptive mutation rate [5][135][136][137][138][139][140][141][142]. Adaptive mutation rate has also been proved to be efficient for hardware evolution [6]: in this work a mobile robot can adapt to unpredictable environments with the help of an evolution algorithm which employs a mutation rate defined according to the normalized fitness. Based on these findings, an adaptive mutation rate p_m was employed in this work, which is defined as:

$$p_m = \begin{cases} p_{\min} & (p_c < p_{\min}) \\ p_c & (p_{\min} < p_c < p_{\max}) \\ p_{\max} & (p_c > p_{\max}) \end{cases} \quad (2)$$

p_c is calculated based on the individual's current fitness f and the maximum fitness f_{\max} , given as:

$$p_c = p'_m \left(1 - \frac{f}{f_{\max}}\right) \quad (3)$$

p_{\min} , p_{\max} and p'_m are pre-defined. In general, p_{\min} multiplied by the number of total molecules should be greater than or equal to 1 and p_{\max} should be equal to or less than 0.5. p_{\max} should not be too high, otherwise it is of no difference from generating the whole genotype randomly from scratch.

p_m starts at high (normally p_{\max}) when the evolution begins, and declines to p_{\min} as the process converges on the final solution. This scheme allows the algorithm to focus on searching for generally good solutions at beginning and then fine tunes them to evolve the best one.

Adaptive mutation rate can be applied to a problem straightforward if the maximum fitness is known. When that is not the case, a relatively good fitness can be specified as the f_{\max} . and it can be stopped evolution if an individual is found which is of this fitness or better than that. It is always possible to evolve from an obtained individual to search for higher fitness solutions when f_{\max} can be further increased.

The probability p_r of a worse offspring replacing its parent is governed by a similar rule:

$$p_r = p'_r \left(1 - \frac{f}{f_{\max}}\right) \quad (4)$$

p'_r is another input parameter, called the *principle maximum mutation rate*. The other part of the product which generates the p_r is a fraction which will reduce from 1 to 0 as the evolution converges. So p_r decreases as the fitness of individual approaches the maximum.

p_r is introduced to aid the avoidance of local fitness maxima. With this mechanism, the algorithm can search for better opportunities in its surrounding area when gets trapped in a local maxima.

In the evaluation step, an optimization to improve performance is introduced: since calculating fitness is quite expensive in terms of time it takes, if the mutation does not change the used molecule (but those not connected in the CGP

encoded circuit, called introns), then the fitness does not need to be recalculated. This mechanism is called *fitness caching* in this work.

6.3 EO Evolutionary Computation Framework

As mentioned at the beginning of this chapter, the French flag problem was addressed as the first application via pure software simulation to verify the model proposed in Chapter 5. In order to speed up the development of the simulation software and make full use of existing codes, it was decided that this program should be written based upon one of the existing evolutionary computation frameworks.

Another design philosophy of the intended simulation software is to be as reusable and generic as possible, to obtain extensibility and keep the maintenance cost low (in terms of the time required to modify, improve or debug existing program).

There are two widely deployed libraries public available as open source projects. One is called GALib [9] and the other is EO [8] library. The first one is extensively employed in academic areas and is rather mature compared with the latter one, although its constrained flexibility breaches its original design philosophy. On the other hand, the latter one is relatively young, but it offers the possibility of more versatile approaches based on its fully object-oriented C++ template mechanism.

These two libraries were compared by E. Lutton, P. Collet, J. Louchet in [10]. Although the codes generated by EAsy Specification of Evolutionary Algorithms (EASEA) [11] were used to compare the two candidate libraries, the generated source codes is of similar quality as written by human directly, thus the comparison should be fair and similar to conditions in real use cases. It was concluded that EO tends to be more accurate (possibly caused by different random number generation scheme), and more efficient on tournaments, although slower on Roulette Wheels and much slower in genome manipulation compared to GALib. [10]

Since a new algorithm would be implemented in this project based on one of the frameworks, the flexibility is a significant characteristic. Consequently, EO

was selected as the base library. The drawbacks of EO are not such big hurdles, as tournaments are better than Roulette Wheels, and the genome representation would be by no means complex.

More details about EO framework is given in Appendix E.

In addition to writing CGP supporting code, which is the representation employed in this work, most of the components were customized to implement the algorithm described in Chapter 5.

- The selection of the Evolution Algorithm proposed is completely straightforward: every individual has equal opportunity to reproduce its offspring.
- A customized replacement component was developed to deploy the mechanism (including steps 5, 6, 7) demonstrated in Figure 6-2.
- No built-in adaptive mutation rate is present in EO framework, so a generic templates-based adaptive mutation operator was implemented.
- In order to output statistics of current population and save them in a given interval, a new Checkpointing class was written.

All the representation dependent components were implemented, such as the evaluation part, where the CGP decoding and the execution of digital circuits that encoded by CGP representation. The entire development support of the digital organism was also implemented which gives the cells the capability to grow and differentiation.

6.4 Experiment

A 6x6-cell sized French flag was the intended final pattern, developing in a 6x6 world.

Although the intended French flag itself has nothing meaningful to do with any useful functionality, this first experiment was intended to verify the model, so the French flag was chosen as an ideal experimental “application”.

2 bits were used to represent the states of cells. 0 represented a stem (dead) cell without any colours (grey in the pictures); 1, 2, 3 were blue, white and red cells respectively.

Chapter 6 Software Simulation of a Pattern Problem

One 8-bit wide chemical was used. The gene setting for sub-circuit NCG and NSG are $1 \times 20 @ 8-20$ and $1 \times 20 @ 2-20$ respectively. As there is no functionality to perform, the EU is omitted in this experiment.

p_{min} , p_{max} , p'_m and p'_r were defined as $1/mols$, 0.5 , 0.5 and $1 \times E-5$ respectively. $mols$ was defined as the total available number of molecules for a sub-circuit, for example, $mols$ of NCG: $mols = 1 \times 20 = 20$. $p_{max} = 0.5$ was chosen because a mutation rate greater than 0.5 is normally considered as meaningless for it discards most result obtained via last variation. On the other hand, a mutation rate less than $1/mols$ is not considered reasonable either since it would lead to no modification at all.

Each cell in the digital organism has a coordinate (x, y) . The cell at the top left corner of the digital organism is defined as at position $(1, 1)$ and the value of x and y increase going down or moving right in the digital organism. These coordinates are not available at all to those cells in the digital organism; instead they are just employed as reference in the transient faults injection experiment and in the content of this thesis.

Initially, all the cells were dead/stem cells except the middle one at $(4, 4)$ which was a red cell (the zygote). The chemicals were all 0 except for at the position of the zygote it was 255 (the maximum chemical available).

As experiment shows that if the pattern can be maintained for 3 steps, the flag tends to be stable, so the fitness for a perfect individual was the total number of cells with correct colours from growth step 6 to step 9.

Although recently a method for estimating the generation and population size needed to evolve a solution using analytical models was presented in [41], it is only applicable to strict binary programming strings (as genome), instead of a list of integers, so it can not be consulted as a valid reference in this work. Alternatively, these parameters were determined empirically. The population size was 10. The evolution strategy would not stop unless it found a perfect solution. Another bash (shell) script wrapper called the evolution software and collected the results: once it found a perfect solution, it would restart the evolution algorithm to continue searching for another solution (until it was instructed to terminate).

In order to make full use of modern CPU's 32 bits wide data bus to accelerate the simulation procedure, multi-bit high level functions were chosen, such as ADD, AND, OR, XOR, RIGHT SHIFT (all of which are 32 bits wide operators).

6.5 Results

Some extremely robust individuals were evolved. The growth process of one of them is shown in Figure 6-3.

In each image, the left half shows the states of the organism, while on the right the chemical is displayed. The top left image in Figure 6-3 is the initial states (step 0). The remaining images, column-wise then row-wise, are pictures demonstrating step 1 to 11 respectively. (This is the convention used throughout this thesis.)

It can be seen that the image at step 8 is identical to the one at step 10, and step 9 and step 11 are exactly identical to each other (in regard to both states and chemicals). So after step 8, the organism stabilized itself in a French flag pattern.

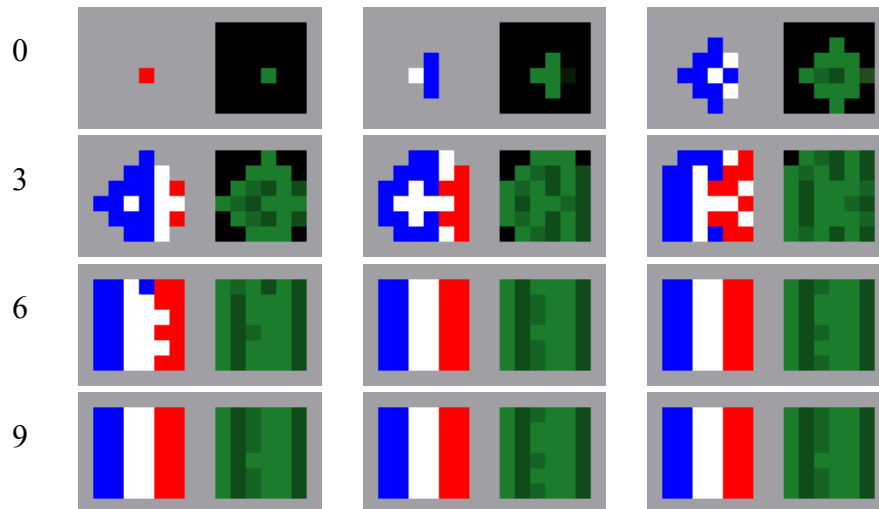


Figure 6-3 Growth of one of the best 6x6 individuals

Wiping all blue cells to dead/stem cells after step 9, this organism could recover flawlessly in 5 steps; changing all blue cells to white, it could recover correctly in 2 steps; changing all blue cells to red cells, again it could recover accurately in just 2 steps.

As the case stands, altering any whole colour area into another one, this organism can recover perfectly within 10 steps.

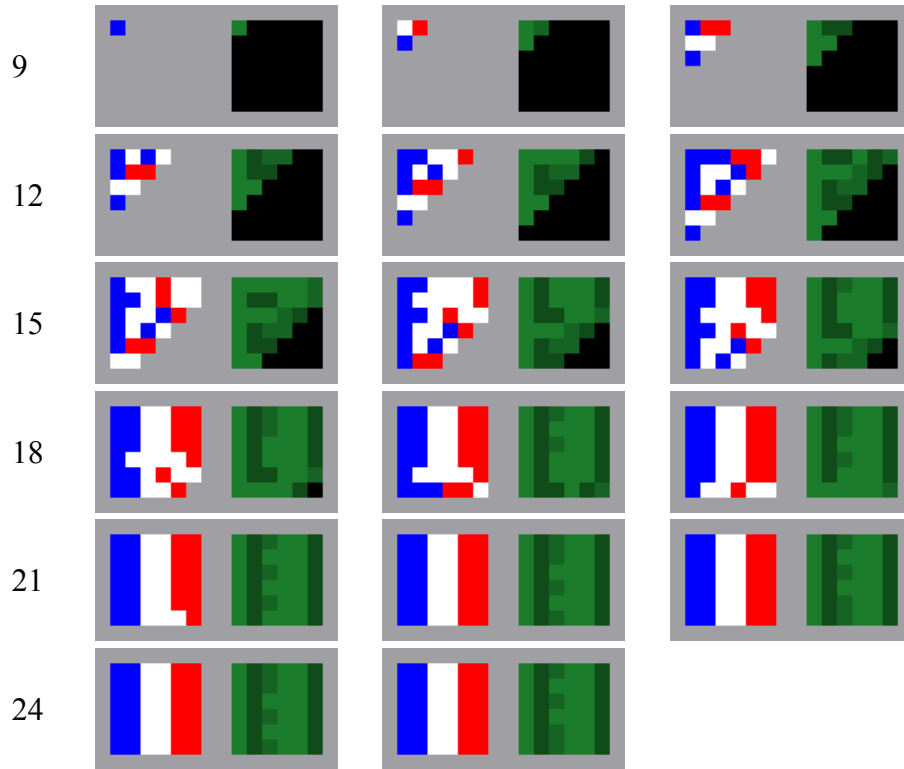


Figure 6-4 Recovery Process of the 6x6 robust individual

Furthermore, not a single combination of transient faults in the chemicals that the organism can not recover from was discovered.

Even with only one live cell left, for example one blue cell at (1, 1), and wiping all chemicals except for the position (1, 1), it could recover totally in 13 steps (As show in Figure 6-4). Top left image is at step 9, the state of the organism after transient damages (modification of cell states and chemicals). At step 22, it recovered French flag pattern, and stabilized again in terms of both states and chemicals (can be seen from step 22 – step 25).

In fact, no situation was found that this organism could not tolerate and recover completely.

So once this artificial organism reached a French flag pattern, it was stabilized even when transient faults are manually injected into the states and/or chemicals of cells.

6.6 Analysis

Various aspects of the details of this model are discussed further in this section with experiments results, including distinctions from previous work,

investigation on the cause of the robustness of the evolved solutions and what is the most utilized inputs and molecule types in the best individuals. Finally, the algorithm proposed is analysed and compared with the classical (1+1)-ES.

6.6.1 Comparison with previous research

The evolved solutions to the French flag problem are more robust with regards to exposing to transient faults than those individuals evolved in previous work by Dr. Miller described in [3] (Figure 6-8 showed the best individual evolved in Dr. Miller's paper) . This model consists of several key differences compared with Dr. Miller's development model; the most notable distinctions are summarized below:

1. No dedicated output in the cell interface to specify which direction to grow as in Dr. Miller's model: in this model, each cell decides its own next state and chemical level;
2. All the cells in the organism are of equal status: no cell can overwrite other cells output;
3. Contrary to 8 immediate neighbor cells in Dr. Miller's work, only 4 adjacent cells are employed in this model. This distinction makes this proposed model takes double time as Dr. Miller's model to cover an entire square world (see Table 6-1 for a comparison).
4. In the experiments carried out in [3], the flag is positioned in the middle of the world without touching any of the borders, while in the experiments described previously in this chapter, the final flags are the same size of the world.

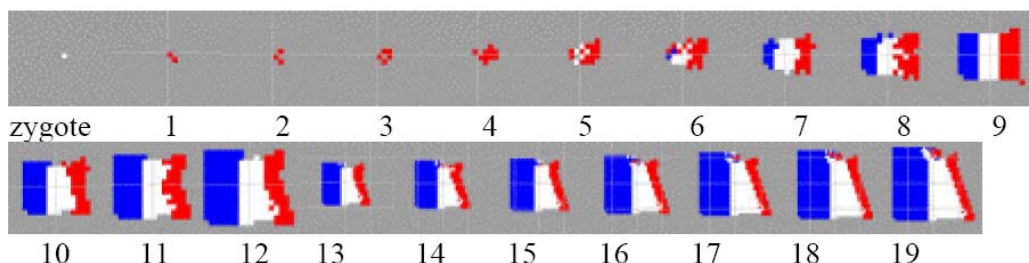


Figure 6-5 Best solution for FF formation

The model proposed herein can be considered as a cellular automation (CA), in which each cell executes some program, rather than “growing” to new positions.

6	5	4	3	4	5	6
5	4	3	2	3	4	5
4	3	2	1	2	3	4
3	2	1	0	1	2	3
4	3	2	1	2	3	4
5	4	3	2	3	4	5
6	5	4	3	4	5	6

3	3	3	3	3	3	3
3	2	2	2	2	2	3
3	2	1	1	1	2	3
3	2	1	0	1	2	3
3	2	1	1	1	2	3
3	2	2	2	2	2	3
3	3	3	3	3	3	3

Table 6-1 Growth speed comparison of this proposed model (left) with Dr. Miller's model (right)

In this example, the grid is 7x7, and the number in each cell of the tables denotes how many steps it requires to grow into this cell: It takes 6 steps for this model to cover the 7x7 grid completely (the largest number on the left table is 6), while it only takes 3 steps in Dr. Miller's work. Thus normally, the growth speed of this model is half the speed of Dr. Miller's model.

6.6.2 Robustness Analysis

Compared to previous work, this model can form more robust FF patterns: the recovery procedure from two damages are demonstrated in Figure 6-9 for Miller's evolved best FF solution: first a rectangle area in the middle of the flag is removed, and then the white and red areas are removed. We can see that, although the regenerated pattern contains all the basic features of a French flag, but it is quite different, especially considering the proportion of white and red areas compared with blue one.

The robustness of this model is considered to be contributed by the constrained development environment (the 4th distinction point mentioned above), which is the default boundary condition as discussed in section 5.6: the cells on the borders always receive 0 as chemical/state inputs from those inexistent adjacent "cells".

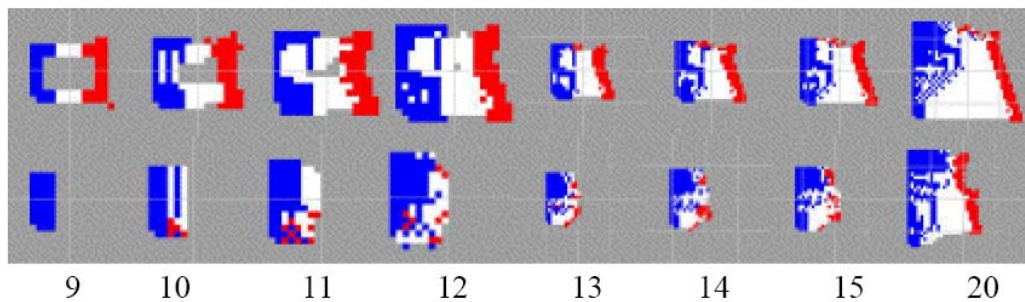


Figure 6-6 Recovery of the best FF solution

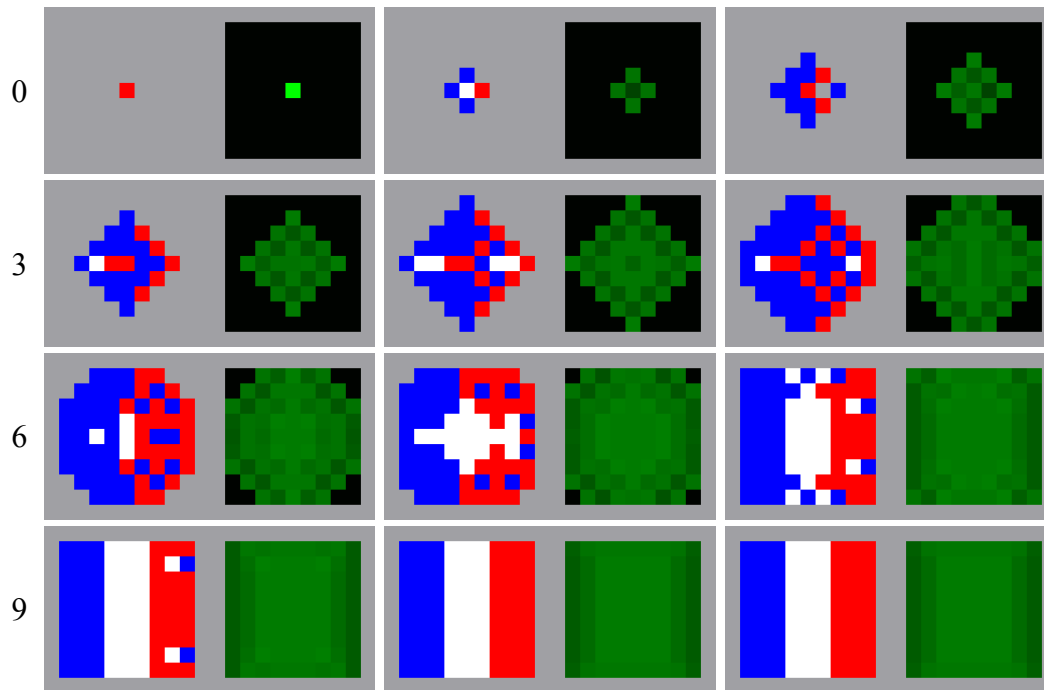


Figure 6-7 Growth of one of the robust 9x9 individuals

An experiment was setup to verify this hypothesis: in a 9x9 world, 6x6 French flag is evolved, which is in the middle of the world, without touching any borders of the world. Some evolved solutions do exhibit transient fault tolerant capability to some extent, however the degree is not in the same level: not a single individual can be found which can tolerant any combination of transient faults stated above in states and/or chemicals.

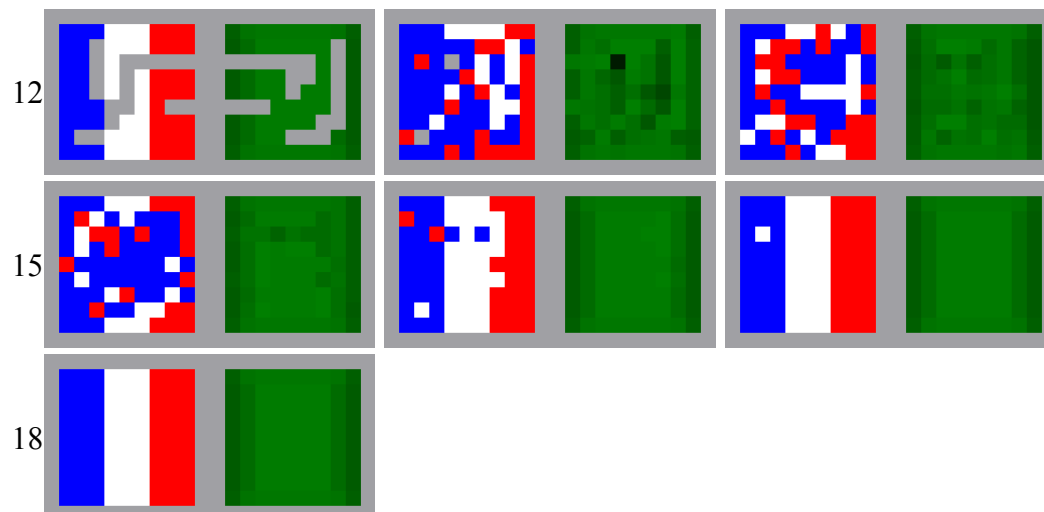


Figure 6-8 Recovery process from a random erasing of states/chemicals of the 9x9 individual

In order to be comparable to what the hardware implementation (see Chapter 7) would deploy, the setup of the original experiment was slightly modified: available molecules are restricted to 4 types of multiplexers (see section 7.3.1 for details) and 1 bit left shifters (LSHIFT1), 2 bits left shifters (LSHIFT2) and their corresponding right shifters (RSHIFT1 and RSHIFT2).

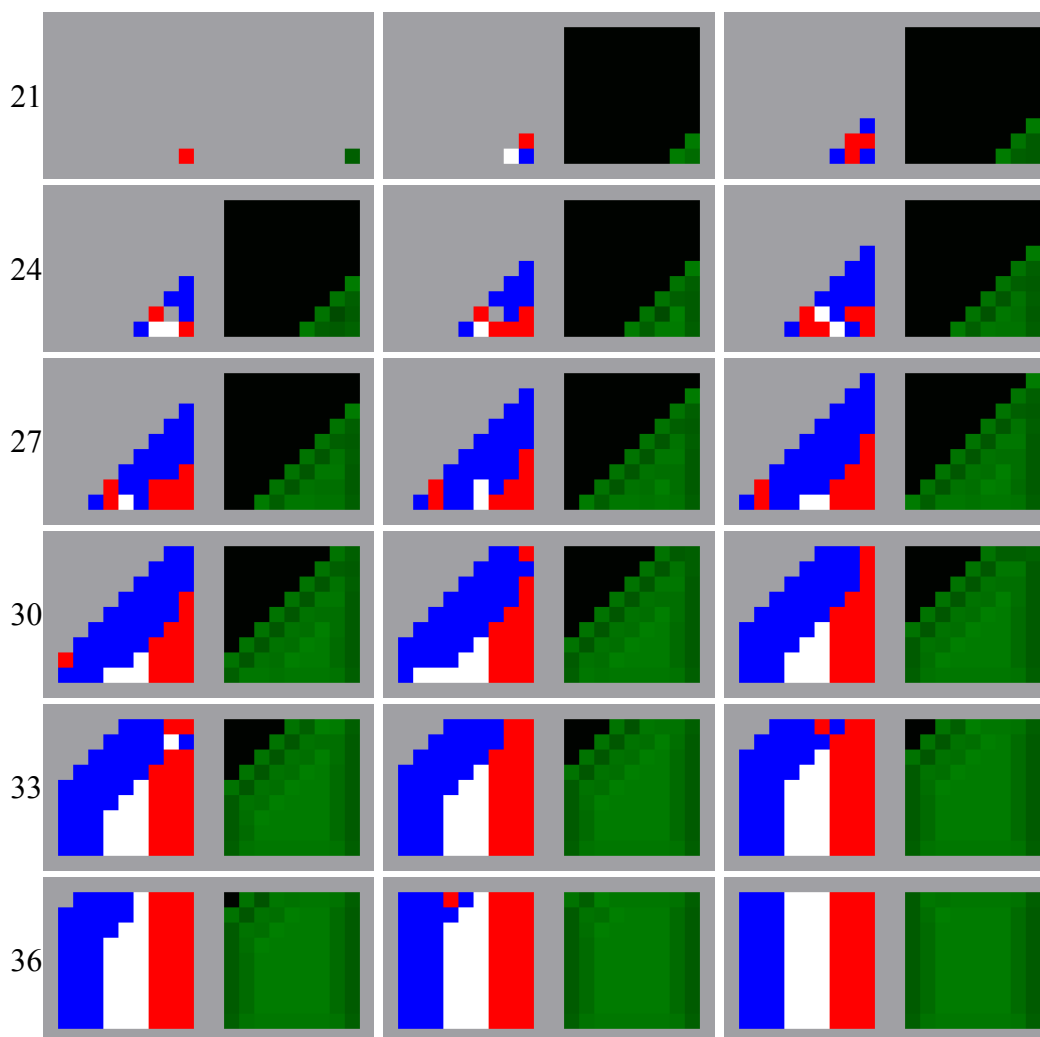


Figure 6-9 Recovery from a single cell of the 9x9 French flag

Although no shift is needed in the hardware implementation, shifts are essential in this software simulation. In the software implementation, in order to make full use of modern CPU's 32 bits wide data bus to accelerate the simulation procedure, all the functions chosen for molecules operate on all bits of a signal: taking a simple AND gate for an example, an AND gate operates on two inputs, assuming North Chemical and South Chemical, then this AND gate performs a bit-by-bit logical operation on the two input chemicals and output one signal with the same width as the input chemical signals. Without shifts, it is impossible

to derive logic from different bits of input signals: only the same bits in each signal could be combined to generate output logic.

9x9 French flags could also be evolved which exhibit perfect recovery behaviours. The growth procedure of one of them is shown in Figure 6-7. Step 10 and 11 are exactly identical with regards to both chemicals and states, so this organism stabilizes in 10 steps. Given that, all the chemicals and states in all the cells in the digital organism are in the exactly same values in both step 10 and step 11, the organism should be indefinitely remain in the same pattern (as shown in step 10) no matter how many more iterations the cell programs are executed, provided conditions don't change.

In Figure 6-8, it is demonstrated the recovery capacity of this robust 9x9 French flag from a erasing of a random combination of missing states and chemicals after the maturity of this organism. Within 7 steps, the organism recovers to the stable state (18 is identical with 10).

This 9x9 French flag can also re-grow to the correct pattern even only one cell is left (see Figure 6-9). This is not so surprising considering that it grows from a single cell in the first place.

6.6.3 Inputs and Molecules usages analysis

7 perfect 9x9 individuals and 6 best 6x6 French flags were analyzed to understand which is the more important signal, chemical or state signals. In Figure 6-10, the overall used Chemical inputs compared with used State inputs are demonstrated. As there are two genes to encode one individual (one for NSG, while the other for NCG), they are separated plotted in the figure. It is clear that more than 70% of connected inputs are of chemical signals, no matter for which gene, so chemical appears to be more important than the state signals. This can be interpreted if we consider the different roles state and chemical play in this model: state is used as a instruction for the Execution unit, while chemical is introduced to act as a signalling system to other cells nearby.

Breaking down the connected chemical inputs in each direction, Figure 6-11 can be obtained: it groups inputs according to where the inputs come from, this cell (Self) or the cell above/below/left to/right to this cell (Up/Below/Left/Right). The usage for the two genes is separated to two groups.

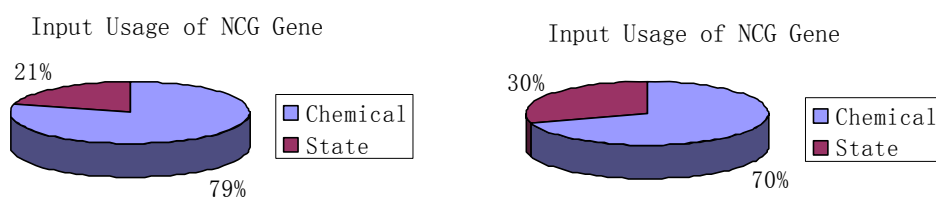


Figure 6-10 Input usage of the two Genes encoding a French flag

It is with no surprise, Left and Right chemical signals are the most deployed ones in both of the circuits (NCG and NSG): they account for about 60% inputs of both circuits. The French flag problem can be solved by one gradient in the horizontal direction, so this is the expected behaviour arising from evolution pressure.

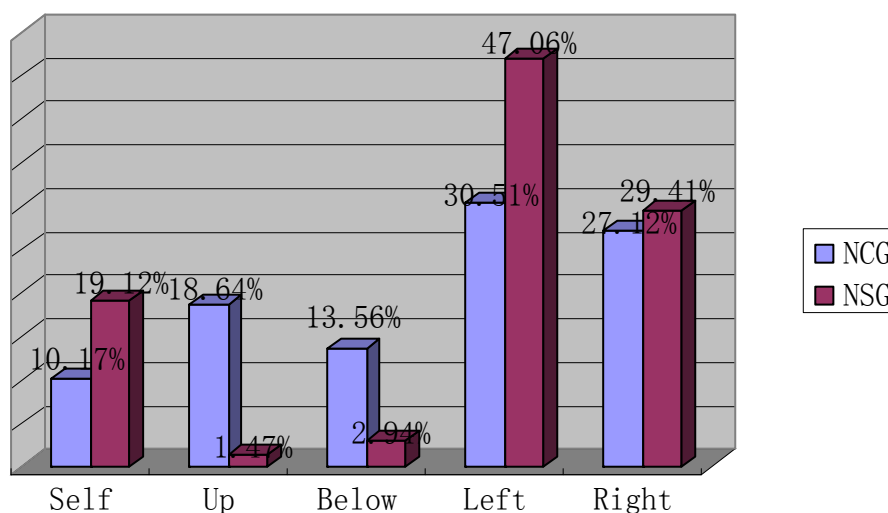


Figure 6-11 Chemical signal usage details

Considering what types of molecules the best individuals normally rely on, Figure 6-12 shows the normalized molecules usage distribution. In this figure, 9x9 sized French flag and 6x6 ones are categorized into two separate bar groups.

MUX4 is statistically the most popular molecules types in the perfect solutions. Compared with left shifters, which account for 15%, is fewer than right shifters, which in total take 30% in both sized flags. In the implementation, with PC hardware, right shifting means move more significant bit to lower bits, so

generally high significant bits in the signals are more used when calculating low significant bits, while the higher bits tend to not rely on lower bits so much.

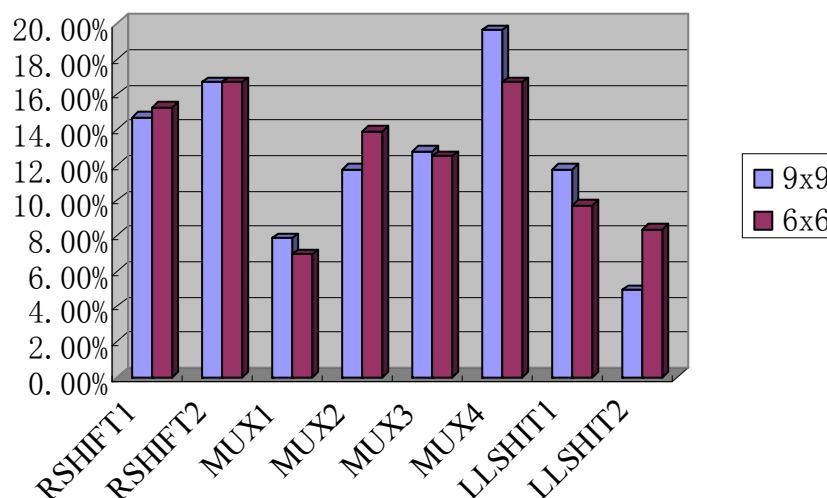


Figure 6-12 Normalized molecule usage in the best solutions

While the 6x6 individual shown in the previous section has a dynamic stable state, which refers to the fact that after maturity, the chemical map is vibrating between two states constantly, static stable state are also observed, such as the 9x9 sized French flag presented in this section.

6.7 Summary

A well known pattern formation problem, French flag, was selected as the first testing application for this proposed developmental model to solve. The problem was first defined, and a new hardware friendly evolutionary algorithm was proposed which was then deployed in this work. The setup of the software simulation and experiment carried out were described. Finally, the outputs and the analysis of the results were examined.

This software simulation demonstrated that with the help of the chemical diffusion mechanism and local chemical exchanges between adjacent cells, the biology inspired development model proposed can be utilized to achieve relatively robust digital organism under the challenge of transient faults. This fault tolerant feature is not explicitly designed into the system, but merges as an intrinsic characteristic.

Chapter 6 Software Simulation of a Pattern Problem

This desired feature is potentially capable of being deployed to bring useful functionalities to applications with intrinsic highly fault-tolerant characteristics. In the next two chapters, the Execution Unit network will be put to action: a simple digital logic circuit will be implemented with this model, not only in software simulation, but also in hardware simulation and FPGA.

Chapter 7

Implementation of Digital Logic System

The development model was demonstrated to be able to generate highly robust cell type patterns in a resource constrained digital organism. In order to have a deeper insight of what the model can achieve and bring a functionality to the developmental model, 2-bit multiplier is targeted with the help of integrated Execution Unit in each cell.

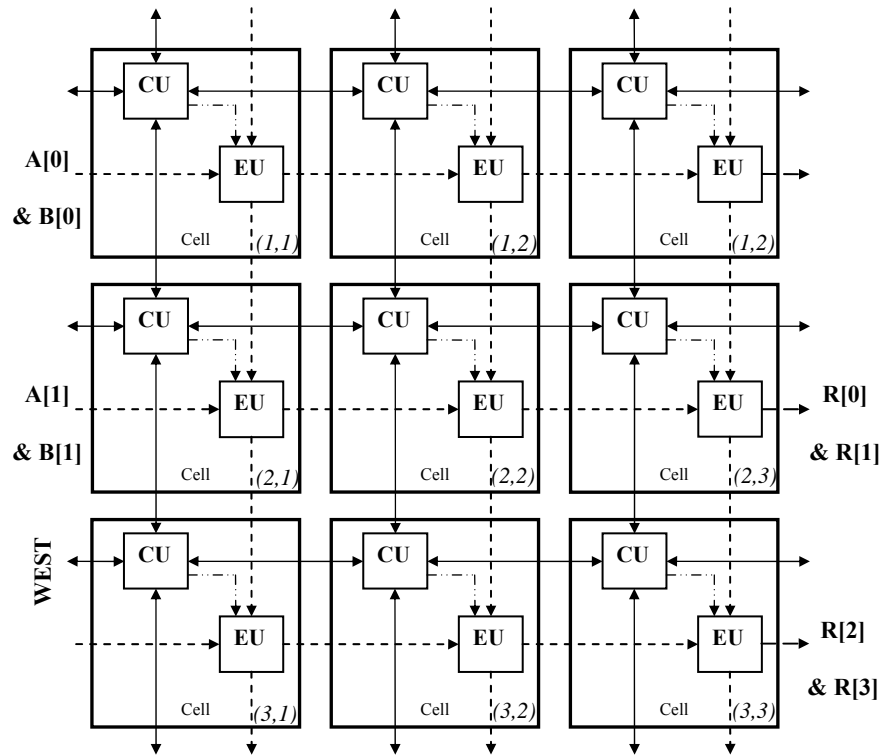
Because of being a fundamental function/component unit and its simplicity, the 2-bit multiplier was selected as an ideal test-case for this model. This chapter is arranged in 5 sections: first the structure of the organism used in this application will be described in section 7.1; the details of the EU component in the model incorporated in this application will be demonstrated in section 7.2; then the software simulation carried out will be discussed in the following section; the hardware implementation and verification follow in the last two sections.

7.1 Digital Organism Structure

The digital organism employed in this application is made up of 3x3 identical digital cells, rather than 6x6 or 9x9 cells as in the French flag problem. Shown in Figure 7-1, the primary structure of this digital organism is the same as the one deployed in previous chapter. However the EU is incorporated to accomplish the real calculation of the multiplier result. The state signal is 2-bit wide (the same as in the French flag problem), while the executing signal is 3-bit wide.

7.2 Incorporating Execution Unit

The inputs to the 2-bit multiplier (A and B) are connected to the Executing Signals of cell (1, 1) and (2, 1). All the other unconnected inputs of the digital organism are fixed to 0 (which is the normal boundary condition described in section 5.6). The output of this multiplier is driven by the output Executing Signals of cell (2, 3) and (3, 3). (see Figure 7-1)



LEGENDS:

CU	Control Unit	EU	Execution Unit
----->	EU Function Selection	↔	States (2b) & Chemicals (4b) Signals
—	Cell border	----->	Executing Signals (3-bit width)
A & B	The multiplier inputs	R	Result output of the multiplier

Figure 7-1 3x3 Cells Digital Organism²²

The EU Function Selection signal (the state of a cell) is 2-bit wide, so there are 4 available values: 0 means this EU will simply propagate its west (left) inputs to its south and east neighbours (just as in French flag experiment), otherwise this is a living cell (it can be in any type among 1, 2 or 3), and the EU will execute and propagate its calculated output to the south (below) and east (right). (See Figure 7-1)

²² In this figure, X[0] denotes the least significant bit of the signal X (can be either A, B or R)

7.3 Software Simulation

The first step of the experiments was software simulation, for the same reasons raised in Chapter 6. This part of the experiment was designed to verify that this model with proper configuration can evolve some solution which exhibits similar transient fault-tolerance capacity.

7.3.1 Evolution Strategy

In order to simplify the implementation, two phases are designed to evolve the entire digital organism:

1. To evolve the structure of the EUC and the distribution of states of the 3x3 cells. The genotype in this phase consists of two parts: the CGP part encodes the EUC structure, while the other is the states for all the 9 cells. The fitness is the correct bits of the multiplier output result: it has two 2-bit inputs, so there is totally $2^2 \times 2^2 = 16$ possible combination of inputs. Since the output of a 2-bit multiplier is 4-bit wide, $16 \times 4 = 64$ is the maximum fitness for the organism in a given step.
2. Structures of NCG and NSG will be evolved to discover a stable solution for the states distribution of cells found in the first phase. This phase is the same as the evolution process described in Chapter 6 except for some parameter values.

Hardware implementation of the digital organism can be achieved after the completion of the two phases.

In order to search for more resource-efficient individuals in preparation of the hardware implementation, once a perfect solution was found, it would receive a fitness bonus based on how many molecules it actually uses: the less the better.

It is thought that Universal-logic-module 2 (abbr. ULM.2, a module which is capable of implementing any function with $n \leq 2$ independent binary input variables is referred to as ULM.n. [13]) is preferable compared to ULM.3 or other more complex functional alternatives: ULM.2 fits the average fan-in requirement of human-design circuits most appropriately and if larger fan-in cells are deployed, wiring density/complexity and wiring channel proportion will increase considerably. [13]

It was also proposed that one particular kind of ULM.2 is the most appropriate candidate for general-purpose fundamental logic element, identified as f_L in [13]:

$$\text{Configuration } f_L, \text{ output} = y_1 y_2 + \bar{y}_1 y_3$$

This configuration, along with negation of the input variables and constant 1 and 0, can realize all 2 input functions.

Based on this theory, the available functions in a molecule are limited to 4 types of multiplexers (the same as in [3] and [12]), which are shown in Table 7-1: these algebraic expressions are all the 4 possible combination of negation of the input variables in the Configuration f_L .

Name	Algebraic Expression
MUX1(A, B, C)	$AC + B\bar{C}$
MUX2(A, B, C)	$\bar{A}C + B\bar{C}$
MUX3(A, B, C)	$AC + \bar{B}\bar{C}$
MUX4(A, B, C)	$\bar{A}C + \bar{B}\bar{C}$

Table 7-1 Available functions for Molecules

The available inputs to a molecule in the hardware implementation are quite constrained (refer to section 8.1.1). As a result, no constant 1 or 0 is available as inputs to molecules. However, 1 and 0 can be achieved directly and efficiently by MUX3(X, X, X) and MUX2(X, X, X) respectively, in which X refers to one input signal to a molecule, with no requirements on its value at all.

7.3.2 Parameters

For the first phase, population size is 20, while p_{min} , p_{max} , p'_m and p'_r are defined as 1/mols, 0.5, 2 and 1xE-6 respectively. The parameters for the other phase are basically the same as these in the experiments discussed in Chapter 6.

The structures of Next Chemical Generator (NCG), Next State Generator (NSG) and Execution Unit Core (EUC) were evolved through evolution. The gene settings for these three separate circuits are 1x6@4-6, 1x9@2-9 and 1x10@3-10 respectively.

7.3.3 Outcome of Simulation

55 individuals which implement 2 bit multiplier were evolved. The occurrence number of each state in those successfully evolved solutions is summarized in Table 7-2.

State	0	1	2	3
Occurrence	130	123	117	125

Table 7-2 Overview of states occurrence in first stage experiment

The percentage each state takes in those evolved individuals is illustrated in the Figure 7-2 pie chart: no state is dominant among the four possible state values; instead each state takes about 25%. This is understandable because no explicit evolution pressure is imposed to favor a particular one (or set) of state(s) over others.

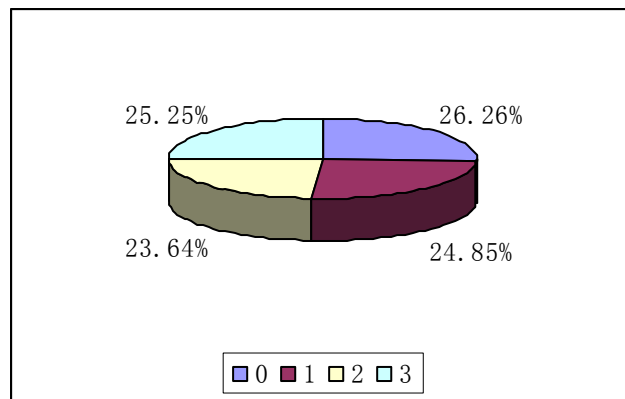


Figure 7-2 Distribution of states

One of the patterns found in the first phase is shown in Table 7-3. This pattern utilizes most available cells with a diverse and complete distribution of states, so it is chosen as the target configuration of the digital organism along with its corresponding EUC structure obtained via evolution.

0	1	3
2	3	2
3	2	3

Table 7-3 Chosen Cell State Pattern

It should be noted that, it is not necessary to choose a pattern in order to obtain a working 2-bit multiplier, however with the two-phase evolution as described in

section 7.3.1, this is required. When the hardware implementation is realized, there is no need to evolve solutions with 2 stages, so no need to specify a pattern any more.

The EUC makes use of 6 molecules out of 8 available ones and the NCG/NSG utilizes 6 out of 9 molecules. Their structure details will be depicted in next section.

7.4 Hardware Implementation of the Digital Organism

After the 2-bit multiplier was successfully evolved in software simulation, the hardware implementation phase began. In this section, the design of the FPGA realization of the model and the evolved sub-circuits will be illustrated in detail.

7.4.1 Top Level Structure

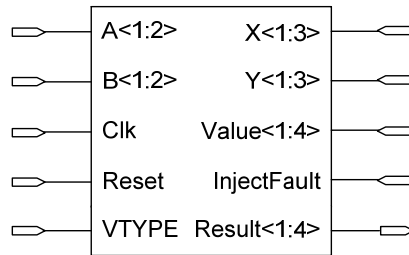


Figure 7-3 Digital Organism External Interface

The external interface of the digital organism is shown in Figure 7-3. Pin *A*, *B* and *Result* are the inputs and output of the 2-bit multiplier. *Clk* is the global clock signal; if the *Reset* pin is high, all the internal registers will be set to their initial values. All the remaining pins are dedicated to injecting transient fault(s) into the digital organism: when *InjectFault* pin is high, *Value* will be written into the chemical of cell at coordinate (*X*, *Y*) if *VTYPE* is low, otherwise the lowest 2-bit of *Value* overwrites the state of the cell. The whole organism stops its normal operation (such as growth) until *InjectFault* pin becomes to low again.

Every cell has an identical structure. Figure 7-4 demonstrates the external interface of a digital cell. Pin *InjectFault*, *VTYPE* and *Value* are connected to their global counterparts. If this cell is at the coordinate (X, Y) and *InjectFault* is active (high), the *CS* pin of this cell will be driven to high and the cell will overwrite its own chemical or state with *Value*.

A “growth step” lasts two clock cycles: at the falling edge of the first clock cycle a live cell (its state is not 0) will overwrite the existing chemical with its generated one; at the rising edge of the second clock, the chemicals diffuse according to the rule described in section 5.4. At the rising edge of the first clock cycle in the next “growth step”, the state will be updated.

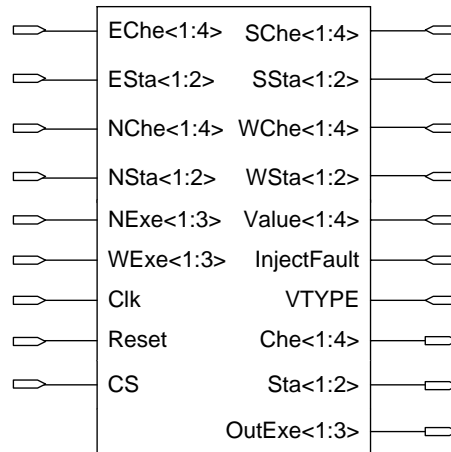


Figure 7-4 Digital Cell External Interface

7.4.2 Sub-Circuits Evolved via Software Evolution

The internal structures of EUC, NCG and NSG obtained via the evolution are given in Figure A-1, Figure A-2 and Figure A-3 (in Appendix A). All the blocks in these three figures are pure combinational sub circuits.

7.5 Hardware Simulation and Verification

The following experiment was then carried out: first, enough time was allocated to let the organism grow and mature; after its maturity, two sets of transient faults were injected: the first set composed of 4 transient errors in the chemicals of cell $(2, 1)$, $(2, 2)$, $(2, 3)$ and $(3, 3)$; the other set of faults were injected into the

Chapter 7 Implementation of Digital Logic System

states of cell (2, 1), (2, 3) and (3, 1). Every fault is to make the corresponding value 0. The interval time between the injections of the two sets of transient faults was more than sufficient for the organism to recover completely and stabilize itself again in terms of chemicals and states of the cells.

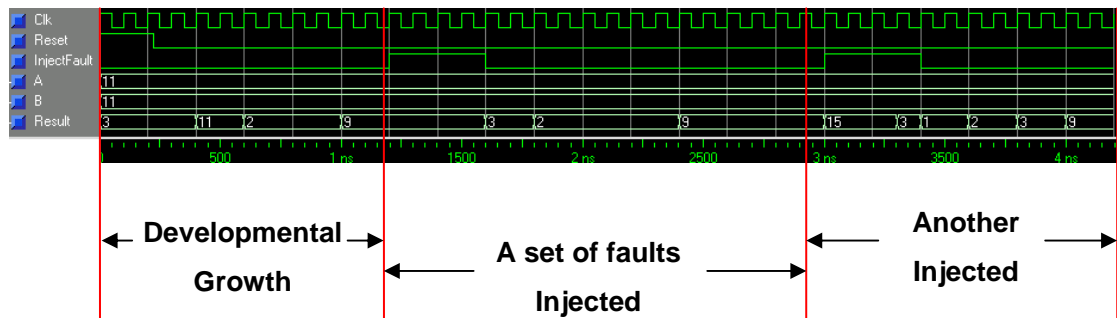


Figure 7-5 Overview of the Simulation Waveform

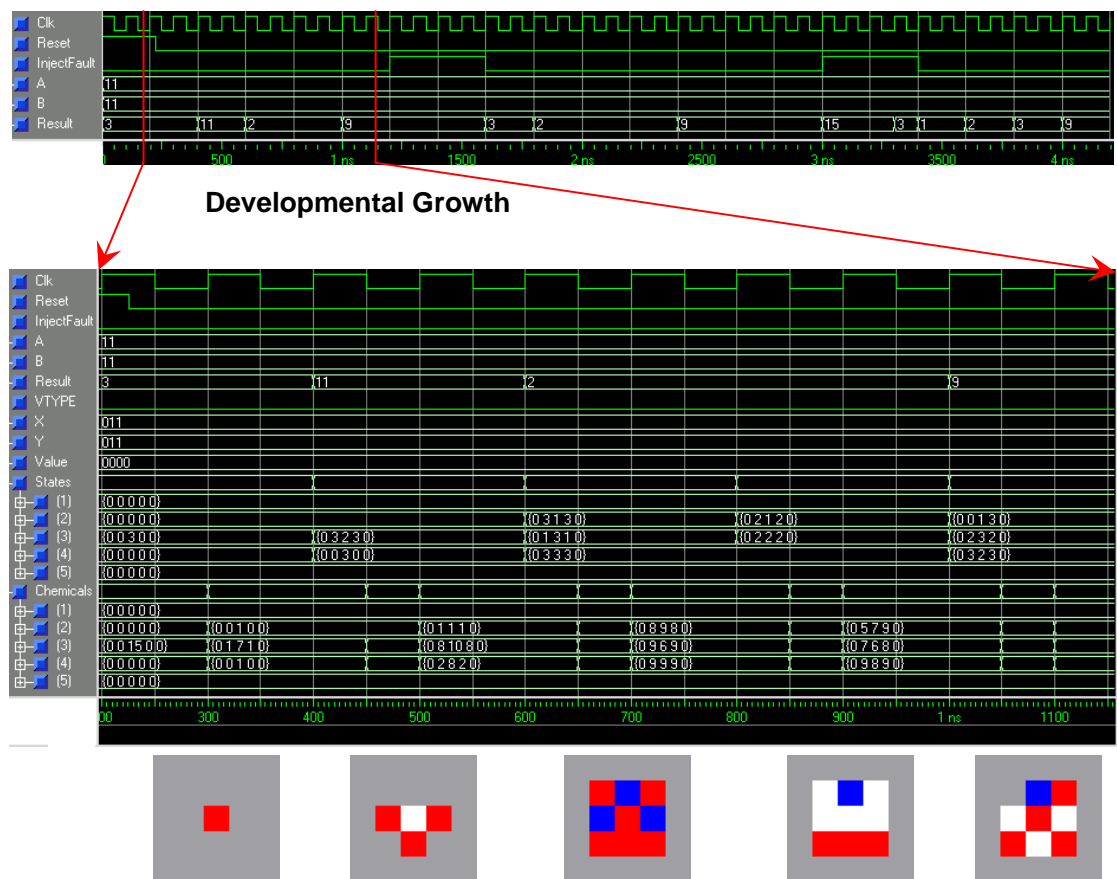


Figure 7-6 Developmental Growth Procedure²³

²³ For clarity, the states of the organism are presented in color at the bottom using the same convention as in the French flag problem.

Figure 7-5 presents the overview of the Very High Speed Integrated Circuit Hardware Description Language (VHDL) simulation waveform²⁴. In general, 3 main stages can be distinguished in the waveform: in the first phase, the digital organism grows and matures; in the other two phases, the organism tolerates different set of faults and struggles to recover.

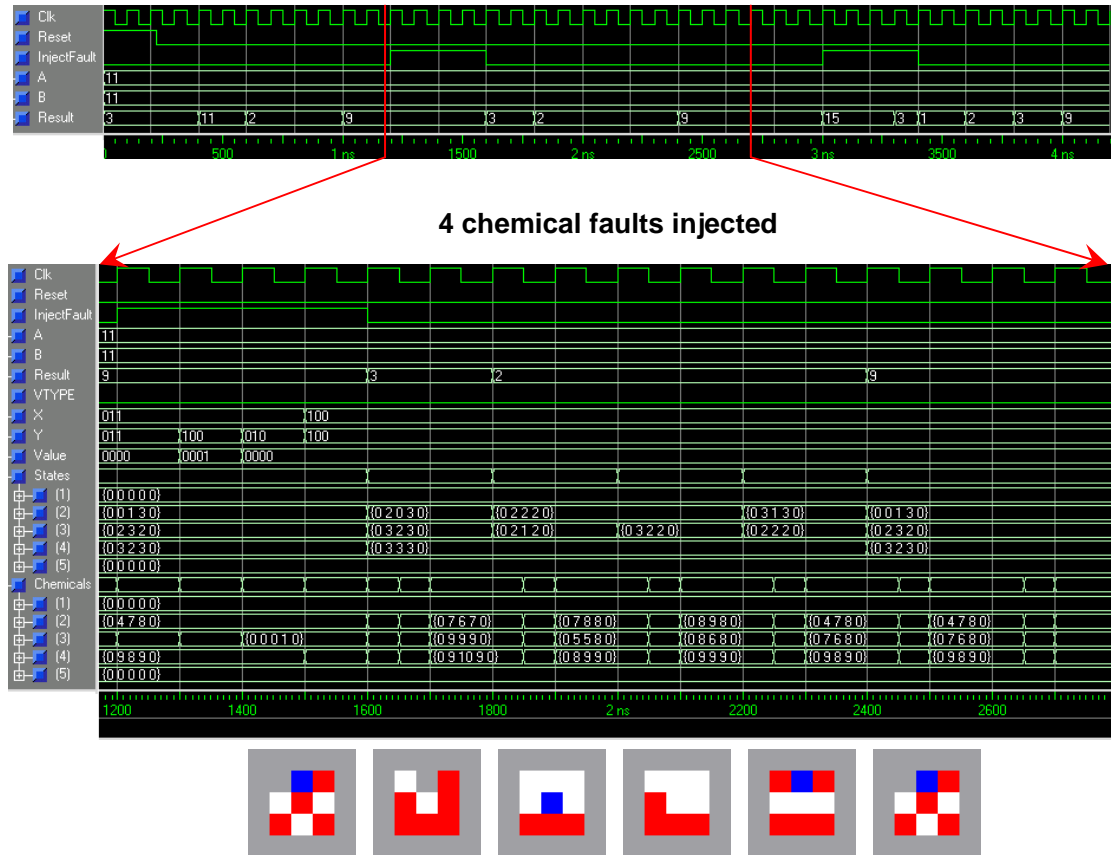


Figure 7-7 Injection of the first set of faults and the recovery procedure

The detail of the waveform for the first phase is demonstrated in Figure 7-6. (The states of the cells are also included for clarity.) It is obvious that the organism matures at 1ns, when the state pattern is identical to the selected one shown in Table 7-3.

The procedure of the first set of transient chemical faults injection is illustrated in Figure 7-7. At the beginning, the chemical of some of the cells are modified, and then the organism resumes growing. It recovers flawlessly at 2.4ns and the output *result* regains the correct value.

²⁴ All the waveforms presented in this work were generated by Modelsim.

Chapter 7 Implementation of Digital Logic System

Figure 7-8 demonstrates the recovery procedure from the second set of transient state faults. The states of the 3 selected “victim” cells are killed (modified to be dead) at the beginning of this period. The organism again recovers completely to the correct pattern at 4ns.

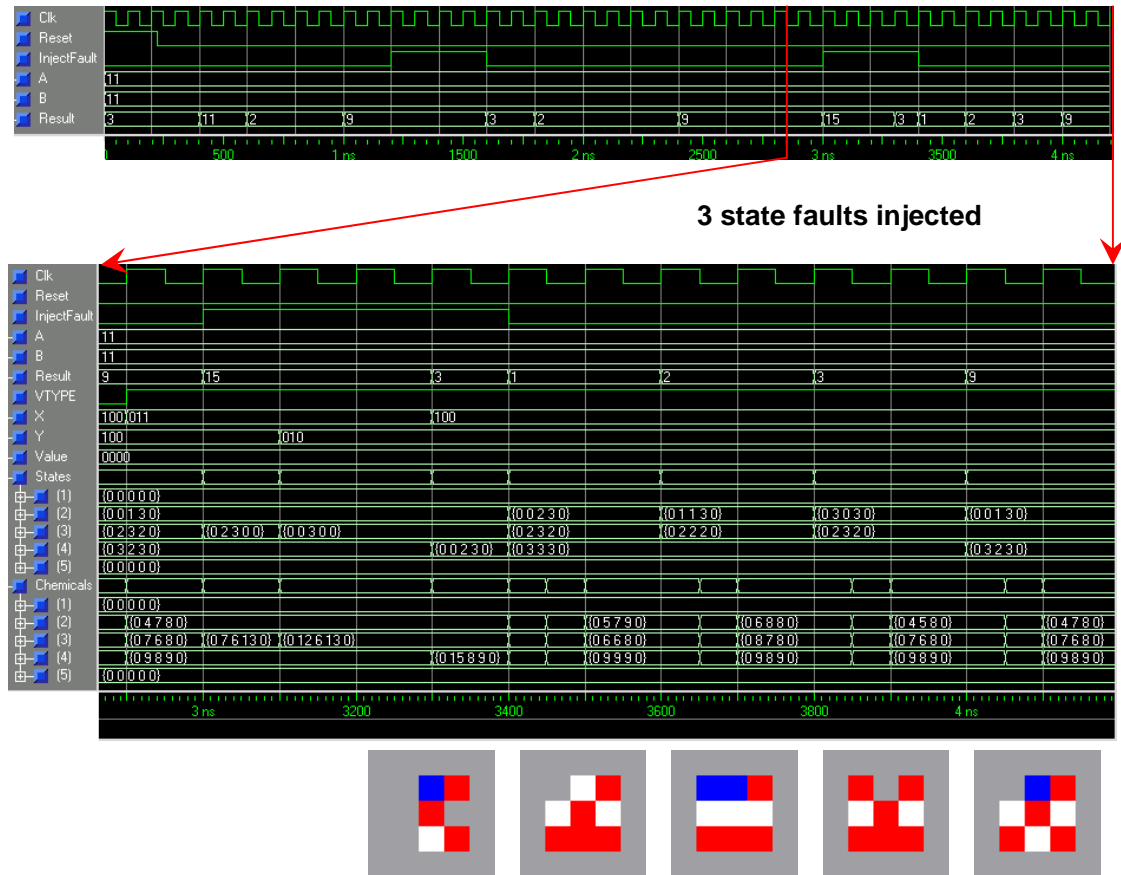


Figure 7-8 Injection of the second set of faults and the recovery procedure

The FPGA on the RC1000 board (details in Appendix B) connects to host PC with very limited data width: only 8-bit read and 8-bit write. So a further FPGA module “IOControler” was implemented to latch all the required inputs and feed them to the digital organism. Another function of IOControler is to cache the output result of the digital organism.

The resources of the XCV1000 FPGA (refer to Appendix C) that was exactly utilized are shown in Table 7-4. DigitalOrganism includes all the resources the 9 DigitalCells take. Similarly, IOControler encapsulates DigitalOrganism, so the former includes all the resources the later occupies.

The entire design (the top level module is IOControler) consumes 2.57% of LUTs and 0.72% of Flip Flops.

Component	LUTs	Flip Flops
NCG	11	
NSG	6	
CU	70	6
ChemicalDiffusionPart	7	5
DigitalCell	102	20
DigitalOrganism	597	151
IOControler	631	178

Table 7-4 Resource Consumption of the Digital Organism FPGA VHDL implementation

Although no other simple digital logic circuits are attempted, such as a 2-bit adder, it is believed that this model can solve that as well, because there is no specifically tuned aspects of this model for this particular functionality, a 2-bit multiplier: for example, all needs to be done in order to evolve a 2-bit adder is to change the fitness so that it compares the results with the correct 2-bit adder outputs and the output from a 2-bit adder is only 3 bits, so no need to check R[0] (in Figure 7-1).

7.6 Summary

A simple combinational circuit was tackled after the French flag problem by this model. This time, Execution Units are introduced to perform the functionality which is calculation of the result for a multiplier.

After incorporating the EU components, the Intrinsic Robust Transient Fault-Tolerant Developmental Model was applied to 2 bit multiplier. In both software simulation and hardware (VHDL) simulation, the evolved solutions exhibit the intrinsic fault tolerant characteristics. Meanwhile considering the design of the hardware structure, it is apparent that the transforming of this model to FGPA implementation is straightforward since this model was designed specifically for hardware platform.

In this chapter, we showed that the model can indeed provide functionality to developments systems, and the VHDL code implementing a perfect individual evolved through the software execution was successfully simulated. In the next

Chapter 7 Implementation of Digital Logic System

chapter, the design and realization of an intrinsic evolvable hardware platform for this experiment will be presented.

Chapter 8

IEWH Implementation of 2-bit multiplier

It was verified that the development model is capable of intrinsic transient fault tolerance and the transforming of the development process of the model to hardware is feasible.

However, due to the complexity and inefficiency²⁵ of the output calculation of the digital circuits generated via evolution, the implemented conventional software evolution to find a robust solution is relatively slow.

Hardware is the ideal platform for the execution of digital circuits, so an intrinsic evolvable hardware platform to accelerate the evolution progress is preferable. In this chapter the intrinsic evolution platform is presented. The infrastructure and design decision of the platform will be discussed first in section 8.1; then the hardware platform and develop environment will be presented in section 8.2 and 8.3 respectively; finally the experiment of this evolvable hardware and results will be discussed in section 8.4.

8.1 Design of the Intrinsic Evolvable Hardware

Although the hardware implementation of digital circuits is much more efficient than the software counterpart, the transforming of the evolvable components to the hardware is not straightforward. In addition, while it is relatively simple to implement the evolutionary algorithm in software, the hardware realization of the EA requires much more work imposed by the complexity of the sequential control circuits which conducts the evolution procedure.

In this sub section, the two issues will be addressed and then the details of the intrinsic evolvable hardware platform will be demonstrated.

²⁵ These limitations are largely imposed by the sequential execution nature of software.

8.1.1 Evolvable Molecule: the Basic Element in the EHW

Just as the situation in software simulation described in Chapter 6, the molecules, defined in section 5.7, are the most fundamental elements of the evolvable components of the model. Each evolvable sub-circuit is composed of several molecules.

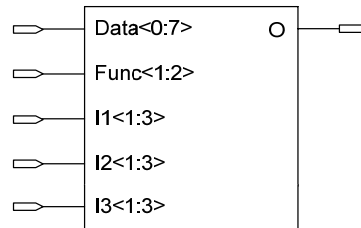


Figure 8-1 Molecule Interface

Rather than distributing the genotype in each cell, in order to save resources and simplify implementation, the genotype is stored centrally in a register outside of the digital organism. Each molecule inputs, including the input selection signals and function selection signal (*I1*, *I2*, *I3* and *Func* in Figure 8-1), are connected to its corresponding bits in the genotype. The *Data* input pin connects to all the input data available to this molecule, which may include inputs to the circuit or the outputs of the molecules on the left of this one.

Due to limited resources in the intended hardware platform, the width of the *data* signal is only 8, so each input selection signal is 3 bit wide. As stated in section 7.3.1, there are 4 functions available for each molecule, so 2 bit is sufficient for *func* signal. In total, each molecule is encoded with $3 \times 3 + 2$ bits.

The internal logical schematic of the molecule is demonstrated in Figure 8-2. *I1*, *I2* and *I3* operate three 8-to-1 multiplexer to select the inputs from the 8-bit width input *Data*. The selected three inputs are then fed to each of the four types of MUX (MUX1 to MUX4). One of the four outputs of the MUXs will be selected by the *Func* input as the final output *O* of this molecule.

This infrastructure of molecule has most similarity with the fundamental multiplexer-based architecture of Xilinx FPGA (see Appendix C), but with less

flexibility, which is a trade-off to eliminates the possibility of evolving some configuration potentially leading to permanent hardware damages.

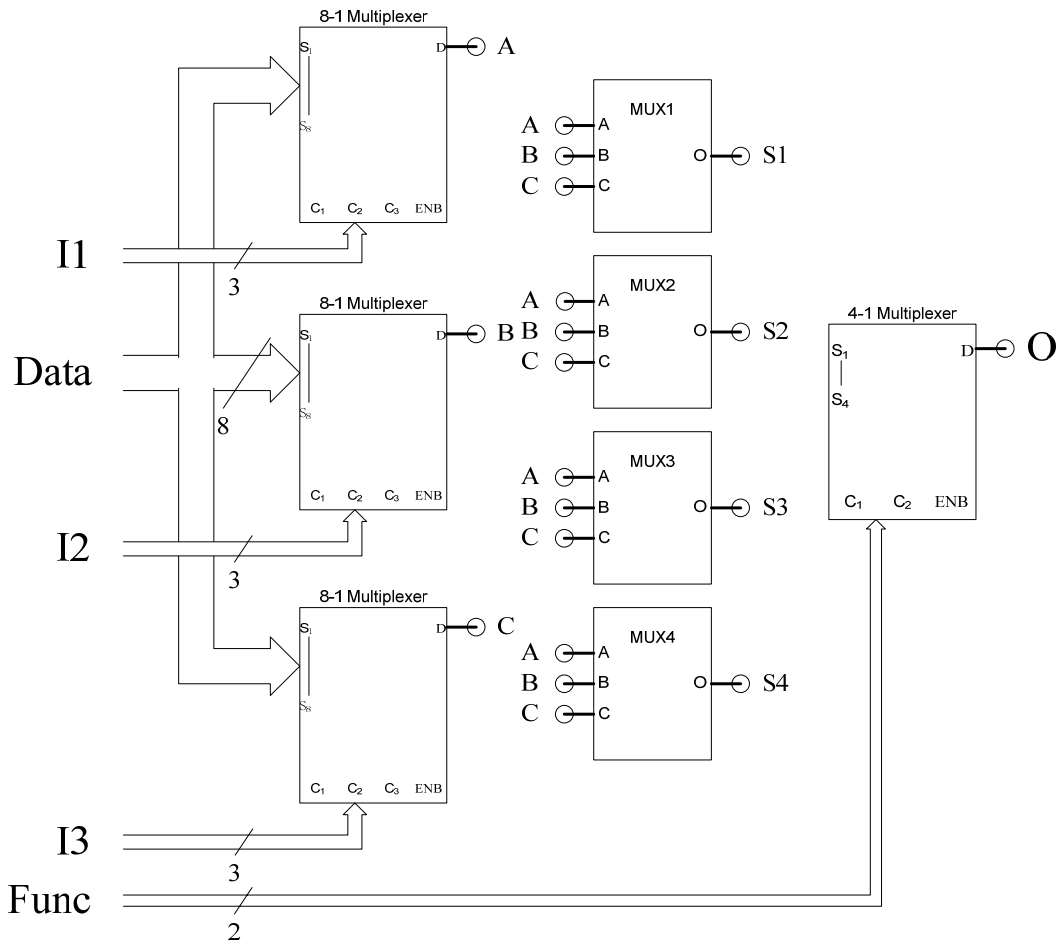


Figure 8-2 The Architecture of Molecule²⁶

I1, I2 and I3 are the input selection signals that select which inputs of the molecules should connect to; Func specify what type of this molecule; Data is all the available inputs to this molecule, while O is the output of this molecule.

8.1.2 Evolutionary Algorithm

No matter how complex a circuit may be, as long as it is implemented in hardware, the execution time of the circuit is only dependent on the global time clock of the system, thus the two phases evolution strategy described in section 7.3.1 are not necessary and were combined together to form a single one.

²⁶ ENB (Enable) pin of each multiplexer is fixed to high (active).

In the unified evolution process, the structures of the entire design, including NCG, NSG and EUC, will be evolved all together. The fitness is the same as in the phase one (in section 7.3.1): the correct bits of the multiplier output result. However, in order to simplify the hardware implementation and consume less hardware resources, the result of the subtraction of the correct bits from the maximum possible value is actually employed as the fitness, so the smaller the better and 0 is the best individual's fitness.

As the entire structure of the digital organism is evolved together, the three genotypes for the three sub-circuits are packed into one single set: the first part encodes NCG, while the second and the last part encode NSG and EUC respectively.

8.1.3 Overview of the EHW

The core component of the Evolvable Hardware platform, the evaluation module, is derived from the hardware implementation described in section 7.4. It was further developed to fulfil the evolvable characteristic.

However, besides this core component, other functional modules were identified according to “divide and conquer” principle. The top level modules are demonstrated in Figure 8-3.

The Intrinsic Evolvable Hardware platform (IEHW) implementation includes 3 main outputs: MAX_Fitness, GenerationCount and Genotype. The first and the second will be updated every generation to reflect latest values, while the last output always propagates the best individual that is evolved so far. Only two inputs are required for this IEHW to function as expected: the global clock signal and reset signal. Other inputs are optional parameters, such as the seed for random number generator (RNG) and stop fitness.

8.1.4 Top-level Hardware Modules in the IEHW

As shown in Figure 8-3, there are 5 functional independent top-level modules which implement the IEHW as a whole.

All the genotypes of each individual are stored in the *Population* module. This is implemented in the FPGA as distributed RAM, for only one individual is manipulated at any given time.

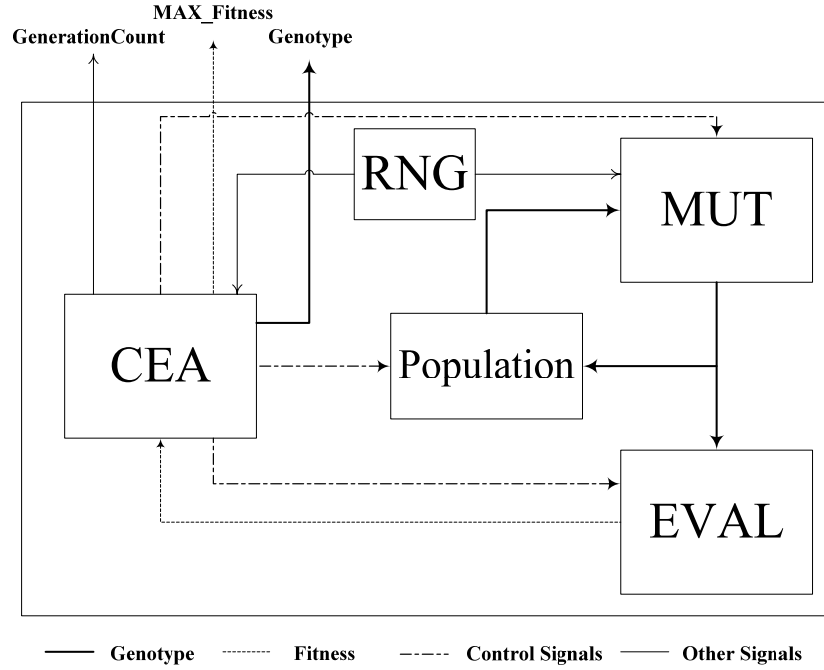


Figure 8-3 Top-level Overview of the Intrinsic Evolvable Hardware Platform²⁷

The Controller of EA (*CEA*) is the central management component which supervises the entire evolution process and all the other modules. The fitness for all the individuals in the population is also stored in this module. The EA is essentially the same as in the software simulation, with one exception: no adaptive mutation rate is employed, although adaptive replacement rate of a bad-offspring overwriting its better parent is used. As fitness now is reversed, the lower the better, principle maximum mutation rate defined in equation 4 has to be modified to the following:

$$p_r = p'_r \frac{f}{f_{\max}} \quad (5)$$

On the right of the equation, the second part of the product is changed compared to equation 4, but this fraction still reduces from 1 to 0 as the evolution converges and as a result, p_r decreases as the fitness of individual approaches the maximum (maintaining the same logic as in equation 4).

²⁷ The last line is the legend of lines deployed in this figure.

The CEA module is realized as a finite state machine (FSM). The *genotype* output signal is driven by the CEA module, however internally CEA just passes the best individual saved in the Population module, so CEA has nothing to do with any genotype signals: it is a representation-independent functional module.

RNG is a 64 bits Linear Feed-back Shift Register (LFSR) (more details are available in the references [14] and [15]), which is employed as a pseudo-Random Number Generator [16]. If supplied, the seed of RNG will also be stored in this module.

The main function of Mutation Module (*MUT*) is to mutate a given genotype and latch the mutated genotype to be used by the *EVAL*. This module reads in the mutation rate and randomly (using RNG) selects molecules to mutate. Molecules are mutated one at a time until the mutation rate is satisfied. This module is also implemented as an FSM. Each molecule change requires two clock cycles. FSM requires another two cycles to start up and terminate. So if only one molecule is changed, four clock cycles are required to complete the mutation.

The core component of this IEHW is the Evaluation Module (*EVAL*), where the Digital Organism resides. Its main purpose is to evaluate the fitness of every individual. This module feeds every possible input to the 2-bit multiplier implemented by the evolved digital organism and sums up the total correct bits. Finally, the result of the subtraction of the total correct bits from the maximum possible correct bits (which is 64 in this case) is the fitness of this individual. The *EVAL* module is made up from 2 FSM: one is used to manage the digital organism and the other is in charge of feeding inputs to the digital organism, calculating correct output bits, the summary and the final subtraction to generate the fitness output of an individual. For each steps, 2 clock cycles are used, while 2 clock cycles are required for the manager FSM to start the growth and one clock cycle is necessary to terminate the evaluation module. Since there are 16 combinations of possible inputs to a 2 bit multiplier, it costs 16 clock cycles to evaluate all of them for a given genotype. For a growth of 6 steps, it takes 2×6 clock cycles plus 16 cycles for evaluation and $2+1$ cycles to start/stop this module.

Rather than a central bus, all the modules have their dedicated input/output signals as shown in Figure 8-3. This feature improves the efficiency of the evolution process. In addition, separate communication channels among components avoid the latency of waiting for the idle of a shared channel, selecting the target component, waiting for response in a central bus communication environment.

8.1.5 Execution Phase of the IEHW

After reset signal is pulsed (low) for one clock cycle, all the modules, including all FSM and internal registers, are cleared to their initial states.

In this state, the IEHW will receive and latch any input parameters if provided, otherwise the default parameters are used.

When the start signal is activated by the host PC, the *CEA* module will take all the responsibilities of the IEHW.

First, the population is initiated one by one, evaluated and saved into *Population*: *CEA* signals the *MUT* to mutate at the highest possible rate so all the molecules in the genotype are randomly generated, then *EVAL* evaluates the individual and propagates the fitness to *CEA*; finally the *CEA* saves the fitness and signal the *Population* module to store the new generated individual. These individuals constitute the initial generation.

After the initial population is ready, the main loop of evolution process begins: in each generation, the *CEA* selects each of the individuals in the *Population* and feeds it to the *MUT*. A fixed mutation rate is employed due to simplicity. For each offspring, only one molecule is mutated (randomly changed). The mutated genotype is then evaluated by the *EVAL* module, and the fitness is again propagated back to *CEA*. If the mutated one (offspring) is better than the original one (parent), or with a probability p_r a worse offspring substitutes the parent, in which case the *CEA* asks the *Population* to store the mutated genotype; otherwise the content of *Population* module is untouched and the mutated one is simply dropped. After all the individuals have undertaken this procedure, a new

generation is created. The evolution will continue to process the next generation unless the stop criterion, the specified fitness²⁸ has been reached, is fulfilled.

When the main loop of the evolution process terminates, the best individual evolved is presented through the *Genotype* output bus, while its fitness and the generation where this evolution stops are propagated out via *MAX_Fitness* and *GenerationCount* respectively.

8.2 Hardware Platform

All of the work described here are achieved in a Dell Dimension 8300 desktop computer (host PC), with a P4 2.8G HT, 1G RAM.

A development board RC1000 Board from Celoxica (which includes an FPGA and other accessories, such as RAM PCI interface, see Appendix B for more details about this board) is plugged in one of the PCI slots in the host PC and all the hardware implementation is realized in this board.

8.3 Develop Environment

The software used in the simulation of this module is developed under Gentoo Linux²⁹, using KDevelop³⁰ Integrated Develop Environment. GNU tool chains³¹ were employed to compile and link the program.

The RC1000 Board driver for Linux kernel 2.4.x and the user space support library are downloaded from Celoxica web site and installed. A dedicated program to interface with the PCI board deploying the support library was implemented. This program can also request current status from the FPGA (such as current best fitness) and output the retrieved information when the user asks it to do so.

²⁸ By default, the specified fitness is the best value an individual can be.

²⁹ Gentoo Linux Homepage: <http://www.gentoo.org/>

³⁰ KDevelop Homepage: <http://www.kdevelop.org/>

³¹ GNU homepage: <http://www.gnu.org/>

8.4 Experiment and Results

The VHDL implementation of the IEHW was synthesized and downloaded to the FPGA. The IEHW was executed to evolve a solution. The implemented IEHW does not take further test on the individuals: the EA won't damage the individuals when they generate the right pattern successfully or give extra bonus to the fitness of those who can tolerate the transient faults and recover.

In this section, the parameters used in this experiment will be presented in subsection 8.4.1. The synthesis report will be described in 8.4.2 and then the results will be discussed in 8.4.3.

8.4.1 IEHW Parameters

The population size employed in this IEHW is 5. The minimum mutation rate p_{min} and maximum mutation rate p_{max} are the same as defined as in section 7.3.2. The replacement rate p_r' is $1xE-6$ too. However, it consumes too many resources to employ float point variables in the FPGA, so circumvention was deployed to implement a very low replacement rate: the current fitness of the individual is left shifted for 6 bits, and then the result is compared with a 32 bits long random integer generated by the *RNG*. If the random integer is smaller than the result, the replacement will occur. The following equation gives the reason for this approximation:

$$\frac{2^6}{2^{32}} \cdot 64 = 9.5367E-7 \approx 1E-6 \quad (6)$$

For a worst individual with 64 as the fitness (the larger the worse), left shift it by 6 bits (equivalent to multiply it by 2^6), when compared with a random generated 32 bits long integer, the probability of the situation when the result is larger than the random integer is approximately $1xE-6$.

The gene settings for Next Chemical Generator (NCG), Next State Generator (NSG) and Execution Unit Core (EUC) are $1x6@4-8$, $1x6@2-8$ and $1x6@3-8$ respectively.

8.4.2 Synthesis Report

The hardware consumption of the entire IEHW is presented in this sub-section. The various layers of modules will be examined in a bottom-up sequence.

The most fundamental building-block for this IEHW is the evolvable molecule, each of which consumes 14 LUTs and no Flip Flops at all, as it is a pure combinational logic circuit.

With the configuration described in the previous sub-section, the NCG, NSG and CU which consist of molecules take 320, 196 and 550 LUTs respectively. CU also utilizes 10 Flip Flops.

A EUC and EU take 84 and 87 LUTs respectively.

One CD module consumes 6 LUTs.

One digital cell occupies 661 LUTs and 19 Flip Flops. The 3x3 digital organism as a whole takes 5732 LUTs (23.3%) and 330 Flip Flops (0.94%).

The EVAL where the digital organism resides consumes 5507 LUTs (22.4%) and 360 Flip Flops (1.46%), while the CEA takes 164 LUTs (0.67%) and 115 Flip Flops (0.47%).

The RNG is a special module which consumes more Flip Flops (64) than LUTs (only 3).

The MUT module occupies 106 LUTs and 86 Flip Flops, while the Population module is RAM based, so it only consumes 3 BRAMs (Block RAM).

The entire design which was downloaded to the FPGA takes 7833 LUTs (31.9%), 926 Flip Flops (3.77%) and 239 IOBs (58.6%) in total.

It can be noted that, the resources can not be calculated reliably based on how many base components are used: for instance, if multiply the LUTs a cell occupies (661) by 9, at least 5949 LUTs should be consumed by the 3x3 digital organism. However, only 5732 LUTs are actually utilized. This is accounted for by the optimization algorithm in the VHDL synthesizer (or compiler): it can scan the circuits and pack two semi-used LUTs to one, or reuse identical LUTs.

In other cases, the resources one module takes is larger than all the sub components consume, due to apparent reason: new functionalities are

incorporated. For example, the amount of Flip Flops used in digital organism is actually more than 9 cells occupy in total, which is caused by the fact that the digital organism has to provide input latch and output buffer for all those sub components (digital cells) in it.

The IOBs are used to interface the EHW platform with the hosting PC. Most of the used IOBs are to propagate the outputs from the EHW, among which the most IOB usage is observed for an output interface that is in charge of transferring the resulting genotype to a memory unit located on the board where the FPGA resides.

8.4.3 Results of Experiment

As in the software evolution, the IEHW won't stop unless a perfect individual is found. In order to read out the genotypes of the solutions, a dedicated Direct Memory Access (DMA) module was implemented to transfer this large amount of data. The FPGA was configured to execute at a 5.00MHz clock.

100 runs³² of the IEHW were carried out in total. On average, each evolution took 7,057,256 generation to find a perfect solution. The minimum, midterm and maximum of the generations are 56,200, 2,875,532 and 51,026,945 respectively. Figure 8-4 shows the distribution of generations for these 100 runs.

On average, it takes about one minute for the IEHW to find one solution, while it normally costs more than 10 minutes to evolve one solution in software simulation.

8.5 Summary

In this chapter, based on the VHDL design of the digital organism described in previous chapter, the possibility of realizing a intrinsic evolvable hardware platform for evolving a 2-bit multiplier with this developmental model was explored

The design of this IEHW platform was first presented, including the detailed structures and how the execution of evolution is handled. Then the development

³² Each run has different Random Generator Seed.

environment was briefly overviewed, while more details are present in corresponding appendix sections. At last, the details of the experiments performed were described and the results were presented, which shows at least one order of magnitude of performance improvement compared with software simulation.

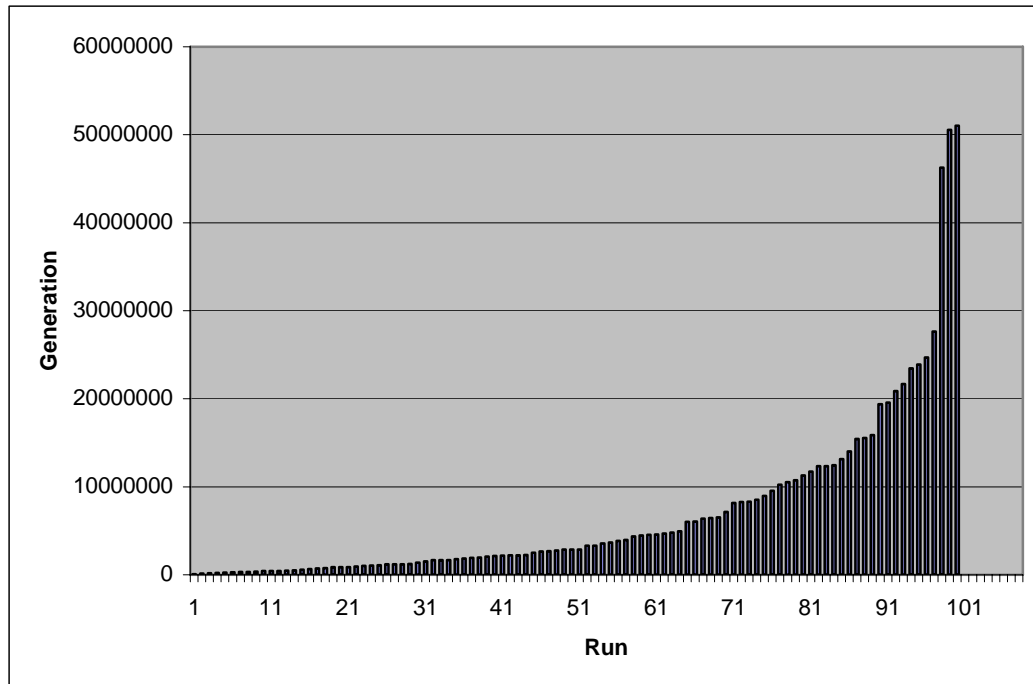


Figure 8-4 The distribution of the generations of the 100 experimental runs³³

So far, only combinational digital logic circuits have been evolved with this model. In the following chapter, a sequential digital logic circuit will be tackled and the model will be further improved.

³³ The sequence of the runs is sorted in the ascending order of the generation.

Chapter 9

Evolution of Sequential Digital System

Sequential logic is indispensable in real world digital systems. However, only combinational logic was evolved in previous chapters: French flag problem is solved with combinational circuits, while the second application, multiplier is on its own a combinational logic. This chapter is dedicated to investigating the feasibility of implementing sequential digital system with the development model.

An introduction to sequential digital circuits and memory will be presented in section 9.1. Adaptive behaviour emerging in the development model proposed will be investigated in section 9.2, while we move on to a real random access memory unit implementation in the following section. Finally, the discussion about the results of the experiments will be discussed in section 9.4.

9.1 Introduction

In digital circuit theory, circuits can be classified into two types: combinational logic and sequential logic. The former type of digital circuits, also known as combinatorial logic, only generates output from present input to the circuit, no other information is necessary. On the other hand, the output of sequential circuits depends not only on current inputs, but also on the internal states of the circuits and previous input (the history of input). Thus the main difference between these two types is that sequential circuits have some kind of storage capacities (memory), while combinational circuits do not have any at all.

One of the simplest forms of sequential digital circuits is memory, which is also extensively used in all ranges of applications. In most cases, memory refers to random access memory (RAM): data can be stored and accessed in any order, not necessarily in sequence, with constant access time. RAM is normally used as the primary storage (main memory) in modern computer systems.

One key characteristic of RAM is that they are normally fast to write and read, however they are volatile: RAM loses all content when power is cut off.

While transient faults account for 80% or more system failures in digital systems [17][182], RAM is considered one of the most exposed components which are vulnerable to transient errors due the high spatial density of storage units and the high volume of information it stores. It is demonstrated that when estimating soft errors occurred in microprocessors, it is important to consider transient faults in memory arrays used by the microprocessors. [181]

While several researches were carried out to build fault tolerant RAM units [72]-[75], most of them are based on conventional fault tolerant techniques. In the proposed development model, the states of cells exhibit strong resistance to transient state flips, which makes it a good candidate to be explored as an implementation of memory units.

In the next section, a unique way of representing information saved in a digital organism to achieve storage behaviour (like latches) will be discussed. The idea will be extended to provide a RAM unit interface which is more similar to normal RAM unit implementations in the sections following the next one.

9.2 Adaptive Behaviour

Before trying to evolving a RAM unit, an intermediary stage was composed: to check whether the model can exhibit any adaptive characteristics: the organism can adjust itself according to an environmental signal. The idea behind this approach is that, the overall state pattern of the digital organism can be considered as information saved within it. If the pattern can change given a different environmental signal is applied, then the pattern can be employed to function as a memory unit.

The software simulation used in Chapter 6 is modified to satisfy the requirements of this experiment: environmental signals are introduced. In order to reuse existing facilities, one of the boundary condition is utilized as environmental signals: the states of the left border of the digital organism are employed as the external signals. In this experiment, state 3 (which is decoded to red in the figures) is the actually used signal applied to the left border.

The aim of this experiment is to evolve a French flag which can transform to an “inverted” French flag (exchanging red area with blue area in a normal French

flag) when the environmental signals are changed to “red”. The fitness of this evolution is the summary of the correct (in terms of states) number of cells before and after the environmental signal is applied. In addition, bonus points are given to those individuals with stable patterns.

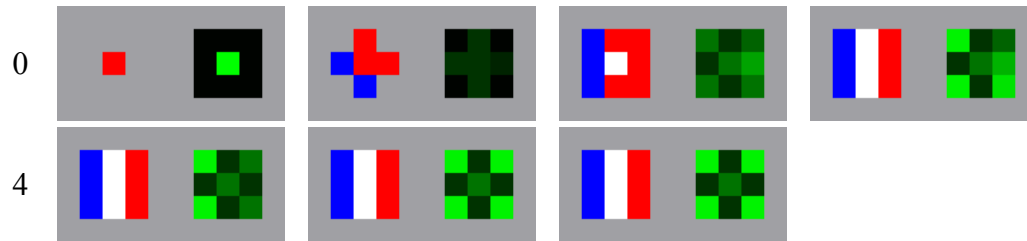


Figure 9-1 Adaptive French flag grows to maturity

One of the evolved adaptive French flag is demonstrated in Figure 9-1. This individual matures at step 5, as step 6 is identical with step 5.

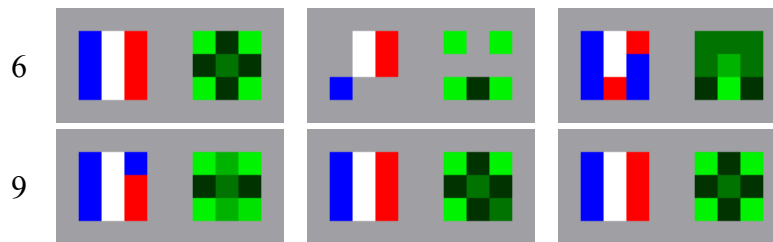


Figure 9-2 Adaptive French flag recovers from a set of transient faults

This evolved solution exhibits transient fault tolerance as well (see Figure 9-2): a set of transient faults (including wiping 4 states and 4 chemicals to zero) were injected at step 7, and the organism recovered to its stable pattern at step 11 (which is the same as step 5 and 6 in Figure 9-1).

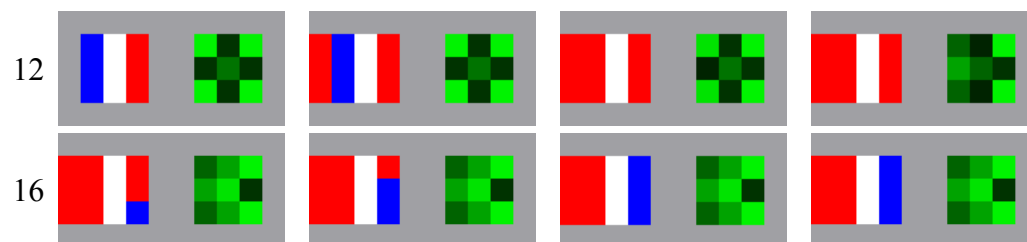


Figure 9-3 The flag transforms to “inverted” flag to adapt to the environmental signal

As shown in Figure 9-3, at step 13, the environmental signal is changed from 0 to 3 (the red band on the left border). By step 18, the “inverted” French flag is stabilized.

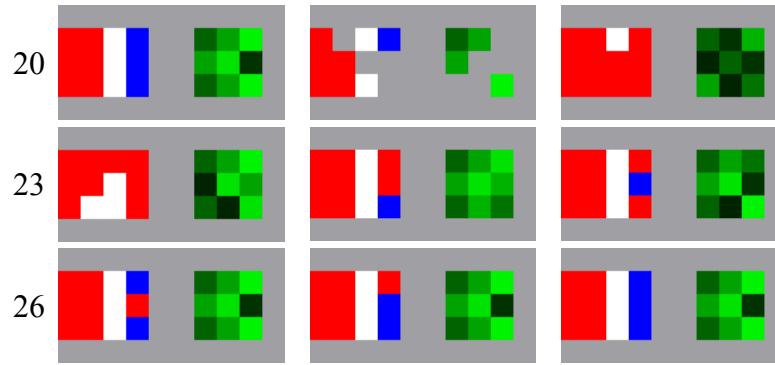


Figure 9-4 Inverted flag recovers from transient faults

Then it is demonstrated in Figure 9-4 that this inverted French flag can also recover from transient faults to its original pattern. 4 states and 5 chemicals are set to 0 in step 21, and the flag successfully regenerates to form the same pattern as in step 18 in Figure 9-3.

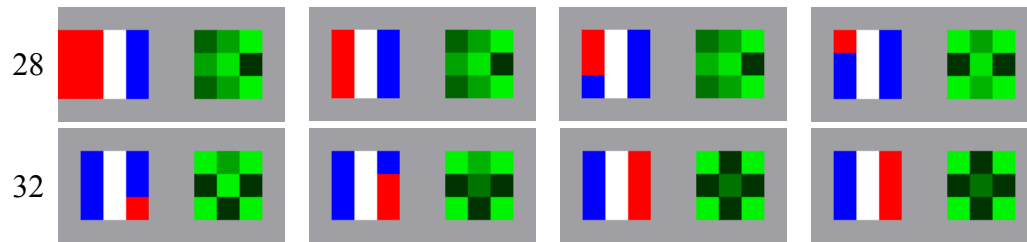


Figure 9-5 Re-grow to French flag after resetting environmental signals

In Figure 9-5, the capacity of transforming back to the original French flag after the environmental signals are set back to 0 is demonstrated. Step 34 and 35 are exactly the same, which is identical as step 5 in Figure 9-1.

These 5 figures show the complete behaviour cycle of the adaptive French flag: by default (with 0 as environmental signals) this organism grows to a robust French flag, while when exposed to environmental signals (3 in this case), it adapts to an inverted French flag, retaining its robustness with regards to transient faults. This organism can also transform back to normal French flag when the external signals are removed.

9.3 Robust Random Access Memory

This adaptive behaviour lays the ground which memory units can be built. However, in the case of memory, the external signal is not always applied: the

writing of data to the memory is triggered by a write-enable signal, and the data to store is only available when write-enable is active.

In order to explore the feasibility of the convention used in memory units, in this sub-section, 1-Bit RAM unit will be investigated first in section 9.3.1, while RAM with more capacity is further examined and limitation of the developmental model is discussed in section 9.3.2.

9.3.1 1-Bit Primitive RAM Implementation

Based on the findings described in section 9.2, it is believed that the states of cells can adapt to an external signal and still maintain its resistance to transient errors in the states.

In order to have a memory implementation, the interface of the RAM units should at least contain these pins specified in Table 9-1.

Input Information	Description
Write-enable	Trigger the writing procedure when active: write input data to the unit specified by Address
Input Data	Data to save (may be combined with Output Data)
Output Data	The Data stored at Address
Address	Which unit in this memory should be used to write/read

Table 9-1 General RAM Memory Interface

9.3.1.1 Input and Output Configuration

Mapping the normal interface of RAM units to the development model involves specifying what acts as the various input and output pins.

In the experiment, all the inputs are special boundary conditions. The chemicals on the left border are assigned as write-enable signal: when maximum chemicals are applied to the left border, the value of the input data should be stored in this organism. The Input Data port of this RAM unit is the states on the left border: when 3 (red in the figures) are applied to the left border, the organism should change to another state to reflect that the saved value is changed.

At this stage of experiment, there is no real output from this organism at the Output Data port, instead the pattern of states is considered as the representation of the value saved in this organism (as the case in the adaptive behaviour experiment described in previous sub section): a French flag represents the

default value 0, while an inverted flag is considered as value 1. As this is only 1-bit RAM unit implementation, so only these 2 values are possible to be stored in this organism.

9.3.1.2 Experiments

The evolution parameters are identical with the experiment described in section 9.2, with one exception: as the target behaviour of this organism is different from the adaptive experiments, the fitness of individuals in this evolution is modified to search for flags which can latch values when signalled to do so and maintain the latched value even when the environmental signals are not present any more. Fitness bonus is also awarded to those individuals with stable patterns.

One of those evolved solutions is shown in Figure 9-6: The organism stabilized in step 3.

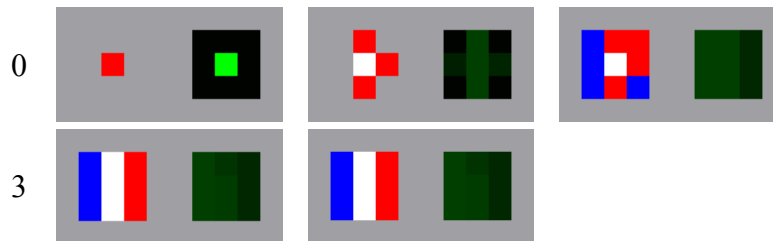


Figure 9-6 1-Bit RAM Growth

In Figure 9-7, robustness of the organism is demonstrated: a set of transient faults, including 3 state and 3 chemical faults, is injected, and the organism manages to recover within 8 steps to retain the French flag pattern, which represents value 0.

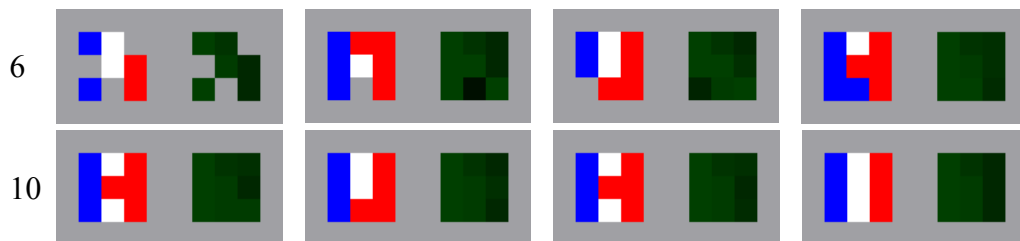


Figure 9-7 1-Bit RAM Recovery from faults with value 0 saved (default)

Write-enable signal (maximal chemicals are applied to the left border) is activated in step 15 (see Figure 9-8), and input data is set (red on the left border), which means the flag should change to an inverted French flag to represent that 1 is saved in this organism. The organism starts to react to this external signal: it

transforms from a French flag to an inverted one, as shown in step 24 and stabilized (step 24 and 25 are identical).

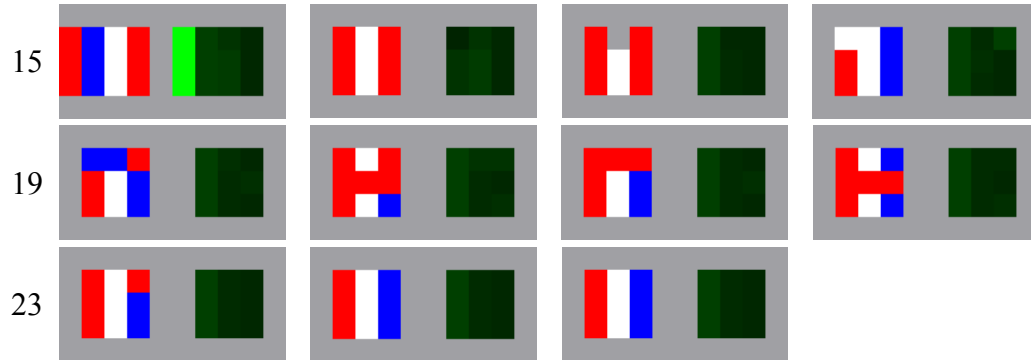


Figure 9-8 1-Bit RAM Stores 1 as its new value

In Figure 9-9, the process of saving 0 to the organism is demonstrated: in step 26, write-enable signal is activated and the input data is 0 (which is shown as grey on the left border in the pictures). The organism adapts to this writing signal and transforms back to the normal French flag: both the states and chemicals at step 30 and 31 are identical to step 4 in Figure 9-6.

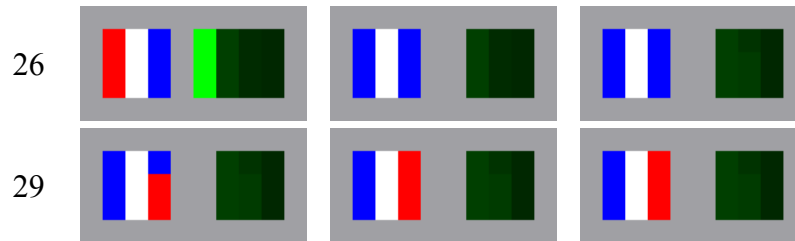


Figure 9-9 Saving 0 to 1-Bit RAM when 1 is stored

9.3.2 Towards Larger Robust RAM

In the previous sub section, 1 bit robust RAM was successfully evolved which exhibit fault tolerant behaviours, however it uses pattern to represent the store value and lacks an output interface. In this section, a robust RAM unit with larger capacity is further targeted at: more specifically, 2 bits and 3 bits RAM unit will be considered and output interface will be incorporated.

9.3.2.1 Modification to the Model

Some of the growth rules are quite restrictive, which eliminate a significant portion of the search space accessible to the evolutionary algorithm. In order to broaden the possible solutions for the EA search, one of the growth rules is

slightly modified to relax constraints: cells with 0 chemical level won't die (change to state 0 automatically). Essentially, the bias towards 0 chemical is removed from this model.

It would be an interesting idea for future research to see how the modification can affect previous experiments, which presumably speed up the evolution.

9.3.2.2 Memory Unit Interface

In order to bring the RAM unit more inline to a real implementation, the RAM unit should have an output interface, rather than relying on the pattern to represent what is store.

For the experiments described below, the output of the RAM unit is connected to the state signals on the right-bottom cell(s). For the 2-bit RAM unit experiment, the state signal of the right bottom cell (*cell (3,3)* in Figure 7-1) drives the output. For the 3-bit one, the additional one bit output is connected to the least significant of the state signal of *cell (2,3)* in Figure 7-1.

The input data pin has to be modified as well, as for 2-bit and 3-bit RAM units, 2 and 3 bits width of input data pin are required respectively. For 2-bit RAM experiment, the 2-bit input is connected to the state signal of the left bottom cell (*cell (3,1)* in Figure 7-1). For 3-bit experiment, the extra input signal is connected to the least significant bit of the state signal of *cell (2,1)* in Figure 7-1.

On the other hand, write-enable signal remains the same as in the 1-Bit RAM experiment: highest concentration of chemicals on the left border of the mechanism activates writing to the RAM unit.

After a value is stored to the organism, the output will be the saved value all the time (because there is no read enable input pin) until a new value is written.

9.3.2.3 Experiment Result

The parameters to the evolutionary algorithms used in the experiments are basically the same as previous setup, with one exception: there are 2 genes and the genes setting is *1x12@6-12, 1x15@2-15*.

Both 2-bit and 3-bit RAM units were successfully evolved: out of 100 runs for each experiment, none failed to evolve max fitness individuals. On average, it

takes about 84K generations/2.6M evaluation to evolve a 2-bit RAM unit, while it takes 1.9M generations/75M evaluation to obtain a 3-bit RAM unit. More details about the results are given in Table 9-2.

	2-bit RAM Experiment	3-bit RAM Experiment
Runs	100	100
Successful Runs	100	100
Average Generation	84,236.2	1,899,136
Average Evaluation	2,697,987	75,018,257
Midterm Generation	71,194.5	1,395,797
Midterm Evaluation	2,189,625	53,961,508
Min Generation	10,945	144,492
Min Evaluation	323,123	4,852,310
Max Generation	416,614	13,358,635
Max Evaluation	14,752,865	554,217,046

Table 9-2 2-bit and 3-bit RAM Unit Experiment Results

9.4 Discussion

Although the experiments described in the previous section are successful, no further attempt was made to evolve RAM units larger than 3-bit. From Table 9-2, we can see that it takes as more than 22 times of generation to evolve 3-bit RAM unit as that of evolving 2-bit RAM units.

In addition, using this model to evolve RAM units requires much more resources compared to traditional fault tolerant technique, such as Triple Modular Redundancy: the FPGA has built-in RAM unit within its CLBs, so it is quite efficient to implement RAM unit directly in FPGA. On the other hand this development model is not specifically designed for sequential logic circuits, such as a RAM unit.

One potential approach to overcome the issue mentioned above is to make use of a specially designed FPGA which has native primitives to takes various development principles into account, such as POEtic chip [186][187]. The “molecule” used in the development module can be made as a building block of a specifically designed FPGA, which can be configured dynamically with a configure binary string when it is in operation (online). This should also be quite beneficial with regards to reduce resource consumption for the combinational applications discussed in the previous chapters.

9.5 Summary

In this chapter, sequential digital circuit was introduced, where special attention was paid to memory units. Adaptive behaviour of the proposed developmental model was then investigated, which led to a 1-bit RAM unit primitive.

After that, larger capacity RAM units were evolved. The experiments can be considered as a proof of principle: the cell states in the organism can serve as a storage device which can provide transient fault tolerance. High concentration of chemicals applied on the left border will trigger a write process to the organism, when the value represented by the state signals connected to the left border will be stored to the organism, while the output is always the currently saved value. While the experiments for 2-bit RAM and 3-bit RAM unit are successful, it is quite hard and inefficient for this developmental model to implement pure sequential circuit logic, such as RAM units.

In the following chapter, the model will be given another new task to tackle, autonomous robot controller.

Chapter 10

Autonomous Robot Controller

In previous chapters, simple combinational digital circuits and sequential digital logics were considered. The objective of this chapter is to build an autonomous robot controller using this developmental model, which is a more challenging problem than a simple combinational logic..

We will begin with an introduction to the robot and its controller in the first section of this chapter. In section 10.2, the robot itself and the tasks for the robot to tackle will be presented. The experiment will be carried out in a simulation program, called Webots, which will be described in section 10.3, along with details about the setup of the experiments. Finally, discussion about the evolved controller will follow.

10.1 Introduction

Although the concept of Robot was present in Lie Zi text, data back in the 3rd century BC [183], there is no widely accepted definition for the term. Different countries have different opinions about what it takes to be a robot. However, some common characteristics among those definitions do exist:

- It should be artificial, rather than originated from nature;
- It has the ability to sense its environment;
- It has intent or agency of its own.

10.1.1 Robot and Application

Robots are playing increasingly important roles in the modern world, from manufacture industries to doing domestic tasks.

Robots are normally used to improve productivity, accuracy, and endurance, such as factory robots in car production. Another common field of robotics is to complete dangerous, dull or inaccessible tasks, such as pipe-cleaning robot, military robots which can defuse roadside bombs.

10.1.2 Autonomous Robot Controller

“Autonomous robots are robots which can perform desired tasks in unstructured environments without continuous human guidance.”[184] Autonomy is particularly desirable when it is difficult for humans to control robots in a timely basis, such as in space exploration, where latency of communication is high and it is impossible to prevent interruptions.

Autonomous robot controllers should have following characteristics [184]:

- Obtaining information from environment;
- Work without human instructions;
- Moving (part of) itself in its operating environment and
- Preventing situation harmful to human, property or itself.

10.2 Robot and its Environment Setup

In order to speed up the evolution process, a simplified theoretical robot model, called Kiki was employed in the experiments. The task given to the autonomous robot controller is to find the *exit* in the surrounding walls of the environment without hitting and obstacles.

10.2.1 Kiki Robot

The simplified theoretical robot, Kiki, is outlined in Figure 10-1. The body of the robot is a cylinder with two wheels attached to the bottom. In addition, two IR sensors are mounted to provide obstacle (walls) detecting capacity. This robot is in fact quite similar to the Kephera robot [188].

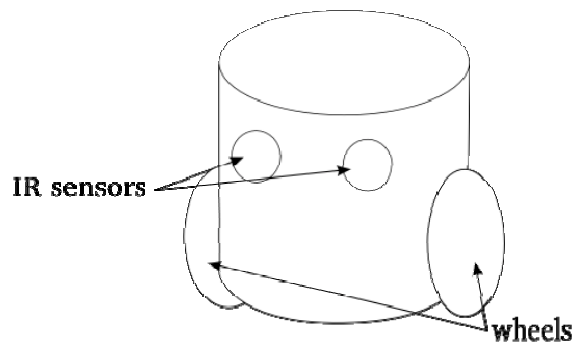


Figure 10-1 Kiki robot

Kiki robot is a Differential-Wheels robot, which has two wheels that can be steered independently: when both of the wheels have the same speed and direction, the robot will move straight forward or backward; by differentiating the speeds of the two wheels, the robot can be made to turn; If the turning directions of the two wheels are opposite, the robot can turn more quickly.. Some key dimensions of the robot are given in Table 10-1.

Item	Size (cm)
Radius of body cylinder	4.5
Height of the body cylinder	8
Radius of the wheels	2.5
Axle length between the two wheels	9

Table 10-1 Dimensions of Kiki Robot

The two infra-red sensors function according to the simple linear curve shown in Figure 10-2.

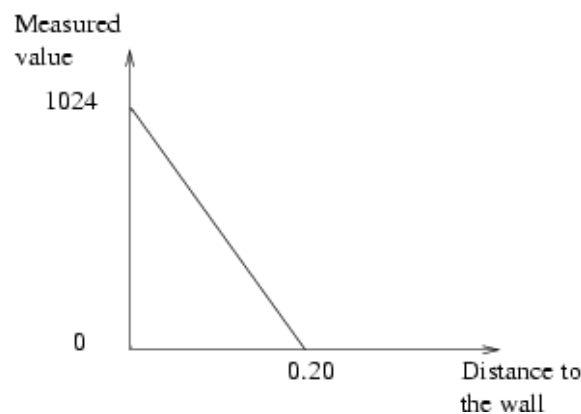


Figure 10-2 Distance measurements of the Kiki sensors.

10.2.2 Robotic Task and Environment

Two environments are constructed for the robot to evolve in: the Kiki robot should avoid collision with any walls and manage to get out of the environment via the exit. The two worlds the robot evolves in are shown in Figure 10-3 and Figure 10-4.

Both the width and height of the two worlds are 1 meter. The exits on both worlds are 17cm wide, while the gaps in the maze are 20cm wide.

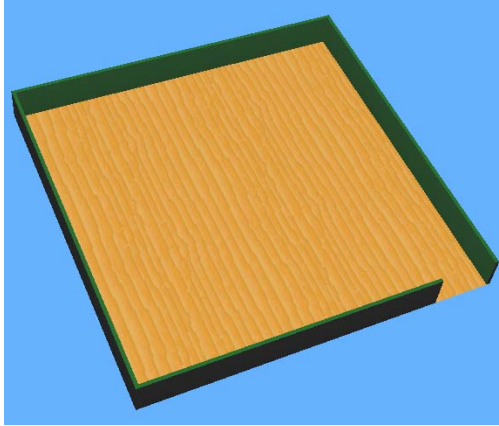


Figure 10-3 Simple environment

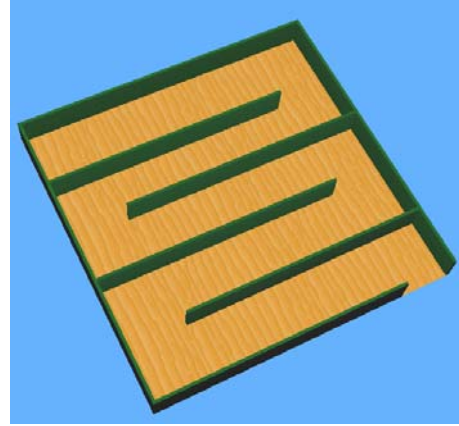


Figure 10-4 Maze environment

10.3 Webots Simulation

The experiments were performed with a simulation program, called Webots, which is capable of simulating 3D mobile robots in complex environment. The simulator can take physical rules into account, thus the result is quite comparable to these that would actually be achieved with a real robot.

10.3.1 Introduction to Webots

In order to run a simulation in Webots, a virtual *world* needs to be created first, which contains the 3D environment, and the robot itself. In addition, physical properties are defined in this virtual world, such as inert object. Robots, also called active objects in Webots, can have different kinds of sensors, locomotion schemas and actuator devices.

The other necessary part of a simulation is a robot controller, which is a program (written in C, C++ or java) to manoeuvre the robot in the environment. An optional program, called supervisor controller, can do more manipulations and administration tasks than a normal robot controller: it can move or rotate any objects in a world, tracking coordinates of any objects, among other interesting things.

10.3.2 Evolutionary Framework Deployed in Webots

Robot controllers were written in C++ in order to be incorporated easily with the evolution framework developed for previous experiments. Supervisor controller is deployed to calculate the fitness based on the robot's position.

10.3.2.1 Robot Controller

The evaluation of the developmental model is moved to a dedicated robot controller program, which receives a genotype from the supervisor via emitter/receiver mechanism provided by Webots for inter-process communication. For each genotype, this controller grows it to a mature organism, and then applies the IR sensor values to the execution input signals on the left and top border of the organism (see ir0 and ir1 in Figure 10-5). For each robot step, the execution circuit is evaluated for each cell and the output execution signals from cell (3, 3) and (2, 3) (see s0 and s1 in Figure 10-5) drive the two wheels mounted on the robot.

As shown in Figure 10-5, executing signals are only 2-bit wide, while the readings from IR sensors are 10-bit, so an encoding schema is used to reduce the IR values: the three most significant bits from retrieved IR values are fed to the organism. s0 and s1 encode the speed of wheels according to Table 10-2.

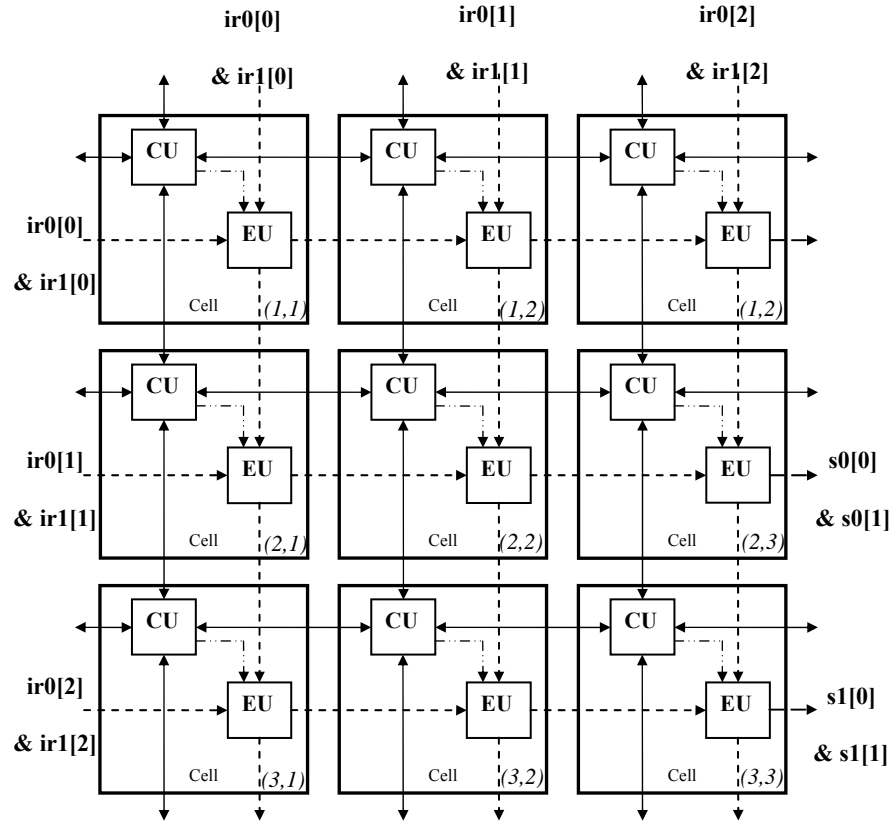
Encoded (s0 & s1) (In binary)	00	01	10	11
Decoded Speed (<i>rad/s</i>)	0	2	0	-2

Table 10-2 Wheel speed decoding schema

10.3.2.2 Supervisor Controller

The evolutionary algorithm is executed by another program, supervisor controller. It maintains the population for the evolution, sends each genotype to the robot controller and checks the position of the robot to calculate the fitness of an individual.

The fitness of an individual is the nearest position to the exit point in its life time (when it is running the robot).



LEGENDS:

CU	Control Unit	EU	Execution Unit
----->	EU Function Selection	↔	States (2b) & Chemicals (6b) Signals
—	Cell border	----->	Executing Signals (2-bit width)
ir0 & ir1	Encoded IR values	s0 & s1	Encoded speeds of the wheels

Figure 10-5 3x3 Cells Digital Organism for robot controller

10.3.3 Experiments and Results

Both evolutions of the two experiments for the two tasks have identical setup of parameters: population size is 50, while p_{min} , p_{max} , p'_m and p'_r are defined as $1/mols$, 0.5, 2 and $1 \times E-6$ respectively. The gene settings are $1 \times 12@6-12$, $1 \times 15@2-15$ and $1 \times 8@2-8$.

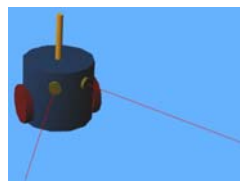


Figure 10-6 Kiki robot

Both tasks are successfully solved by multi-breed EA. In order to draw the trajectory, a pen device is attached to the Kiki robot (see Figure 10-6 and Appendix Appendix D).

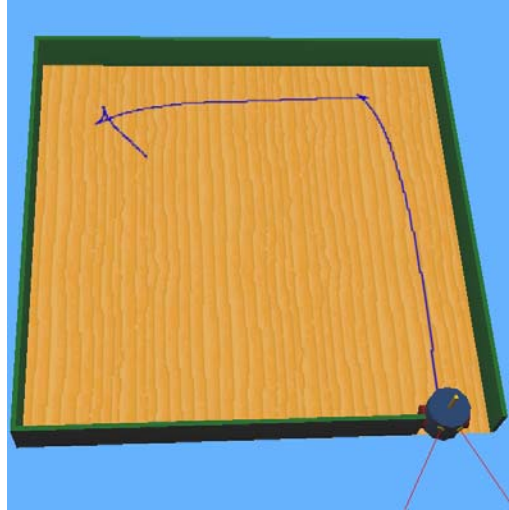


Figure 10-7 Trajectory of Kiki for the simple world

Figure 10-7 and Figure 10-8 show the trajectories for the solutions for the two worlds respectively. These are examples of some of the best evolved solutions.

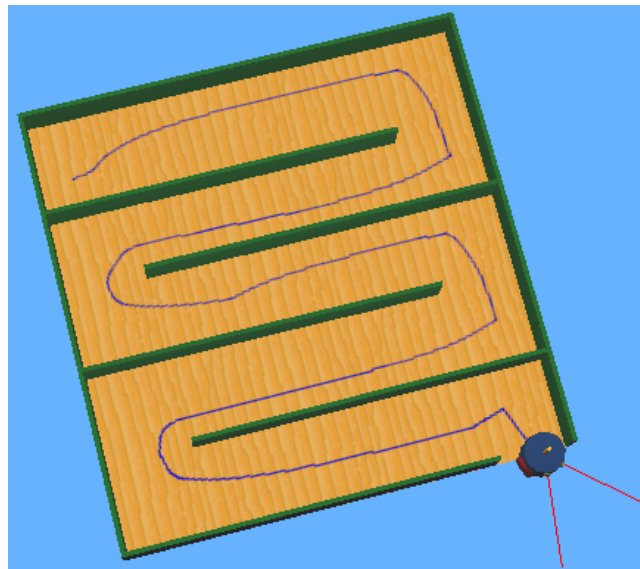


Figure 10-8 Trajectory of Kiki for the maze world

The solutions found can also work even if the start position and orientation of the robot are modified, see Figure 10-9 and Figure 10-10 for two examples for the two worlds.

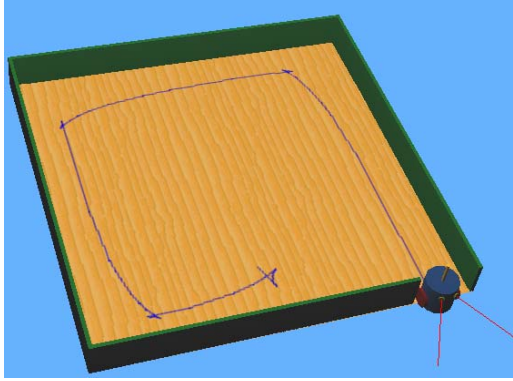


Figure 10-9 Trajectory for the simple world with different starting point

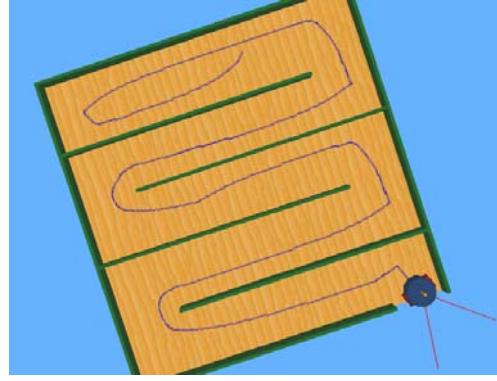


Figure 10-10 Trajectory for the maze world with different starting point

Transient faults can also be tolerated by evolved robot controllers. As the growth is quite fast, so the regeneration (re-growth) of the organism has to be suspended in order to see the malfunction of the robot.

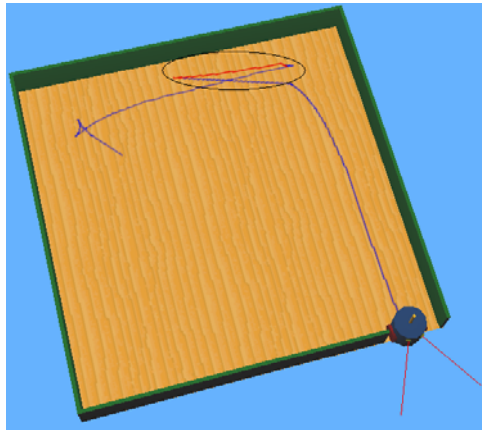


Figure 10-11 Trajectory for the simple world with faults

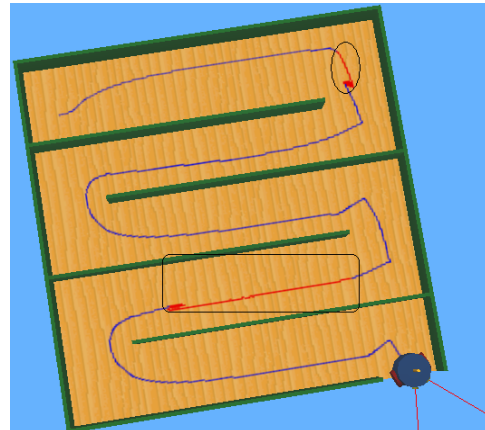


Figure 10-12 Trajectory for the maze world with faults

In the two images above (Figure 10-11 and Figure 10-12), the circled parts of the trajectories (in red) are drawn when transient faults are introduced to the organism states and the re-growth of the organism is suppressed, while blue lines are drawn when no transient faults are injected. When transient faults are present, the robot controller does not have a clear aim: it either moves backwards as in Figure 10-11, or it is stuck and *shakes* in one position and refuses to move forward as in the two cases in Figure 10-12. However, once these faults disappear, the controller recovers and a good operation is resumed.

10.4 Discussion

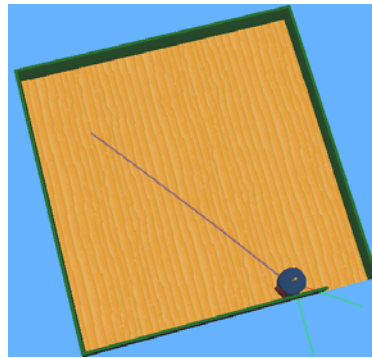


Figure 10-13 A case where the simple controller fails

For the simple world, the evolved strategy of the robot controller is quite simple: the robot can just make use of its left IR sensor to detect walls and when something is detected it just turns right. It is quite easy to conceive a starting position/orientation to break the controller, such as the case shown in Figure 10-13 where the left IR sensor is not triggered until it is too late for the robot to turn around.

On the other hand, the controller evolved for the maze world is equipped with a more advanced strategy: if it hits a wall, try to turn left; if there is no way on the left, turn right. This controller can also find its way out of other modified maze worlds, such as these shown in Figure 10-14 and Figure 10-15.

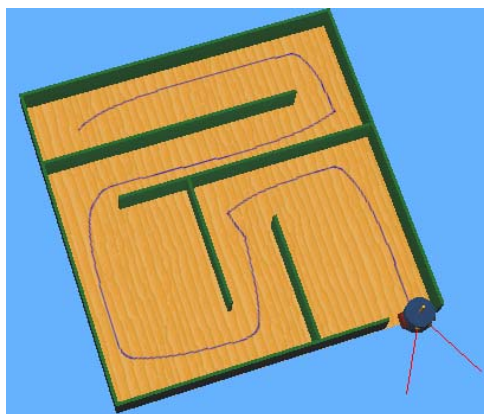


Figure 10-14 New maze world 1

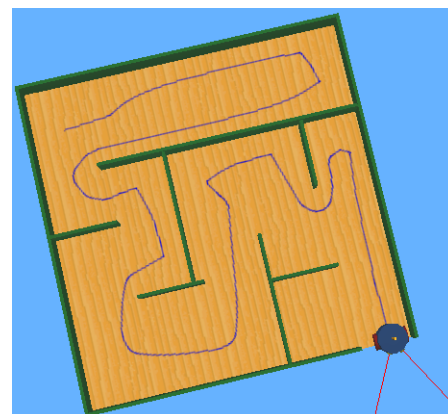


Figure 10-15 New maze world 2

However, the controller obtained from the maze world also has weak points. It has no way to avoid walls if neither of its IR sensors can detect it, such as the

case in Figure 10-16: it's not until the robot is too close to the wall that the right IR sensor has a reading so it does not have enough space to turn around. This issue would be solved with more IR sensors.

Tyrrell et al. considered a novel evolutionary algorithm to evolve robot controllers which can avoid collision with obstacles [48]: at any given time, a parent population of individuals are stored (called P), as well as several mutated "clones" of P (each clone has same number of individuals of P, each of which is a mutation of corresponding one in P). The mutation rate is adaptive based on the fitness of the current individual in P, which is quite similar to the adaptive mutation rate deployed in this work. The fitness for an individual depends on the distance it moves forward and the time it spends.

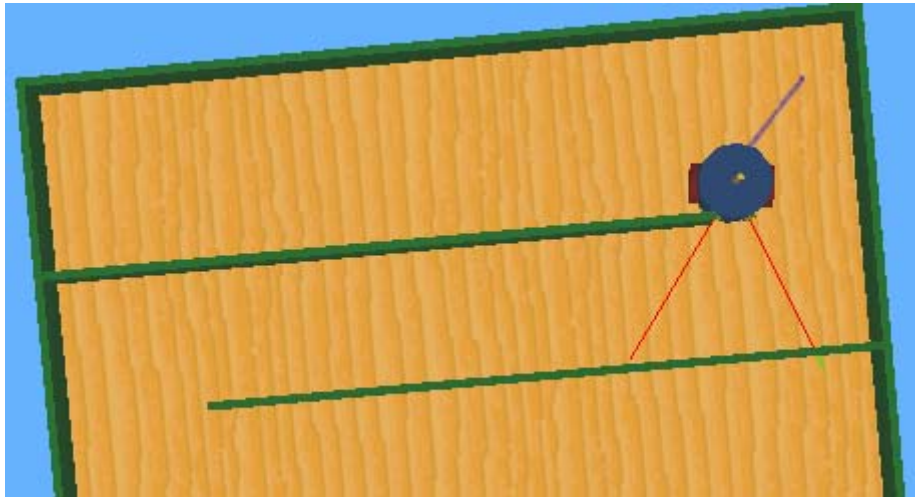


Figure 10-16 The maze controller fails when the wall is not detected

The task can be successfully solved by the proposed method. In addition, fault tolerance was investigated in two cases: when a fault presents before the evolution or it is introduced after an acceptable solution is arrived at. In both cases, the fault introduced is to cover one of the sensors mounted on the robot. It was demonstrated that, if a fault pre-exists before evolution, the method can still discover solutions, although in some cases it may need more time to do so. On the other hand, if a sensor fault is introduced during runtime (after a controller is evolved), the fitness may drop significantly (when one of the front sensor is covered which has more weight for the correct operation of the controller). It is indeed still possible to recover from this kind of fault by restarting the evolution

algorithm to search for new solutions based on current population. It was stated that, after 10 generations, the best individual can regain its original fitness.

Compared to the evolution only approach of Tyrrell et al, the introduction of development in this model brings several advantages over the previous method:

- In order to continue evolution to recover from faults, a population of solutions has to be maintained all the time, which introduces more overhead.
- There has to be a mechanism to constantly detect faults and activate re-evolution when that happens. This requires more resource and power consumption. The detecting mechanism requires human design and itself may be prone to faults.
- In reality, in run time, it is hard or even impossible to evaluate fitness: taking a robot operating in Mars for example, even if a fault is detected, how to re-evaluate an evolved solution? It is not so responsible if trying to navigate the robot with a just evolved solution which may lead the robot direct to a cliff.

On the other hand, in the proposed model, although development introduces overhead to the system as well, once in operation, development makes sure the system is always consistent: there is no need to design extra mechanism neither to detect faults nor to activate any other procedure to recover from them. In essence, development helps us embed fault tolerance right into the functionality.

10.5 Summary

In this chapter, development principle is incorporated with evolution to solve autonomous robot navigation problem: controllers are evolved utilizing the proposed developmental cellular model to control robots in two environments. The controller is built with simple combinational logic gates as basic building blocks.

It was shown that with the help of evolution, the developmental model can successfully discover solutions of robot controllers which are capable of autonomous navigating in the environment and find out the exit. In addition, transient fault tolerant features are also observed from the evolved solutions. In

Chapter 10 Autonomous Robot Controller

the more challenging problem, the maze world, evolved controllers can also successfully navigate in unknown worlds, which shows the adaptive nature of the evolved controllers.

Chapter 11

Conclusions

11.1 Summary and Achievement

With fast pace of development of electronic applications, more robust and adaptive implementations of digital circuits are desired. One potential candidate for new design philosophies is biology developmental principles.

The key to the success of developmental procedures in living creatures is that the whole organism is grown from one single special cell and all the resulting cells have identical genotypes.

When a system can not perform its designed functionality, a system fault occurs. If a fault can be fixed without replacing any part of the system, it is called a transient fault. Most system faults are caused by transient faults according to previous researches. As a result transient faults are targeted in this thesis to improve robustness of a system.

Derived from biological developmental principles, a cellular array of digital cells model was proposed (Chapter 5). Essentially this model is composed of identical cells which only have knowledge about their own local environment, without any predefined global coordination systems. Each cell consists of three components, a Control Unit, a Execution Unit and a Chemical Diffusion model. In order to have a hardware-friendly evolutionary algorithm, multi-breed EA was derived from simulated annealing like EA, called HereBoy.

With the CUs and CDs networks, this model has been utilized in Chapter 6 to evolve solutions for the French Flag problem in software simulation. It has been able to generate structures capable of self-repair, exhibiting extremely high transient fault-tolerant capability. This fault-tolerant ability is not designed consciously, but emerges under the evolution pressures.

The EUs network has been incorporated in Chapter 7 to implement a two bits multiplier. It has been demonstrated that the biological development model proposed in this work can be applied to useful application and the solution

discovered through evolution exhibits the intrinsic highly fault-tolerance feature similar to its living organism counterpart: the best solutions found can virtually tolerate any transient damages.

In Chapter 8, an intrinsic evolvable hardware platform implemented based upon off-shelf FPGA board has been demonstrated. The IEHW platform has been arranged in an “application independent” manner so that it is easy to reuse most of the framework for other applications, more complex ones. The implemented full-IEHW has been able to carry out almost the same evolution algorithm as realized in software with significant accelerated evolution speed.

Followed in the next chapter, adaptive behaviours of the model were investigated and sequential digital system was attempted. 2 and 3 bit RAM units were evolved which also exhibit transient fault tolerant features. However, as the model is not specific designed to fit sequential circuits requirements, the implementation is quite resource consuming.

EUs were employed again in Chapter 10 to evolve autonomous robot controllers. Controllers were successfully evolved in two environment and they can also recovery from transient faults to resume proper autonomous navigating behaviour. In addition, adaptive behaviour was observed where a controller can manoeuvre a robot in an unknown environment.

In summary, the major novel contributions reported in this thesis are:

1. A cellular model inspired by biological development principles is conceived. With the help of Execution Unit which is built-in for each cell, this model can bring useful functionality to development systems, rather than pure pattern formation capacities. Unlike most models proposed before, this model does not rely on any global coordinates of cells to function.
2. Intrinsic fault tolerance can be obtained using this model for a variety of digital systems, from combination to sequential and it can also be embedded in a robot to control its navigation and
3. The model is capable of exhibiting adaptive behaviours according to the environmental signals, demonstrated by the RAM unit and the robot autonomous controller.

Other substantial contributions include:

1. A generic self contained intrinsic evolvable hardware platform is designed and implemented in off-shelf FPGA chips and an evolution engine is demonstrated for evolving 2-bit multipliers;
2. Novel RAM unit designing method is presented with inherit transient fault tolerance and
3. Adaptive and robust robot controllers built with basic combinational logic gates are illustrated, which deploys both development and evolution to provide fault tolerance.

11.2 Further Research Areas

As reported in this thesis, using the development model, several problems and applications are solved and it appears that development can give rise to intrinsic fault tolerance and adaptive behaviours. However, this is with an overhead with regards to hardware resource consumption and computational power. In addition, we only tackle small problems, with robot controller as the most complex one among them. More efforts needs to be invested to research the scalability of development model and whether this can be made to help design the next generation of even more sophisticated systems.

In addition, with regards to the work reported here, the hardware implementation may receive more improvement, such as incorporating adaptive mutation in the IEHW platform.

With the IEHW, we can also carry out more researches about the impacts of different parameters to the evolution outcome.

At present, only the 2-lowest bits of chemicals are used in NSG, ignoring the top 6 bits and new generated chemicals will overwrite previous ones. Effort may be put into the manipulation mechanism of chemicals, such as introducing another kind of chemical behaving as energy and some anisotropic chemicals. It is appreciated if a circuit capable of a particular function is able to grow and mature in a restrained area by its own, rather than just occupying all available resources.

Chapter 11 Conclusions

With regards to the robot controller, learning capacities is considered quite helpful for advanced creatures. The states of the organism may be considered as a memory, which can change based on past experience of navigating history. Another related approach is to pass the current speeds of the two wheels back to the EUs network for history aware navigation.

Another direction our research may head for is co-evolution. Harrison and Foster investigated sorting network topologies utilizing a co-evolutionary approach [63]. It was demonstrated that when evolving with a corresponding “fault” test sequence (in the expense of 50% evaluation performance degradation); more fault tolerant solutions could be found. This kind of co-evolution can be incorporated into the current model to increase the probability of finding robust solutions.

Reference

- [1] Stephen L. Wolfe: "Molecular and Cellular Biology (1993)", Chapter 1, Wadsworth Pub. Co., c1993
- [2] Stephen L. Wolfe: "Molecular and Cellular Biology (1993)", Chapter 2, Wadsworth Pub. Co., c1993
- [3] Julian F. Miller, "Evolving developmental programs for adaptation, morphogenesis, and self-repair" (2003), Seventh European Conference on Artificial Life, Lecture Notes in Artificial Life, Vol. 2801, pp. 256-265
- [4] J. Miller and P. Thomson: "Cartesian genetic programming", Lecture Notes in Computer Science, Vol. 1802, pp. 121-132, Poli, R., Banzhaf, W., Langdon, W.B., Miller, J. F., Nordin, P., Fogarty, T.C., (Eds.)
- [5] D. Levi, "HereBoy: A Fast Evolutionary Algorithm", Proceedings of the 2nd NASA/DoD Evolvable Hardware Workshop, IEEE Computer Society, Los Alamitos, Ca, July 2000
- [6] R. Krohling, Y. Zhou and A. Tyrrell, "Evolving FPGA-based robot controller using an evolutionary algorithm", 1st International Conference on Artificial Immune Systems, Canterbury (Sep 2003)
- [7] Schwefel, H.-P. "Numerical Optimization of Computer Models", Chichester: Wiley (1981)
- [8] EO home page: <http://eodev.sourceforge.net>
- [9] M. Wall, GALib home page: <http://lancet.mit.edu/ga/>
- [10] Evelyn Lutton, Pierre Collet, Jean Louchet, "EASEA Comparisons on Test Functions: GALib versus EO", Artificial Evolution: 5th International Conference, Evolution Artificielle, EA 2001, Le Creusot, France (Oct 2001)
- [11] EASEA Millennium Edition (v0.6c) page: <http://www-rocq.inria.fr/EASEA/>
- [12] Julian F. Miller, "Evolving a self-repairing, self-regulating, French flag organism", GECCO 2004: Genetic and Evolutionary Computation Conference, Seattle, WA, USA, June 26-30, 2004. Proceedings, Part I
- [13] X. Chen and S. L. Hurst, "A comparison of universal logic-module realizations and their application in the synthesis of combinatorial and sequential logic networks", IEEE Transactions on Computers vol. C-31 pp. 140-147, 1982

Reference

- [14] P.H.R. Scholefield, "Shift Registers Generating Maximum-Length Sequences", *Electronic Technology*, 10-1960, pp.389-394
- [15] S.W. Golomb, *Shift Register Sequences*, Holden-Day, San Francisco, 1967
- [16] Peter Alfke, "Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators", Xilinx Application note, XAPP 052 July 7,1996 (Version 1.1)
- [17] H. Ball and F. Hardy, "Effects and detection of intermittent failures in digital systems" 1969 FJCC, AFIPS Conf. Proc., vol. 35, pp.329-335
- [18] Lewontin, Richard, "The Genotype/Phenotype Distinction", *The Stanford Encyclopedia of Philosophy (Spring 2004 Edition)*, Edward N. Zalta (ed.), URL = <<http://plato.stanford.edu/archives/spr2004/entries/genotype-phenotype/>>.
- [19] Ortega, C., Mange, D., Smith, S.L. and Tyrrell, A.M. 'Embryonics: A Bio-Inspired Cellular Architecture with Fault-Tolerant Properties' *Journal of Genetic Programming and Evolvable Machines*, Vol 1, No 3, pp 187-215, July 2000.
- [20] Jackson, A.H. and Tyrrell, A.M. 'Implementing Asynchronous Embryonic Circuits using AARDVArC', 4th NASA Workshop on Evolvable Hardware, Washington, pp 231-240, July 2002
- [21] Canham, R. and Tyrrell, A.M. 'An Embryonic Array with Improved Efficiency and Fault Tolerance', 5th NASA Conference on Evolvable Hardware, Chicago, pp 265-272, July 2003
- [22] D. Dumitrescu, B. Lazzerini, L.C. Jain and A. Dumitrescu 'Evolutionary Computation', 2000 CRC, pp.1-11
- [23] Fogel, D.B.: 'Evolutionary Computation: Toward a New Philosophy of Machine Intelligence', IEEE Press, Piscataway, NJ, 1995
- [24] Zebulum, R.S., Vellasco, M.M.R., and Pacheco, M.A.C.: 'Evolutionary Electronics: Automatic Design of Electronic Circuits', CRC Press, Boca Raton, FL, 2001
- [25] T. Back, U. Harnmel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," *IEEE Trans. Evol. Comput.*, vol. 1, pp. 3-17, Apr. 1997.
- [26] J. H.Holland, "Outline for a logical theory of adaptive systems," *J. Assoc. Comput. Mach.*, vol. 3, pp. 297-314, 1962.
- [27] J. H.Holland, "Adaptation in Natural and Artificial Systems" Ann Arbor, MI: Univ. of Michigan Press, 1975.

Reference

- [28] J. H. Holland and J. S. Reitman, "Cognitive systems based on adaptive algorithms," Pattern-Directed Inference Systems, D. A. Waterman and F. Hayes-Roth, Ed., New York: Academic, 1978.
- [29] L. J. Fogel, "Autonomous automata" Ind. Res., vol. 4, pp. 14-19, 1962.
- [30] L. J. Fogel, "On the organization of intellect", Ph.D. dissertation Los Angeles: University of California, 1964.
- [31] I. Rechenberg, Evolutions strategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Stuttgart, Germany: Frommann-Holzboog, 1973.
- [32] I. Rechenberg, "Evolutions strategie '94," Werkstatt Bionik und Evolutions technik. Stuttgart, Germany: Frommann-Holzboog, vol. 1, 1994.
- [33] H.-P. Schwefel, Evolutions strategie und numerische Optimierung, Dissertation, Germany: Technische Universität Berlin, May 1975.
- [34] H.-P. Schwefel, Evolution and Optimum Seeking., ser. Sixth-Generation Computer Technology Series, New York: Wiley, 1995.
- [35] J. R. Koza, Genetic Programming: On the Programming of Computers by Means of Natural Selection. Cambridge, MA: MIT Press, 1992.
- [36] L. Altenberg, "The evolution of evolvability in genetic programming," Advances in Genetic Programming. Cambridge, MA: MIT Press, pp. 47-74, 1994.
- [37] JR Koza, SH Al-Sakran and LW Jones, "Automated re-invention of six patented optical lens systems using genetic programming", Proceedings of the 2005 conference on Genetic and evolutionary computation, pp.1953-1960
- [38] Christof Teuscher, Daniel Mange, André Stauffer and Gianluca Tempesti, "Bio-inspired computing tissues: towards machines that evolve, grow, and learn", Biosystems, Volume 68, Issues 2-3, February-March 2003, pp. 235-244
- [39] D. B. Fogel, "An introduction to simulated evolutionary optimization," IEEE Trans. Neural Networks, vol. 5, pp. 3-14, Jan. 1994.
- [40] T. Back, U. Harnmel, and H. P. Schwefel, "Evolutionary computation: Comments on the history and current state," IEEE Trans. Evol. Comput., vol. 1, pp. 3-17, Apr. 1997.
- [41] James Hereford and David Gwaltney, 'Design Space Issues for Intrinsic Evolvable Hardware', Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, June 2004, pp. 231-234

Reference

- [42] Hunter,D., ‘Some lessons learned on constructing an automated testbench for evolvable hardware experiments’, Evolutionary Computation, 2004.CEC2004.Congress on, June 2004, pp. 1808–1812S
- [43] X. Yao and T. Higuchi, “Promises and challenges of evolvable hardware”, IEEE Trans. Syst., Man, Cybern.t, C, vol. 29, Feb. 1999.
- [44] H.de Garis, LSL evolvable hardware workshop report, Japan: ATR, Tech. Rep., Oct. 1995.
- [45] A. J.Hirst, “Notes on the evolution of adaptive hardware,” Proc. 2nd Int. Conf. Adaptive Comput. Eng. Design (ACEDC'96), I.Parmee, Ed.
- [46] D Levi, SA Guccione ,“GeneticFPGA: Evolving Stable Circuits on Mainstream FPGA Devices” - Evolvable Hardware, Proceedings of the First NASA/DoD, pp. 12-17, 1999
- [47] Hollingworth, G., Smith, S., and Tyrrell, A.M.: ‘Safe Intrinsic Evolution of Virtex Devices’. Proc. 2nd NASA/DoD Workshop on Evolvable hardware, Silicon Valley, CA, July 2000, pp. 195–202
- [48] Tyrrell, A.M., Krohling, R.A., Zhou, Y. ‘Evolutionary algorithm for the promotion of evolvable hardware’, IEE Proceedings of Computers and Digital Techniques, Volume: 151, Issue: 4, July 2004, pp. 267-275
- [49] Sakanashi, H., Iwata, M., Higuchi, T., “Evolvable hardware for lossless compression of very high resolution bi-level images”, Computers and Digital Techniques, IEE Proceedings, July 2004, Volume: 151, Issue: 4, pp. 277 – 286
- [50] Iwata, M., Kajitani, I., Liu, Y., Kajihara, N., and Higuchi, T., ‘Implementation of a gate-level evolvable hardware chip’, Evolvable systems: from biology to hardware, Lect. Notes Comput. Sci., 2001, 2210, pp. 38–49
- [51] Murakawa, M., Yoshizawa, S., Kajitani, I., Yao, X., Kajihara, N., Iwata, M., and Higuchi, T.: ‘The GRD chip: genetic reconfiguration of DSPs for neural network processing’, IEEE Trans. Comput., 1999, 48, (6), pp. 628–639
- [52] Hereford, J., Pruitt, C.: “Robust sensor systems using evolvable hardware”, Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, pp 161-168, June 24-26, 2004
- [53] Amaral,J.F.M., Amaral,J.L.M., Santini,C., Tanscheit,R., Vellasco,M., Pacheco,M.: “Towards evolvable analog artificial neural networks controllers”, Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, pp 97-103, June 24-26, 2004

Reference

- [54] M Ju, Hui Li, Meng, Hiot Lim and Qi, Cao, “An intrinsic evolvable and online adaptive evolvable fuzzy hardware scheme for packet switching network”, Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, pp 109-112, June 24-26, 2004
- [55] A.Thompson, P.Layzell, and R. S.Zebulum, “Explorations in design space: Unconventional electronics design through artificial evolution”, IEEE Transactions on Evolutionary Computation, this issue, vol. 3, no. 3, pp. 167-196, September 1999.
- [56] Yang Zhang, Smith, S.L., Tyrrell, A.M., “Digital circuit design using intrinsic evolvable hardware”, Evolvable Hardware, 2004, Proceedings, 2004 NASA/DoD Conference, June 24-26, 2004, PP. 55 - 62
- [57] Islam, M., Terao, S., and Murase, K.: ‘Effect of Fitness for the Evolution of Autonomous Robots in an Open-Environment’, Lect. Notes Comput. Sci., 2001, 2210, pp. 171–181
- [58] Islam, M., Terao, S., and Murase, K.: ‘Incremental Evolution of Autonomous Robots for a Complex Task’, Lect. Notes Comput. Sci., 2001, 2210, pp. 181–191
- [59] J. R. Koza, F. H. Bennett, D. Andre, M. A. Keane, and F. Dunlap, “Automated synthesis of analog electrical circuits by means of genetic programming”, IEEE Trans. Evol. Comput., vol. 1, pp. 109-128, July 1997.
- [60] T.Hikage, H.Hemmi, and K.Shimohara, “Evolutionary methods for smoother evolution”, Proc. ICES98, ser. Lecture Notes in Computer Science, vol. 1478, pp. 115-124, 1998.
- [61] Oltean, M., Grosan, C., “Evolving digital circuits using multi expression programming”, Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on, PP.87 - 90, June 24-26, 2004
- [62] Shanthi, A.P., Muruganandam, P., Parthasarathi, R., “Enhancing the development based evolution of digital circuits”, Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, PP.91 - 94, June 24-26, 2004
- [63] Harrison, M.L., Foster, J.A., “Improving the survivability of a simple evolved circuit through co-evolution”, Evolvable Hardware, 2004.Proceedings.2004 NASA/DoD Conference on, PP.24 - 26, June 24-26, 2004

Reference

- [64] André Stauffer, Daniel Mange and Gianluca Tempesti, “Bio-inspired Computing Machines with Self-repair Mechanisms”, *Biologically Inspired Approaches to Advanced Information Technology*, Volume 3853/2006, pp. 128-140, 2006
- [65] Sosnowski, J., “Transient fault tolerance in digital systems”, *Micro*, IEEE, Volume: 14 , Issue: 1, Feb. 1994, ISSN: 0272-1732, pp. 24 – 35,
- [66] X. Castillo, S.R.McConnel and D.P.Siewiorek, “Derivation and calibration of a transient error reliability model”, *IEEE Transactions on Computers*, No.7, July 1982, pp.658-671
- [67] I.Lee, R.K.Iyer and D.Tang, “Error/failure analysis using event logs from fault tolerant systems”, *Proc. Of IEEE FTCS*, 1991, pp. 10-17
- [68] Y.Tamir and M.Trembley, “High performance fault-tolerant VLSI systems using microrollback”, *IEEE Trans. On Comp.* vol. 39, No. 4, April 1990, pp. 548-554
- [69] Ohntishi, K. and Takagi, H., “Feedback model inspired by biological development to hierarchically design complex structure”, *Systems, Man, and Cybernetics*, 2000 IEEE International Conference on, Volume: 5, 8-11 Oct. 2000 PP 3699 - 3704 vol.5
- [70] Gomaa, M.,Scarborough, C., Vijaykumar, T.N., Pomeranz, I., “Transient-fault recovery for chip multiprocessors”, *Computer Architecture, Proceedings. 30th Annual International Symposium on*, pp. 98- 109, 2003
- [71] P.Marchal, A.Stauffer, “Binary Decision Diagram Oriented FPGAs”, *Proc. of the ACM International Workshop on Field-Programmable Gate Arrays*, Berkeley, February 1994
- [72] Katsinis, C. and Hecht, D., “Fault-Tolerant Distributed Shared Memory on a Broadcast-Based Architecture”, *Parallel and Distributed Systems*, IEEE Transactions on, Dec. 2004, pp. 1082 – 1092, Volume: 15, Issue: 12, ISSN: 1045-9219
- [73] Myung-Hyun Lee, Young Kwan Kim and Yoon-Hwa Choi, “A defect-tolerant memory architecture for molecular electronics”, *Nanotechnology*, IEEE Transactions on, March 2004, pp. 152 – 157, Volume: 3, Issue: 1, ISSN: 1536-125X

Reference

- [74] Naden, R. and West, F., "Fault-tolerant memory organization: Impact on chip yield and system cost", *Magetics, IEEE Transactions on*, Sep 1974, pp. 852 - 855
Volume: 10, Issue: 3, ISSN: 0018-9464
- [75] Mazumder, P., "Design of a fault-tolerant three-dimensional dynamic random-access memory with on-chip error-correcting circuit", *Computers, IEEE Transactions on*, Dec. 1993, pp. 1453 – 1468, Volume: 42, Issue: 12, ISSN: 0018-9340
- [76] CS & BB, Celoxica ltd., 'RC1000 Hardware reference Manual', Version 2.3, Document number: RM-1120-0
- [77] Xilinx ltd., 'Virtex™ 2.5 V Field Programmable Gate Arrays', DS003-1 (v2.5), April 2, 200, Product Specification
- [78] Beagle 2's Official Site: <http://www.beagle2.com/>
- [79] ESA/UK Commission of Inquiry report. (PDF file):
<http://www.bnsc.gov.uk/assets/channels/resources/press/report.pdf>
- [80] Clark LD, Clark RK, Heber-Katz E, "A new murine model for mammalian wound repair and regeneration". *Clin Immunol Immunopath* 88 on 1998, pp 35-45
- [81] Michalopoulos GK, DeFrances MC, "Liver regeneration", *Science*, vol. 276, pp. 60, 1997.
- [82] J.P.Brockes, "Amphibian Limb Regeneration: Rebuilding a Complex Structure", *Science* vol. 276, no 5309. pp. 81-87, 1997
- [83] Endler JA, "Natural Selection in the Wild", Princeton, New Jersey: Princeton University Press, ISBN 0-691-00057-3, 1986
- Williams GC (1966). *Adaptation and Natural Selection*. Oxford University Press.
- [84] Sober E, "The Nature of Selection: Evolutionary Theory in Philosophical Focus", University of Chicago Press, ISBN 0-226-76748-5, 1993
- [85] "Understanding Evolution", University of California, Berkeley, available online at http://evolution.berkeley.edu/evolibrary/article/0_0_0/evo_17 and http://evolution.berkeley.edu/evolibrary/article/0_0_0/evo_16
- [86] Reznick DN, Shaw FH, Rodd FH, Shaw RG, "Evaluation of the Rate of Evolution in Natural Populations of Guppies (*Poecilia reticulata*)", *Science*, vol. 275, pp. 1934-1937, 1997
- [87] Douglas J. Futuyma, "Evolutionary biology", Sinauer Associates, c1998

Reference

- [88] C. Teuscher, "Turing's Connectionism. An investigation of neural network architectures", Heidelberg: Springer-Verlag, 2002
- [89] Hugo de Garis, "Evolvable Hardware: Principles and Practice", CACM Journal, Communications of the Association for Computer Machinery (CACM Journal), , August 1997
- [90] T. Higuchi et al., "Evolving Hardware with genetic learning: A first step towards building a Darwin machine", Proc. Of the 2nd Intl. Conf. on Simulation of Adaptive Behaviour, J-A. Meyer, H. Roitblat, and S. Wilson, Eds. Cambridge, MA: MIT Press-Bradford Books, 1993, pp. 417-424
- [91] S. Kirkpatrick, C. D. Gelatt and M. P. Vecchi "Optimization by Simulated Annealing", Science, Vol 220, Number 4598, pages 671-680, 1983
- [92] V. Cerny, "A thermodynamical approach to the travelling salesman problem: an efficient simulation algorithm", Journal of Optimization Theory and Applications, 45:41-51, 1985
- [93] T. Bäck and H.-P. Schwefel, "An overview of evolutionary algorithms for parameter optimization", Evolutionary Computation 1(1), pp. 1-23, 1993
- [94] J. Torresen, "Possibilities and limitations of applying evolvable hardware to real-world application", in 10th international Conference on Field Programmable Logic and Applications (FPL-2000), 2000
- [95] Koza, J.R., Keane, M.A., Streeter, M.J., "Routine high-return human-competitive evolvable hardware", Evolvable Hardware, 2004. Proceedings. 2004 NASA/DoD Conference on, pp. 3-17
- [96] S. A. Kauffman, "Metabolic stability and epigenesis in randomly constructed genetic nets", J. Theoret. Biol 22, 434-467, 1969
- [97] Anna Gambin, Sławomir Lasota and Michał Rutkowski, "Analyzing stationary states of gene regulatory network using Petri nets", Silico Biology 6, 0010, 2006
- [98] H. Matsuno, A. Doi, M. Nagasaki and S. Miyano. "Hybrid Petri Net Representation of Gene Regulatory Network". In Pacific Symposium on Biocomputing, World Scientific Press, pages 341-352, 2000.
- [99] Min Zou and Suzanne D. Conzen, "A new dynamic Bayesian network (DBN) approach for identifying gene regulatory networks from time course microarray data", Bioinformatics 21(1), pp. 71-79, 2005

Reference

- [100] Xiaobo Zhou, Xiaodong Wang, Ranadip Pal, Ivan Ivanov, Michael Bittner and Edward R. Dougherty, “A Bayesian connectivity-based approach to constructing probabilistic gene regulatory networks”, *Bioinformatics* Vol. 20 no. 17, pp. 2918-2927, 2004
- [101] Ilya Shmulevich, Edward R. Dougherty, Seungchan Kim and Wei Zhang, “Probabilistic Boolean networks: a rule-based uncertainty model for gene regulatory networks”, *Bioinformatics* Vol. 18 no. 2, pp. 261-274, 2002
- [102] Schäfer, J. and K. Strimmer, “Learning large-scale graphical Gaussian models from genomic data”, *AIP Conference Proceedings* 776, *Science of Complex Networks: From Biology to the Internet and WWW*, Aveiro, PT, pp. 263-276, 2004.
- [103] Tetsuya MATSUNO, Nobuaki TOMINAGA, Koji ARIZONO, Taisen IGUCHI and Yuji KOHARA, “Graphical Gaussian Modeling for Gene Association Structures Based on Expression Deviation Patterns Induced by Various Chemical Stimuli”, *IEICE Transactions on Information and Systems* 2006 E89-D(4):1563-1574
- [104] X. Wu, Y. Ye, K. Subramanian, L. Zhang, “Interactive Analysis of Gene Interactions Using Graphical Gaussian Model”, *Proceedings of the 3rd ACM SIGKDD Workshop on Data Mining in Bioinformatics*
- [105] Raffaella Gentilini, “Toward Integration of Systems Biology Formalism: The Gene Regulatory Networks Case”, *Genome Informatics* 16(2), pp. 215–224, 2005
- [106] Cardelli, L., “A compositional approach to the stochastic dynamics of gene networks”, In *Lecture Notes in Computer Science*, Springer-Verlag, London, pp. 4-4, 2005
- [107] M.J.L. de Hoon, S. Imoto, K. Kobayashi, N. Ogasawara and S. Miyano, “Inferring gene regulatory networks from time-ordered gene expression data of *Bacillus subtilis* using differential equations”, *Pacific Symposium on Biocomputing*, 8, 17-28, 2003.
- [108] A Lindenmayer, “Mathematical models for cellular interactions in development”, *Journal of Theoretical Biology*, vol. 18, pp. 280-299, 1968
- [109] The original image is from http://en.wikipedia.org/wiki/L_Systems

Reference

- [110] C. Gershenson, "Classification of random Boolean networks", in *Artificial Life VIII: Proceedings of the Eight International Conference on Artificial Life*, R. K. Standish, M. A. Bedau, and H. A. Abbass, Eds., MIT Press, pp 1-8, 2002
- [111] Timothy G. W. Gordon, Peter J. Bentley, "Towards Development in Evolvable Hardware", *Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware (EH'02)*, pp. 241-250, July 15-18, 2002
- [112] Timothy G. W. Gordon, "Exploring models of development for evolutionary circuit design", in *Congress on Evolutionary Computation (CEC2003)*, IEEE Press, 2003, pp. 2050-2057
- [113] Koopman, A. and Roggen, D., "Evolving Genetic Regulatory Networks for Hardware Fault Tolerance", *Proceedings of Parallel Problem Solving from Nature 8*, pp. 561-570, 2004
- [114] Bentley, P. J. "Adaptive Fractal Gene Regulatory Networks for Robot Control", In Miller, J. (Ed.) *Workshop on Regeneration and Learning in Developmental Systems, Genetic and Evolutionary Computation Conference (GECCO 2004)*, 2004
- [115] Pauline C. Haddow, Gunnar Tufte, "Bridging The Genotype-Phenotype Mapping For Digital Fpgas," eh, p. 0109, *The Third NASA/DoD Workshop on Evolvable Hardware*, 2001
- [116] Mandelbrot, B., "The Fractal Geometry of Nature", W.H. Freeman & Company, 1982
- [117] P.C. Haddow, G. Tufte, and P. van Remortel, "Shrinking the Genotype: L-systems for EHW?," *The 4th Int. Conf. on Evolvable Systems: From Biology to Hardware*, Tokyo, Japan, 2001
- [118] Gunnar Tufte and Pauline Catriona Haddow, "Biologically-Inspired: A Rule-Based Self-Reconfiguration of a Virtex Chip", *Proc. of International Conference on Computational Science 2004*, 2004
- [119] Tufte, G. and Haddow, P.C., "Building knowledge into developmental rules for circuit design", In: *Proc. 5th Intl. Conf. on Evolvable Systems, ICES'2003. Volume 2606 of LNCS.*, Springer-Verlag, pp. 69-80, 2003
- [120] J. F. Miller and P. Thomson, "A Developmental Method for Growing Graphs and Circuits", *Fifth International Conference on Evolvable Systems: From Biology to Hardware*, Trondheim, March 17-20, 2003, *Proceedings published as Lecture Notes in Computer Science, Vol. 2606*, pp. 93-104

Reference

- [121] Mizoguchi, Junichi, Hemmi, Hitoshi, and Shimohara, Katsunori.,
“Production genetic algorithms for automated hardware design through an
evolutionary process”, Proceedings of the First IEEE Conference on
Evolutionary Computation. IEEE Press, vol. I. pp. 661-664, 1994
- [122] Hemmi H., Mizoguchi J and Shimohara K., “Evolving Large Scale Digital
Circuits”, Proc. of the 5th Int. Forkshop on the Synthesis and Simulation of
Living Systems”, Nara, Japa2, pp. 168-173, 1996
- [123] J. R. Koza, “Genetic Programming”, Cambridge, MA: MIT Press, 1992
- [124] J. R. Koza, F. H. Bennett III, D. Andre and M. A. Keane, “Reuse,
parameterized reuse, and hierarchical reuse of substructures in evolving electrical
circuits using genetic programming”, in Proc. of the 1st Int. Conf. on Evolvable
Systems (ICES 96), T. Higuchi et al., Eds. Berlin: Springer-Verlag, 1996, pp.
312-326
- [125] J. R. Koza, M. A. Keane, M. J. Streeter, W. Mydlowec, J. Yu, and G.
Lanza, “Genetic Programming IV: Routine Human-Competitive Machine
Intelligence”, Kluwer Academic Publisher, 2003
- [126] Jason D. Lohn, Silvano P. Colombano, “Automated Analog Circuit
Synthesis Using a Linear Representation”, Lecture Notes in Computer Science,
Volume 1478, Jan 1998, pp. 125-133, 1998
- [127] Lohn, J.D.; Colombano, S.P., “A circuit representation technique for
automated circuit design,” Evolutionary Computation, IEEE Transactions on ,
vol.3, no.3, pp. 205-219, Sep 1999
- [128] Mattiussi, C., and Floreano, D., “Evolution of Analog Networks using
Local String Alignment on Highly Recognizable Genomes”, in Proceedings of
the 2004 NASA/DoD Conference on Evolvable Hardware, 2004
- [129] Gordon, T.G.W., Bentley, P.J., “Development brings scalability to
hardware evolution”, Evolvable Hardware 2005, Proceedings, 2005 NASA/DoD
Conference on 29 June-1 July 2005, pp. 272-279, 2005
- [130] P. Marchal et al., “Embryological development on silicon”, in Artificial
Life IV, R. Brooks and P. Maes, Eds: MIT Press, pp. 365-366, 1994
- [131] D. Mange, M. Goeke, D. Madon, A. Stauffer, G. Tempesti, and S. Durand,
“Embryonics: A new family of coarse-grained field-programmable gate array
with self-repair and self-reproducing properties”, in Towards Evolvable

Reference

Hardware, New York: Springer-Verlag,, vol. 1062 of Lecture Notes Comput. Sci., pp. 197-220, 1996

[132] D. Mange, M. Sipper, A. Stau#er, and G. Tempesti, "Toward robust integrated circuits: the embryonics approach", in IEEE, 88, no. 4, pp. 516-541, 2000

[133] Charles Darwin and Alfred Wallace, "On the Tendency of Species to form Varieties; and on the Perpetuation of Varieties and Species by Natural Means of Selection", Journal of the Proceedings of the Linnean Society, Zoology 3: pp. 45-62, 1858

[134] Spears, William M., "Crossover or Mutation?", In Proceedings of Foundations of Genetic Algorithms Workshop, pp. 221-237, 1992

[135] Back, T., "The interaction of mutation rate, selection, and self-adaptation within a genetic algorithm", Parallel Problem Solving from Nature, II, pp. 85-94, 1992

[136] Thomas Bäck, "Optimal Mutation Rates in Genetic Search", Proceedings of the 5th International Conference on Genetic Algorithms, pp. 2-8, 1993

[137] Thomas Back and M. Schutz, "Intelligent mutation rate control in canonical genetic algorithms", Proceedings of the 9th International Symposium, ISMIS 96, Zakopane (Poland), Springer-Verlag, Berlin (Germany), pp. 158-167, 1996

[138] Fogarty T., "Varying the Probability of Mutation in the Genetic Algorithm", Proc. of the Third International Conference on Genetic Algorithms, pp. 104-109, Morgan Kaufmann, 1989

[139] Jurgen Hesser and Reinhard Manner, "Towards an optimal mutation probability for genetic algorithms", In Parallel Problem Solving from Nature, pp. 23-32, 1990

[140] Thomas Jansen, Ingo Wegener, "On the Choice of the Mutation Probability for the (1+1) EA", Proc. of the 6th Parallel Problem Solving from Nature, Springer, pp.89-98, 2000

[141] Muhlenbein, H., "How genetic algorithms really work. i. mutation and hillclimbing", In: Proc. of PPSN II, pp. 15-25, 1992

[142] G. Ochoa, I. Harvey, and H. Buxton, "On recombination and optimal mutation rates", In Proceedings of Genetic and Evolutionary Computation Conference (GECCO-99), Morgan Kaufmann, pp. 488-495, 1999

Reference

- [143] J. Miller, "On the filtering properties of evolved gate arrays", the first NASA/DoD Workshop on Evolvable Hardware, pages 2-11, 1999.
- [144] Miller, J.F., D. Job, and V.K. Vassilev, "Principles in the Evolutionary Design of Digital Circuits - Part I", Genetic Programming and Evolvable Machines, 1(1): pp. 7-35, 2000
- [145] B. Rajan and S.Ravi, "FPGA Based Hardware Implementation of Image Filter With Dynamic Reconfiguration Architecture", IJCSNS International Journal of Computer Science and Network Security, VOL.6 No.12, December 2006
- [146] T. Higuchi, M. Iwata, I. Kaijitani, M. Murakawa, S. Yoshizawa, and T. Furuya, "Hardware evolution at gate and function level", in Proc. Int. Conf. Biologically Inspired Autonomous Syst.: Computation, Cognition Action, Durham, NC, Mar. 4-5, 1996
- [147] Masiero, L.P., Pacheco, M.A.C., Barbosa, C.R.H.; Santini, C.C., "Molecular circuit design", Evolvable Hardware, 2005. Proceedings. 2005 NASA/DoD Conference on, pp. 307 – 312, 2005
- [148] Gunnar Tufte and Pauline C. Haddow, "Towards Development on a Silicon-based Cellular Computing Machine", Natural Computing, pp.387-416, September, 2005
- [149] Gunnar Tufte and Pauline C. Haddow, "Extending Artificial Development: Exploiting Environmental Information for the Achievement of Phenotypic Plasticity", Evolvable Systems: From Biology to Hardware, pp. 297-308, 2007
- [150] Kumar, P.N., Suresh, S., Perinbam, J.R.P., "Digital image filter design using evolvable hardware", Computer and Information Science, 2005. Fourth Annual ACIS International Conference on, pp. 483-488, 2005
- [151] Parag K. Lala, "Self-Checking and Fault-Tolerant Digital Design", Morgan Kaufmann Publishers, 2001
- [152] Baron R. and Higbie L., "Computer Architecture", Addison-Wesley, 1992
- [153] Anderson, T., and Lee, P., "Fault Tolerance: Principles and Practices", Prentice Hall International, 1981
- [154] Proceedings of the First Symposium on Large-Scale Digital Calculating Machinery, Harvard University Press, 1948

Reference

- [155] "Type 650 Magnetic Drum Data-Processing Machine - Manual of Operation," first revision, Form 22-6060-1, IBM, 590 Madison Avenue, New York 22, NY (June 1955)
- [156] Sperry Rand Corporation, "UNIVAC 1110 System Description", UP-7841, 1970
- [157] Redmond, Kent C.; Thomas M. Smith, "Project Whirlwind: The History of a Pioneer Computer", Bedford, MA: Digital Press, ISBN 0-932376-09-6, 1980
- [158] Avizienis A., "Fault-Tolerance - The survival attribute of digital systems", Procs. IEEE, Vol.66, Num.10, Oct., 1978
- [159] Von Neumann J., "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, Shannon C. and McCarthy J. (eds.), Annals of Math Studies, Num.34, Princeton Univ. Press, pp.43-98, 1956
- [160] Chen, L. and A. Avizienis, "N-version programming: A fault tolerance approach to reliability of software operation", Digest 8th Int. Symp. Fault-Tolerant Computing, pp. 3-9, 1978
- [161] Randell, B., "Fault tolerant computing system", 6th School of Computing, European Organization for Nuclear Research, pp. 362-389, September 1980
- [162] Hecht, H., "Fault Tolerant software", IEEE Trans, Reliability, pp. 227-232, August 1979
- [163] Hopkins, A. L., "A fault-tolerant information processing concept for space vehicles", IEEE Trans. Computers, C-20, pp. 1394-1403, November 1971
- [164] Avizienis, A., et al. "The STAR (self testing and repairing) computer: An investigation of the theory and practice of fault-tolerant computer design", IEEE Trans. Computers, pp. 1312-1321, November 1971
- [165] Wensley, J.H., et al. "SIFT: Design and analysis of a fault-tolerant computer for aircraft control", Proc. IEEE 66, pp. 1240-1255, October 1978
- [166] R. Dawkins, "The selfish gene", Oxford, Oxford University Press, 1999
- [167] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, G. Klimeck, Y. Jin, R. Tawel, V. Duong, "Evolution of Analog Circuits on Field Programmable Transistor Arrays," eh, p. 99, The Second NASA/DoD Workshop on Evolvable Hardware (EH'00), 2000

Reference

- [168] Keymeulen, D., Zebulum, R.S., Jin, Y. and Stoica, A., "Fault-tolerant evolvable hardware using field-programmable transistor arrays", *Reliability, IEEE Transactions on*, Volume: 49, Issue: 3, pp. 305-316, Sep 2000
- [169] Shanthi, A.P. and Parthasarathi, R., "Exploring FPGA structures for evolving fault tolerant hardware", *Evolvable Hardware*, 2003, Proceedings, NASA/DoD Conference on, pp.174-181, 9-11 July 2003
- [170] Stauffer, A., Mange, D., Tempesti, G. and Teuscher, "A Self-Repairing and Self-Healing Electronic Watch: The BioWatch", In *Proceedings of the 4th international Conference on Evolvable Systems: From Biology To Hardware*, Lecture Notes In Computer Science, vol. 2210. Springer-Verlag, London, pp. 112-127, 2001
- [171] Perez, A. and Sanchez, E., "The FAST architecture: a neural network with flexibleadaptable-size topology", *Microelectronics for Neural Networks*, Proceedings of Fifth International Conference on, pp. 337-340, 1996
- [172] Nolfi, S. and Parisi, D., "Learning to adapt to changing environments in evolving neural networks", Technical Report, Institute of Psychology, National Research Council, Rome, pp. 95-15, 1995
- [173] D. Floreano and J. Urzelai, "Evolution of plastic control networks", *Autonomous Robots*, vol. 11, pp. 311-317, 2001
- [174] Bradley, D., Ortega-Snchez, C. and Tyrrell, A., "Embryonics + immunotronics: a bio-inspired approach to fault tolerance", *Proceedings of 2nd NASA/DoD Workshop on Evolvable Hardware*, pp. 215-233, Lohn, J. et al. (eds.), IEEE Computer Society, 2000
- [175] S. H. Fuller and S. P. Harbison, "The C.mmp multiprocessor", Carnegie-Mellon Univ., Dep. Comput. Sci., Pittsburgh, PA, Tech. Rep., Oct. 1978.
- [176] M. Morganti, G. Coppadoro, and S. Ceru, "UDET 7116-Common control for PCM telephone exchange-Diagnostic software design and availability evaluation," in *Proc. FTCS-8*, 178, pp. 16-23.
- [177] M. Morganti, Private communication, 1978
- [178] M. Geihufe, "Soft errors in semiconductor memories", in *Dig. COMPCON Spring*, 1979, pp. 210-216.
- [179] V. J. Ohm, "Reliability considerations for semiconductors memories," in *Dig. COMPCONSpring*, 1979, pp. 207-209.

Reference

- [180] D. P. Siewiorek, V. Kini, H. Mashburn, S. R. McConnel, and M. M. Tsao, "A case study of C.mmp, Cm*, and C.vmp-PartI-Experiences with fault tolerance in multiprocessor systems," *Proc. IEEE*, vol. 66, pp. 1178-1199, Oct. 1978.
- [181] N. Cohen et. al., "Soft error considerations for deep-submicron CMOS circuit applications", *Technical Digest of International Electronic Devices Meeting (IEDM)*, pp.315-318, 1999
- [182] R. Iyer and D. Rosetti, "A statistical load dependency of CPU errors at SLAC," *Proceedings of FTCS*, 1982
- [183] Needham, J., "Science and Civilisation in China: Volume 2, History of Scientific Thought", Cambridge University Press, 1991, ISBN 0521058007
- [184] Autonomous robot, http://en.wikipedia.org/wiki/Autonomous_robot, From Wikipedia
- [185] Tagkopoulos, I., Zukowski, C., Cavelier, G., and Anastassiou, D., "A custom FPGA for the simulation of gene regulatory networks", In *Proceedings of the 13th ACM Great Lakes Symposium on VLSI (Washington, D. C., USA, April 28 - 29, 2003)*, GLSVLSI '03, ACM Press, New York, NY, pp.132-135. 2003
- [186] Gianluca Tempesti, Daniel Roggen, Eduardo Sanchez, Yann Thoma, 'A POetic Architecture for Bio-Inspired Hardware', *Proc. 8th Intl. Conf. on the Simulation and Synthesis of Living Systems (Artificial Life VIII)*, Sydney, Australia, 9-13 Dec. 2002. MIT Press, Cambridge, MA, 2002, pp. 111-115
- [187] J.-M. Moreno, Y. Thoma, E. Sanchez, O. Torres, G. Tempesti, 'Hardware Realization of a Bio-inspired POetic tissue', *Proc. 2004 NASA/DoD Conference on Evolvable Hardware*, pp. 237-244, IEEE Computer Society, Los Alamitos, 2004
- [188] The Khepera Miniature Mobile Robot Specification, <http://diwww.epfl.ch/lami/robots/K-family/Khepera.html>
- [189] Miller J. F. "What bloat? Cartesian Genetic Programming on Boolean problems", *Genetic and Evolutionary Computation Conference*, Late breaking paper (2001) 295 - 302
- [190] Melhem, R., Mosse, D., Elnozahy, E., "The interplay of power management and fault recovery in real-time systems", *Transactions on Computers*, pp. 217-231, Volume: 53, Issue 2, 2004

Reference

- [191] Seong Woo Kwak, Byung Jae Choi, Byung Kook Kim, “An optimal checkpointing-strategy for real-time control system sunder transient faults”, Reliability, IEEE Transactions on, pp. 293-301, Volume: 50, Issue 3, 2001
- [192] Prvulovic, M., Zheng Zhang, Torrellas, J., “ReVive: cost-effective architectural support for rollback recovery in shared-memory multiprocessors”, Computer Architecture, 2002. Proceedings. 29th Annual International Symposium on, pp. 111-122, 2002
- [193] Koza, John R., “Genetic Programming: on the programming of computers by means of natural selection”, MIT press, Cambridge London (1992)
- [194] WOLPERT, L., “Principles of Development”, 2nd ed. Oxford University Press, Oxford, UK, 2002

Appendix

Appendix A Schematics of Circuits for multiplier

These three sub circuits were found in the experiments described in Chapter 7.

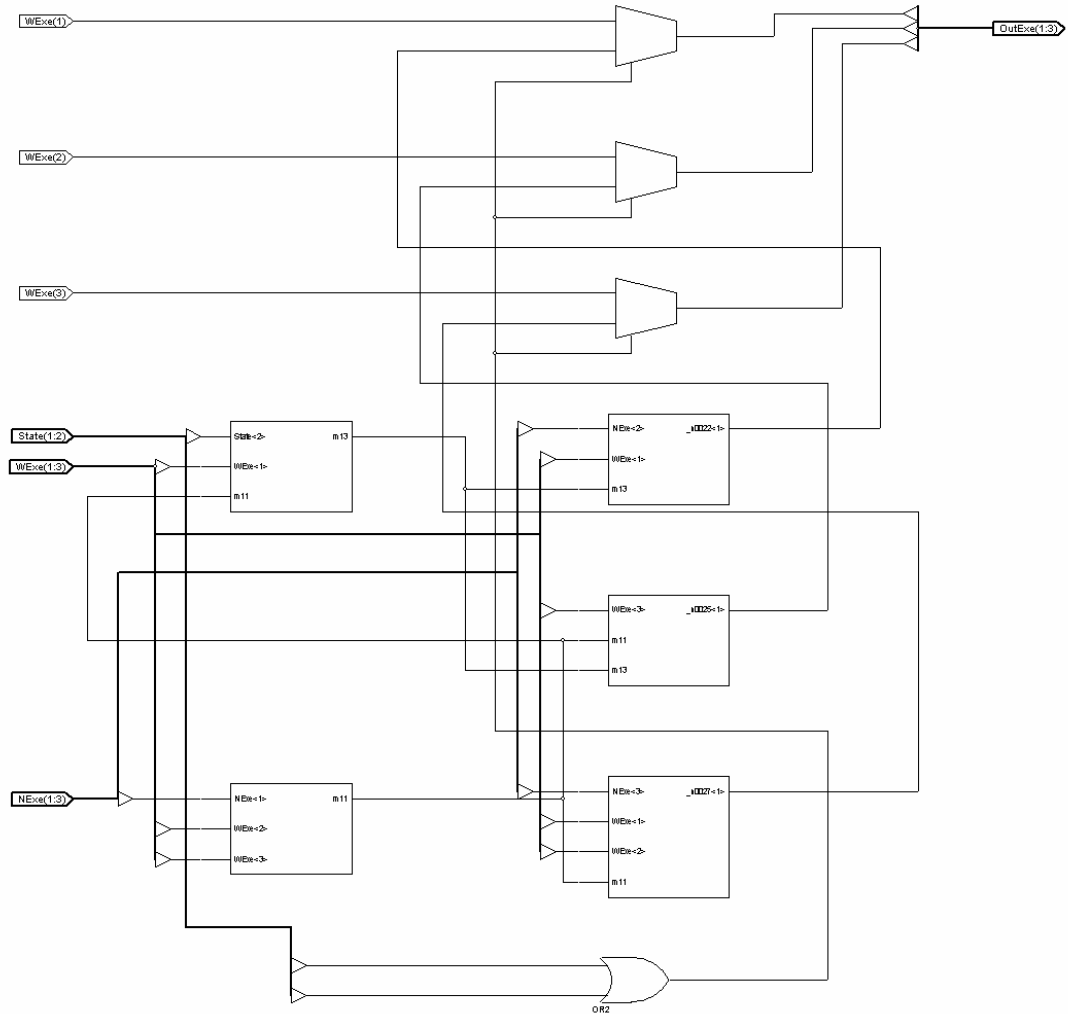


Figure A-1 Execution Unit Core Structure

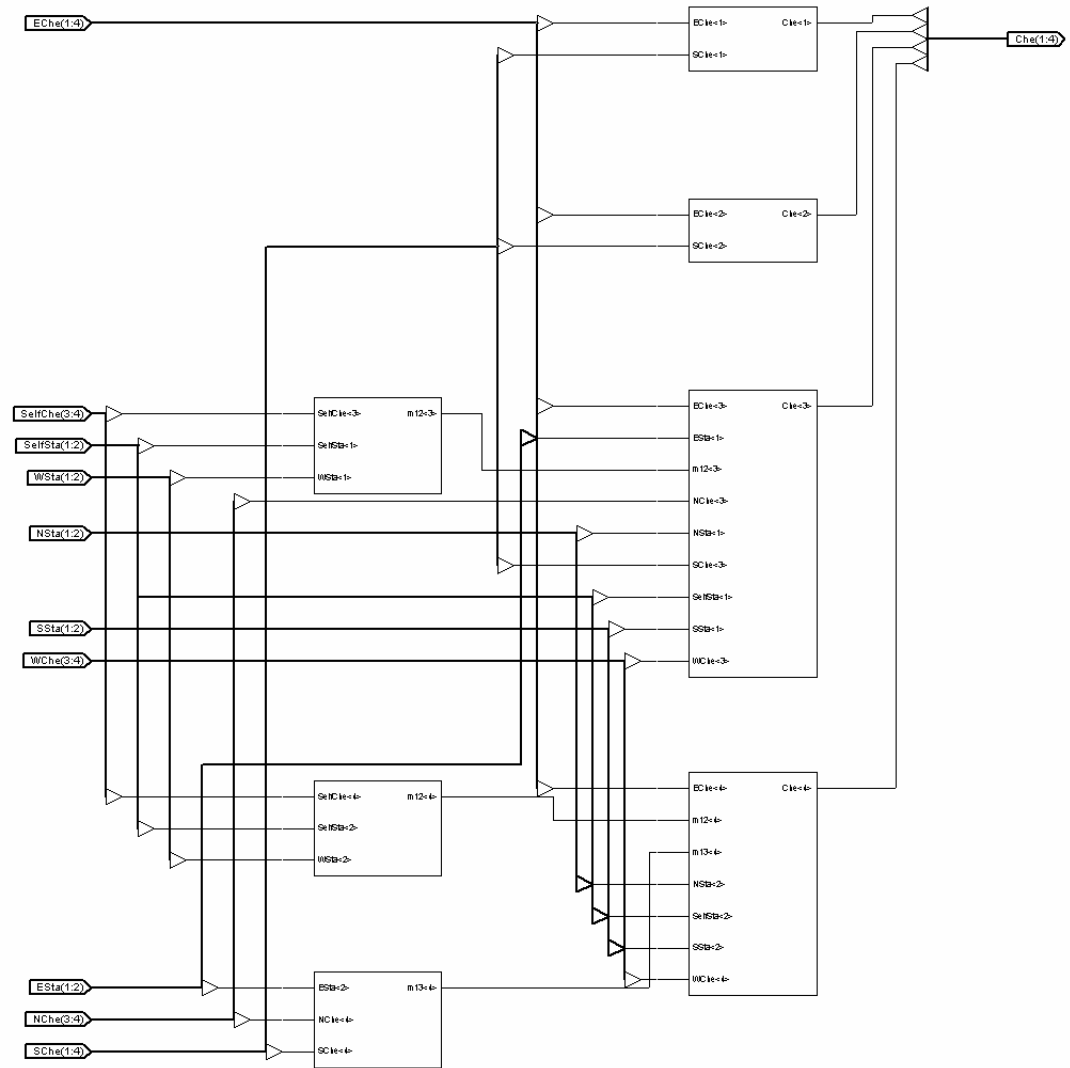


Figure A-2 Next Chemical Generator Structure

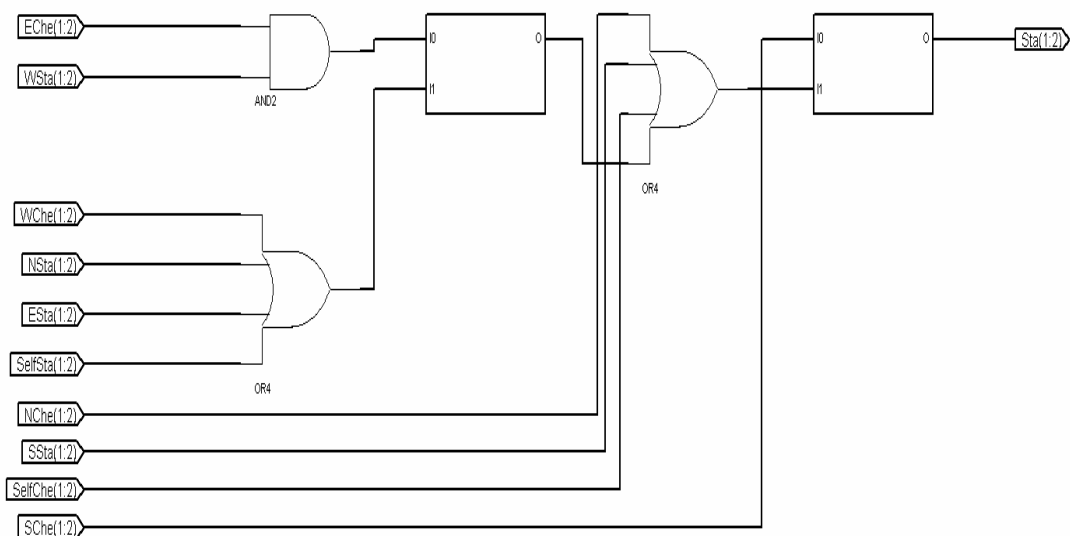


Figure A-3 Next State Generator Structure

Appendix B RC1000 Board from Celoxica

The RC1000 is a PCI bus plugin board for Desktop Computer. It consists of a large Xilinx Virtex FPGA with four banks of RAM for data processing operations, plus two PMC sites for I/O with the outside world. [77]

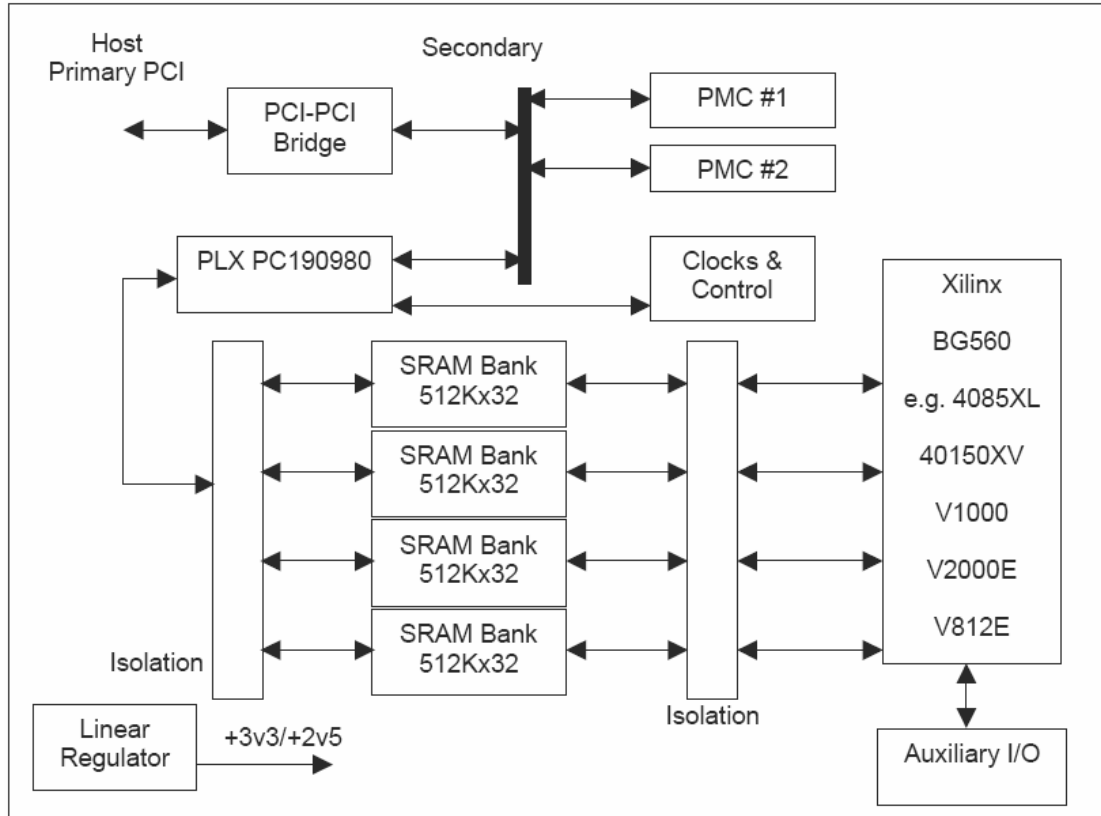


Figure B-1 RC1000 Board Block Diagram [77]³⁴

The block diagram of the board is presented in Figure B-1. Four memory banks are installed in this board, each of which is 2Mbytes and accessible by both of the FPGA and any device on the PCI bus (including host PC).

PLX PCI9080 PCI controller chip on the board delivers three methods of transferring data or communicating between the FPGA and any other PCI devices: Memory banks can act as the bridge in bulk data transfer mode using DMA; Two unidirectional 8 bits ports are allocated in the FPGA for direct communications between the FPGA and the PCI bus; two PLX user I/O pins are provided for single bit communications.

³⁴ The board used in this research is installed with a Virtex V1000 FPGA.

Appendix

The FPGA can use four clock resources, two programmable clocks, an external one and the PCI9080 local bus clock, selectable by a jumper on the board. The programmable clocks can be configured by the host PC to be in the range of 400 kHz to 100 MHz.

The installed FPGA in the board used in this work is Virtex V1000 from Xilinx Ltd (details are given in the next appendix section)

Appendix C XCV1000 FPGA from Xilinx³⁵

The internal structure of Virtex FPGA is shown in Figure C-1. Two kinds of re-programmable components are available: configurable logic blocks (CLBs) and input/output blocks (IOBs).

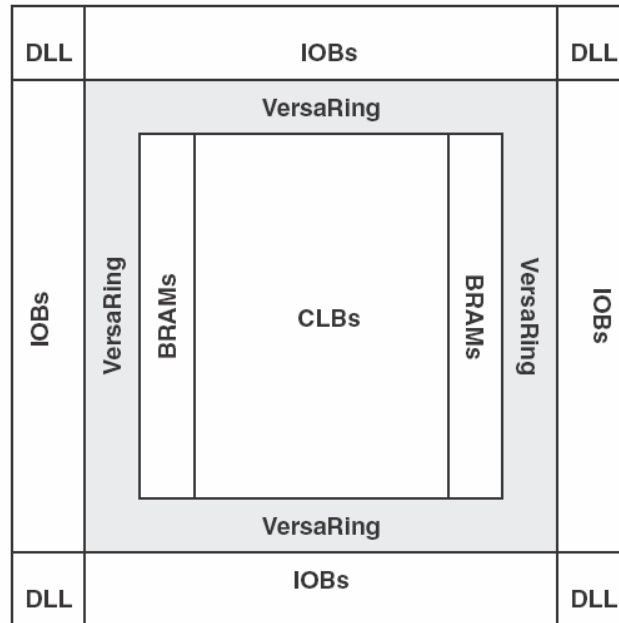


Figure C-1 Virtex Architecture Overview

A general routing matrix (GRM) is utilized to interconnect CLBs. Besides the CLBs, other three types of components are connected to GRM: Dedicated block memories, 3-State buffers, Clock delay-locked loops (DLLs).

The FPGA is re-configured in the means of modifying the values stored in static memory cells which control the configurable components, including logic elements and interconnect resources.

The Virtex IOB, which comprises three storage elements (configurable either as edge-triggered D-type flip-flops or as level sensitive latches), support a wide variety of I/O signalling standards.

Each Virtex CLB contains two slices (refer to Figure C-2), each of which contains two logic cells (LC). The basic building block of CLB is LC, which

³⁵ The content of this appendix section is based on [76] and [77]

Appendix

includes a 4-input function generator, carry logic, and a storage element. In addition, each CLB contains combination logics to provide five/six inputs function.

The implementation of function generators is 4-input look-up tables (LUTs). LUTs can also be configured to be 16 x 1-bit synchronous RAMs or 16-bit shift registers. Furthermore, a 16 x 2-bit or 32 x 1-bit synchronous RAM can be implemented by combining the two LUTs in the same slice.

The two storage elements in the Virtex slice can be configured as edge-triggered D-type flip-flops or as level-sensitive latches. The initial values saved in the D flip-flops can also be configured.

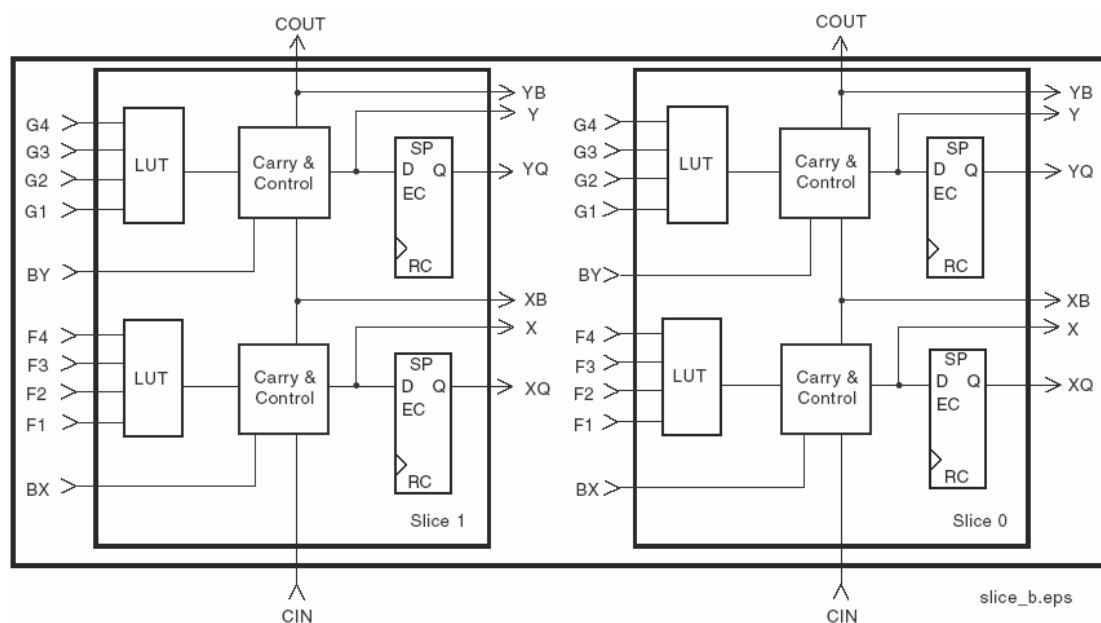


Figure C-2 Structure of Virtex CLB

Dedicated carry logic are supplied in the Virtex CLB to accelerate arithmetic functions.

3-state drivers are implemented in each CLB to drive on-chip buses.

Large block memories, called SelectRAM, are incorporated to complement the distributed LUT memories. Each SelectRAM block cell is 4096-bit and can be configured to 1/2/4/8/16.

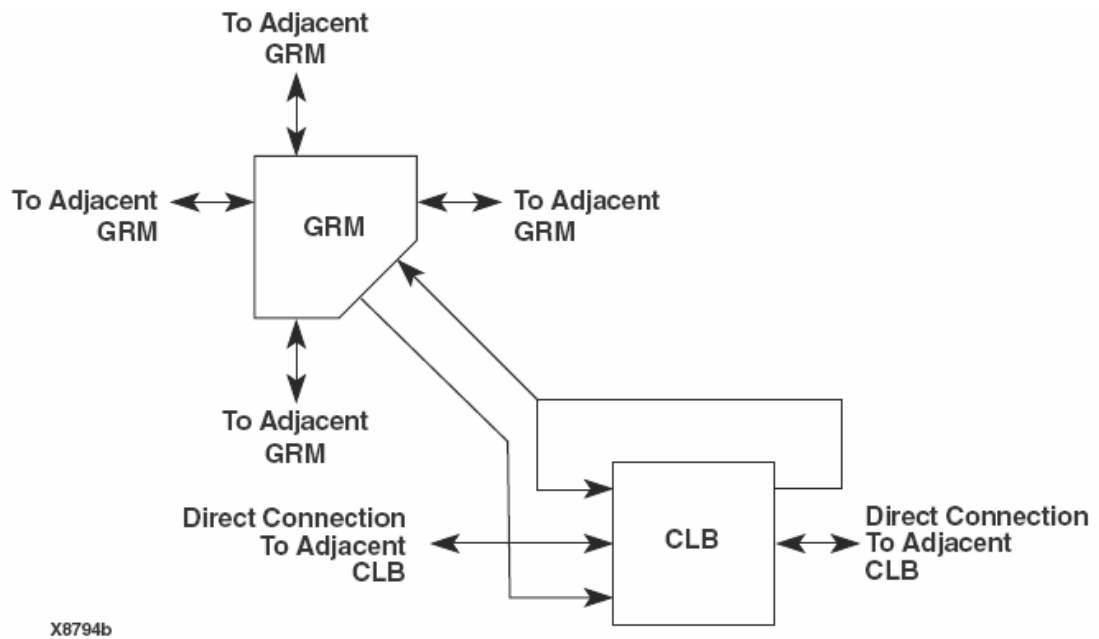


Figure C-3 Virtex Local Routing

Routing resources in Virtex FPGA can be classified to six types: local routing, general purpose routing, global routing, I/O routing, dedicated routing and clock distribution network.

Appendix D World setup in Webot

In order to modelling the environment and the robot (the world), following Webots nodes are employed:

- The walls are implemented as Extrusion node, which can specify geometric shapes based on a 2D cross-section extruded along a three dimensional spine;
- The robots are made up of following nodes:
 - A DifferentialWheels node to model the robot with a cylinder node as main body of the robot;
 - Two DistanceSensor nodes to implemented the two IR sensors with small cylinder nodes as the actual sensors;
 - Two cylinder nodes as the wheels;
 - A Pen Node to aid the drawing of trajectories of robots.

Appendix E Introduction to EO

EO is an ANSI-C++ compliant templates-based evolutionary computation library. It offers a large number of classes for a wide range of evolutionary computation categories. It is fully object oriented and component-based, so it is easy to subclass existing abstract or concrete class to implement custom evolution algorithm. [8]

The abstraction of components and control flow in EO framework is demonstrated in Figure E-1 Components and Flow Chart of EO [8]

. No matter how special an EA may be, it has the common features illustrated in this figure.

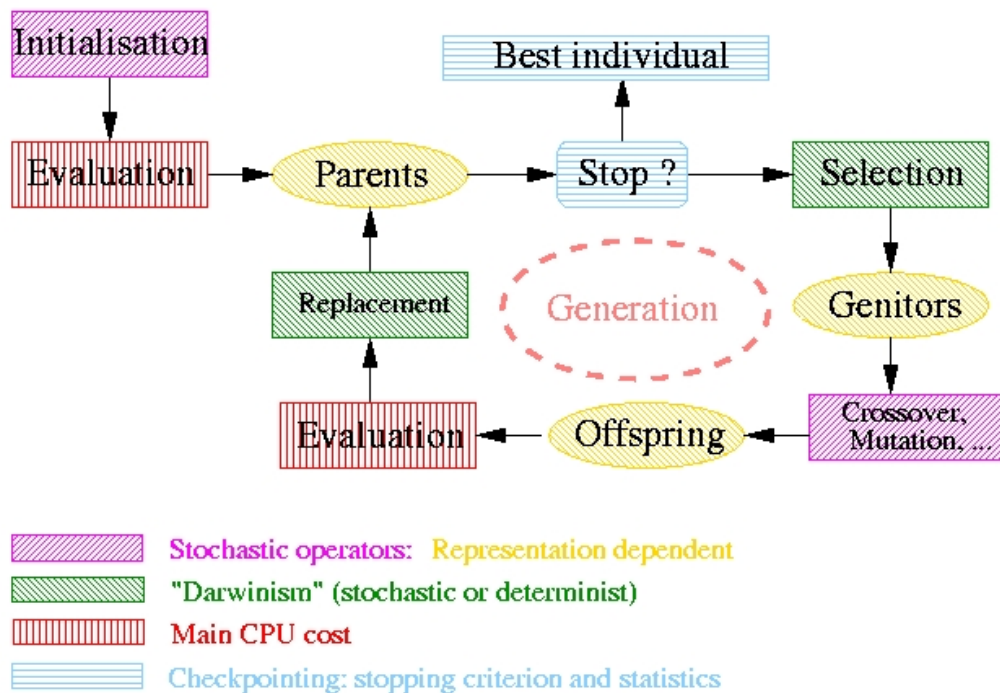


Figure E-1 Components and Flow Chart of EO³⁶ [8]

At the very beginning of each generation loop, the selection phase takes place first, in which some of the individuals from the whole population are selected as

³⁶ Genitors are those individuals selected in the selection phase which are permitted to "reproduce" in the next phase.

Appendix

the parents that are later altered by the variation operators to generate offspring. The selection phase mimics the first step of the Darwinism, where the best individuals are permitted to reproduce. [8]

The replacement phase is the second step of the artificial Darwinism: where the best individuals survive. This phase takes place after all the offspring are born through variation operators. The new individuals become the new population and substitute their preceding. [8]

Another feature introduced in EO framework to insert customized program is called *Checkpointing*. This mechanism is incorporated to allow some codes be executed at the end of each generation. Among the most important of all kinds of Checkpointing supported by EO, are stop criteria decision, calculating, outputting or saving some statistics of current population, updating some dynamical variables of a customized algorithm. [8]