2018

Toward Biologically-Inspired Self-Healing, Resilient Architectures for Digital Instrumentation and Control Systems and Embedded Devices

Shawkat Sabah Khairullah
Virginia Commonwealth University

TOWARD BIOLOGICALLY-INSPIRED SELF-HEALING, RESILIENT ARCHITECTURES FOR DIGITAL INSTRUMENTATION AND CONTROL SYSTEMS AND EMBEDDED DEVICES

A Dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy

In Computer Engineering at Virginia Commonwealth University

By

SHAWKAT SABAH KHAIRULLAH

Master of Science, University of Mosul, 2011

Adviser: Dr. Carl R. Elks,

Assistant Professor, Department of Electrical & Computer Engineering

Doctoral Committee:

Dr. Robert H. Klenke, Department of Electrical & Computer Engineering

Dr. Alen Docef, Department of Electrical & Computer Engineering

Dr. Tim Bakker, Commonwealth Center for Advanced Manufacturing

Dr. Fadi Obeidat, Synopsys, Computer Integrated Systems Design Company

Virginia Commonwealth University

Richmond, Virginia

November, 2018

Acknowledgement

First and foremost, I would like to express my sincere gratitude and special appreciation to my advisor Dr. Carl R. Elks for his help, motivation, and generous support of this research project. I appreciate the years under his guidance and I was inspired by his wisdom, and more important, by his attitude for research and life, which will benefit me for lifetime. I would have not completed this work without him and could not have imagined having a better advisor for my graduate studies. I would also like to thank the members of my advisory committee: Dr. Robert H. Klenke, Dr. Tim Baker, Dr. Alen Docef, and Dr. Fadi Obeidat for their insightful comments and feedback that greatly helped me improve the quality of my research work. I must express my profound gratitude to all members of my family, my wife and colleagues for providing me with unfailing support and continuous encouragement throughout the years of my Ph.D. work. Finally, I dedicate this dissertation to two people: my beloved mother, Wisal who passed away, but her soul is still with me and my father, Sabah who have prayed for me and made many sacrifices on my behalf. They are my role models of honesty, strength, and persistence and I owe a debt of gratitude to them.

List of Publications

- **Journal Papers:**

1- *1st Published Research Author*– S. S. Khairullah & C. R. Elks, A Bio-Inspired, Self-Healing, Resilient Architecture for Digital Instrumentation and Control Systems and Embedded Devices – June 2018.

   - Publisher: Journal of Nuclear Technology: vol. 202, no. 2-3, pp. 141-152, Jun. 2018.

2- *In preparation*– S. S. Khairullah & C. R. Elks, BioSymPLe: Biologically Inspired Self-Healing Hardware Architecture for Critical Applications in Automation, IEEE Transactions on Very Large Scale Integration (VLSI) Systems. ArXiv link.

3- *In preparation*– S. S. Khairullah & C. R. Elks, Self-Healing Bio-Inspired Hardware Systems: A Survey, Current Trends and Challenges, ACM Journal on Computing Surveys. ArXiv link.

- **Conference Papers:**

1- *1st Published Research Author*– S. S. Khairullah, Tim. Bakker & C. R. Elks, Toward Biologically Inspired Self-Healing Digital Embedded Devices: BIO-SymPLe – June 2017.

   - Publisher: Proceedings of the ANS NPIC & HMIT 2017 Conference: 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies.

TABLE OF CONTENTS

Contents	Page
TABLE OF CONTENTS	**vi**
LIST OF FIGURES	**x**
LIST OF TABLES	**xiv**
LIST OF ABBREVIATIONS	**xv**
Abstract	**xviii**
CHAPTER 1 INTRODUCTION	**1**
1.1 Motivation	1
1.2 Domain of Applicability: Manufacturing, Extreme Environments, Healthcare	3
1.3 Challenges	3
1.4 Resilience and Self-Healing Concepts	5
1.5 Objectives of the Research	7
1.6 Expected Contributions to the Field	8
1.7 Outline of The Dissertation	11
CHAPTER 2 BACKGROUND AND LITERATURE REVIEW	**12**
2.1 Essential Concepts of Dependable Computing	12
2.1.1 Attributes	12
2.1.2 Impairments to Dependability	13
2.1.3 Responding to Active Faults	16

2.1.4 Important Prinicples of Safety Critical Systems………......................……..…..……….. 17

2.2 Discussion on Bio-inspired Self-Healing Digital Systems ……....………………....….……….... 18

2.3 Fundamental Background on Field Programmable Gate Arrays (FPGAs)…….……………..….. 21

2.4 Self-Healing Process Characterization …………………..……....………………....….………... 23

2.5 Taxonomy of HW based Self-Healing Strategies ……....…………………….......….……….... 25

2.5.1 Cell Elimination Strategy…………………….………………......................……..…..…….... 26

2.5.2 Row Elimination Strategy…………………………..……………………….……………...…. 28

2.5.3 A Combination of Cell and Row Elimination Strategy……….……………………...…...….... 29

2.5.4 Cluster Elimination Strategy……………..…………..….……………….……………..….... 30

2.5.5 Re-routing Elimination Strategy………….………………….………………..……......…..… 31

2.5.6 Voting-by-Majority Elimination Strategy ………………...…...…………….…....……... 32

2.5.7 Unitronics Elimination Strategy ………………..…………..……………..…….…......... 33

2.6 Proposed Elimination Strategies…..…..……..…………..…..……………...…..……..…........ 34

CHAPTER 3 THE BIOLOGICALLY-INSPIRED SELF-HEALING ARCHITECTURE - BIOSYMPLE **37**

3.1 Design Objectives for BioSymPLe ……………………...…………….…………..…....…. 37

3.2 Biologically-inspired Model for BioSymPLe…….…………….....……………..…….….…...... 39

3.3 Architectual Design Principles for BioSymPLe ..…………………………….………..…....….. 42

3.4 Overview of BioSymPLe High Level Perspective...……………………….………..…....….. 43

3.4.1 Version1 Concept of BioSymPLe Architecture…………….…...….….…………………….. 46

3.4.2 Operational Detail of BioSymPLe ……………….…...….….………………………..…....….. 47

3.5 The Biological-inspired Cell ………………………………………………….……..…………. 49

3.6 Fault-Aware Generic Functional Block (FAGFB) Semantics ………………………………... 51

3.7 Fault-Tolerance Techniques for FAGFB …………………………………………………. 54

3.8 Reconfiguration by Graceful Degradation ………………………………………………….. 58

3.9 Scan Time……………….……………………………..…………………….……..……… 61

3.10 Execution of Functional Cells.….………………….……………….……..…………........ 61

3.11 Communication Models …………………….……………………………….…………… 63

3.11.1 Single Control Multiple Data (SCMD)…..……………………….……..……..... 63

3.11.2 Multiple Control Multiple Data (MCMD)………..…………………….……..... 64

3.11.3 Control Registers …..……….………………………..…………….……..….……. 64

3.11.4 Communication Bus ……….…..…….…………………………….……..……..... 65

CHAPTER 4 PRELIMINARY APPLICATIONS AND SIMULATION RESULTS 67

4.1 Case Studies and Applications applied to BioSymPLe…………..…….……..……..….……...... 67

4.2 Fault Injection Results for the First Case Study………………………..……........……..…... 67

4.3 Fault Injection Results for the Second Case Study………………......................………..…..... 70

4.4 Emergency Diesel Generator (EDG)…………………………....……………..…….…….….. 72

4.5 Cruise Control System (CCS)…………………..………………..……………....………….…..... 77

4.6 Example of Formal Verification of Properties in BioSymPLe………………..…………….. 81

4.7 Comparative DependabilityAssessment Results …………..…………...….…………………….. 85

4.7.1 Reliability and safety Modeling Analysis of BioSymPLe …………………………...... 90

4.7.1.1 Markov Process basics ……………………..….……………………………… 91

4.7.2 Markov Mathematical Analysis……………………………….……….……..……… 96

4.7.3 Reliability Modeling Analysis of One Lane of BioSymPLe….……………….……..……..… 99

4.7.4 Markov Mathematical Analysis ……………………....….………………..……… 102

4.7.5 Simulation Results …………………………………....….………………..……… 106

4.8 Version2 Concept of BioSymPLe Architecture ……………………….……..…....………… 111

CHAPTER 5 CONCLUSION AND FUTURE WORK **113**

5.1 Research Activities …………………………………………………………...…………...…. 113

5.2 Conclusion and Results Review……………...…………..…………………….…..….......…….. 114

5.3 Future Work ………………………………………..………………….…….……......….… 115

CHAPTER 6 BIBLIOGRAPHY **118**

LIST OF FIGURES

Figure Contents Page

Figure 1-1: Attributes of a resilient hardware architecture... 5

Figure 2-1: Cause-effects relationship among faults, errors, and failures using the three-universe model …………... 14

Figure 2-2: A 2D array of embryonic hardware architecture ……….………..……..……..… 20

Figure 2-3: Stratix® IV FPGA architecture ….………………………………………...…… 22

Figure 2-4: High-level block diagram of the Stratix IV ALM …………………….……….…… 23

Figure 2-5: The key attributes of self-healing in digital systems …………………………..…...… 23

Figure 2-6: Cellular bio-inspired reconfiguration methods …………………………….…..………… 26

Figure 2-7: Cell elimination as a self-repairing reconfiguration strategy …………………………… 27

Figure 2-8: Row elimination as a self-repairing reconfiguration strategy ……………………..…… 28

Figure 2-9: A combination of cell and row elimination strategies ……………………………..…..… 30

Figure 2-10: Cluster architecture strategy and process of self-repair ………………….………..… 31

Figure 2-11: Re-routing architecture strategy and process of self-repair ………………...………… 32

Figure 2-12: Unitronics architecture strategy ……………………………………………………….. 33

Figure 2-13: The first proposed elimination strategy and process of self-repair ….…………………… 35

Figure 2-14: The second proposed elimination strategy and process of self-repair ….……………..… 36

Figure 3-1: Biological cell life cycle ……………………………………………………….…… 39

Figure 3-2: Biological stem cell differentiation ………………………………………….….…… 41

Figure 3-3: The pathway from DNA to protein ……………………………………….……….. 42

Figure 3-4(a): High level perspective of BioSymPLe architecture ……….……………………….……... 44

Figure 3-4(b): The four organizational sublayers of the critical service layer……………….……... 45

Figure 3-5: BioSymPLe architecture version1 concept …………………………………..……… 46

Figure 3-6: The internal structure of the biological-inspired cell ……………………...….….…… 50

Figure 3-7: The internal structure of the configuration memory ………………………………... 50

Figure 3-8: The Fault-aware Generic functional block ………………………………………........ 52

Figure 3-9: Duplication with comparison functional unit …………….…………….….…………… 56

Figure 3-10: Register redundancy scheme ……………………….……………………..….……..… 57

Figure 3-11: Reconfiguration by graceful degradation ……………………………………………… 58

Figure 3-12: Voting on one lane of BioSymPLe ……...…………………………………………….. 59

Figure 3-13: The functional B cell Scan cycle …………………………..………………….…….…… 61

Figure 3-14: Event and data interface of four sequential B cells ………………………...….…….. 62

Figure 3-15: Event and data interface of four parallel B cells …………………………………….. 62

Figure 3-16: Single control multiple data (SCMD) communication model ……………….…...…. 63

Figure 3-17: Multiple control multiple data (MCMD) communication model……………………… 64

Figure 3.18 (a): Writeable control register ……………………...…………………………………… 65

Figure 3.18 (b): Readable status register ………………………………………………………….. 65

Figure 3-19: State machine based control unit ……………………………………………….…… 66

Figure 4-1 (a): Block diagram of the BioSymPLe architecture ……………………..……………… 68

Figure 4-1 (b): Time diagram simulation results ………………………………………….……….. 69

Figure 4-2 (a): Block diagram of one B cell and one T cell ……………………...………………….. 70

Figure 4-2 (b): Time diagram simulation results …………………………………………………..… 71

Figure 4-3: Logic diagram for starting the EDG ………………………………………..…….. 72

Figure 4-4: A simplified block diagram for EDG implemented on BioSymPLe …………………….….… 73

Figure 4-5 (a): Time diagram simulation analysis of a first sequential fault injection in the EDG application implemented on BioSymPLe architecture by connecting four critical service layers …….. 74

Figure 4-5 (b): Time diagram simulation analysis of a second sequential fault injection in the EDG application implemented on BioSymPLe architecture by connecting four critical service layers ……. 75

Figure 4-6 (a): Time diagram simulation of injecting three sequential transient faults in the input registers of Bio-Operational B Cell ……………………….…………………………………………..…. 76

Figure 4.6 (b): Time diagram simulation of injecting one permanent fault in the 61131-generic functional block ………………………………………………………………………….………………. 76

Figure 4-7(a): A closed loop cruise control system CCS …………………………………………… 78

Figure 4-7(b): A simplified block diagram for CCS implemented on BioSymPLe ……………...…. 79

Figure 4-8 (a): Generic proof structure of Simulink DV …………………………………………… 83

Figure 4-8 (b): Functional property proving ……………………...……………………………… 84

Figure 4-9: Safety1 property proving/ transient fault assertion ……………………………………. 85

Figure 4-10 (a): Self-healing capacity coverage between different architectures as system size increases ...………………………………………………………………………………………………… 87

Figure 4-10 (b): Area overhead between different architectures as system size increases ……...… 88

Figure 4-11: Transition state diagram for a reconfiguration by graceful degradation ………..……. 94

Figure 4-12: Transition state diagram for one lane of BioSymPLe ………………….………… 99

Figure 4-13: A smaller coverage causes higher unreliability ……………………………………….. 107

Figure 4-14: As time goes on reliability decreases with a constant failure rate λ …………………… 108

Figure 4-15: A higher coverage causes smaller unreliability ………………………………………… 109

Figure 4-16: A higher coverage causes smaller probability of failure for different failure rates …….. 110

Figure 4-17: BioSymPLe architecture version2 concept ……………………….……….…………… 111

LIST OF TABLES

Table Contents Page

Table 3-1: Functionality executed in the data flow division………………….………….………. 53

Table 3-2: State machine based truth table……………………………….…………….……..... 60

Table 4-1: CCS functionality is mapping on 17 functional cells …………………………………. 81

Table 4-2: Operational semantics of CCS application……………………………………………. 81

Table 4-3: Self-healing capacity coverage and area overhead comparisons in the EDG implementation with N * N array of cells ..…………………………………...…………………………………….. 86

Table 4-4: Comparison between BioSymPLe and SymPLe architectures………………………..…. 90

Table 4-5: State number and status for the markov modeled system based on Fig 4.11……….……. 96

Table 4-6: State number and status for the markov modeled system based on Fig 4.12…….……….. 101

Table 4-7: Failure rates for different componenets for the system modeled using the markov model of Fig 4.12………………………………………………...…………..………..……………… 102

Table 4-8: Failure and repair rates of the hardware components in BioSymPLe…….……………. 107

Table 4-9: Comparison results between the two versions of BioSymPLe……...…….……………. 112

LIST OF ABBREVIATIONS

I&C	: Instrumentation and Control
DNA	: Deoxyribonucleic acid
CCFs	: Common Cause Failures
CPSs	: Cyber Physical Systems
ASIC	: Application Specific Integrated Circuit
FPGA	: Field Programmable Gate Array
VHDL	: Very-High-Speed Integrated Circuit Hardware Description Language
RC	: Reconfigurable Computing
V&V	: Verification and Validation
D3	: Diversity and Defense in Depth
PLC	: Programmable Logic Control
EM	: Electromagnetic
HIRF	: High-Intensity Radiated Fields
FBD	: Functional Block Diagram
LD	: Ladder Diagram
SFC	: Sequential Functional Chart
ST	: Structured Text
IL	: Instruction List

HW	: Hardware
EHW	: Evolvable hardware
BIST	: Built-In Self-Test
FCRs	: Fault Containment Regions
LUT	: Look Up Table
ALM	: Adaptive Logic Module
IOB	: Input/Output banks
DSP	: Digital Signal Processing
IP	: Intellectual Property
LABs	: Logic Array Blocks
EMI	: Electromagnetic Interference
GCs	: a group of functional bio-inspired cells
CSL	: Critical Service Layer
RD	: Reconfiguration by Degradation
GFB	: Generic Functional Block
IEC	: International Electrotechnical Commission
FAGFB	: Fault-Aware Generic Functional Block
DWC	: Duplication with a Comparison
RRS	: Register Redundancy Scheme
CM	: Configuration Memory

RU	: Reservation Unit
BU	: Buffer Unit
BSMs	: BioSymPLe Machines
WCR	: Writeable Control Register
RSR	: Readable Status Register
SCMD	: single control stream multiple data stream
MCMD	: multiple control stream, multiple data stream
EDG	: Emergency Diesel Generator
EPRI	: Electric Power Research Institute
NPP	: Nuclear Power Plant
CCS	: Cruise Control System
PI	: Proportional Integral
DV	: Design Verifier
TMR	: Triple Modular Redundancy
MTBF	: Mean Time between Failures
MTTR	: Mean Time to Repair

Abstract

TOWARD BIOLOGICALLY-INSPIRED SELF-HEALING, RESILIENT ARCHITECTURES FOR DIGITAL INSTRUMENTATION AND CONTROL SYSTEMS AND EMBEDDED DEVICES

By Shawkat Sabah Khairullah, Ph.D.

A Dissertation submitted in partial fulfillment of the requirements for the degree of

Doctor of Philosophy at Virginia Commonwealth University.

Virginia Commonwealth University, 2018.

Adviser: Dr. Carl R. Elks,

Assistant Professor, Department of Electrical & Computer Engineering

Digital Instrumentation and Control (I&C) systems in safety-related applications of next generation industrial automation systems require high levels of resilience against different fault classes. One of the more essential concepts for achieving this goal is the notion of resilient and survivable digital I&C systems. In recent years, self-healing concepts based on biological physiology have received attention for the design of robust digital systems. However, many of these approaches have not been architected from the outset with safety in mind, nor have they been targeted for the automation community where a significant need exists. This dissertation presents a new self-healing digital I&C architecture called BioSymPLe, inspired from the way nature responds, defends and heals: the stem cells in the immune system of living organisms, the life cycle of the living cell, and the pathway from Deoxyribonucleic acid (DNA) to protein. The BioSymPLe architecture is integrating biological concepts, fault tolerance techniques, and operational schematics for the international standard IEC 61131-3 to facilitate adoption in the automation industry. BioSymPLe is organized into three hierarchical levels: the local function migration layer from the top side, the critical service layer in the middle, and the global function migration layer from the bottom side. The local layer is used to monitor the correct execution of functions at the cellular level and to activate healing mechanisms at the critical service level. The critical layer is allocating a group of functional B cells which represent the building block that executes the intended functionality of critical application based on the expression for DNA genetic codes stored inside each cell. The global layer uses a concept of embryonic stem cells by differentiating these type of cells to repair the faulty T cells and supervising all repair mechanisms. Finally, two industrial applications have been mapped on the proposed architecture, which are capable of tolerating a significant number of faults (transient, permanent, and hardware common cause failures CCFs) that can stem from environmental disturbances and we believe the nexus of its concepts can positively impact the next generation of critical systems in the automation industry.

CHAPTER 1

INTRODUCTION

1.1 Motivation

Automation technology is rapidly trending toward universes of networkable embedded computing objects working together to attain new opportunities and capabilities in ways we could not imagine a few decades ago. Networkable embedded computing is defined as clusters of loosely connected, smart, autonomous units coordinating their behaviors/actions to realize both local and global goals across specific domains. These domains could be medical, manufacturing, commerce, energy, transportation, etc… The terms Internet of Things, Internet of Ads (Appliance Devices) and Internet of Industrial Things are examples of networkable embedded computing objects for specific domains. Cyber Physical Systems (CPSs) are special, but important class of networkable embedded computing objects in which the confluence of physical interactions, embedded computing and wireless communication results in a transformative way in which society can interact with the real physical world. As result, applications with enormous societal impact and economic benefit are emerging at the convergence of Cyber Physical Systems (CPSs), ubiquitous wireless communication, and low-cost/small form factor computing devices. Examples at this technology nexus are; smart cities, smart energy distribution networks, distributed manufacturing and production, medical assistance devices, emergency and disaster preparedness and response – to name a few.

For CPSs and Internet of Things to fully realize their potential, the computing nodes, objects, sensors must become dependable and trustworthy – that is we will need to rely on devices that have properties of resilience, self-healing and self-protection [1]. Unexpected failures must diminish, and resilient system design and architectures must become an established option.

Among the many active areas of research in CPSs, research in reconfigurable computing (or processing), and resilient computing is very active and enduring. Reconfigurable computing (RC) devices or units are systems or architectures (Hardware HW or Software SW) that are able to adapt to the application or environmental changes on the fly. For example, Field Programmable Gate Array (FPGA) contains reconfigurable logic blocks, memory elements, multiplexers, and routing units that can be modified (partially reconfigured) to perform the functionality of one application at one time and reconfigured again to match the new application at different time [2], [3], [4], [5], [6]. In contrast to General Purpose Processing, which executes a set of fixed instructions on fixed hardware sequentially, Reconfigurable Computing devices can change their internal hardware partially or completely at run-time by downloading a configuration bitstream file from a configuration memory while the system continues to function [7], [3], [4], [8].

Reconfigurable Computing is well suited for the emerging CPS needs to develop more resilient and self-healing architectures. Therefore, the need for new classes of reconfigurable systems that have the capability of being not only adaptable at run time, but also fault tolerant and self-healing in disruptive environmental conditions is highly relevant.

Since von Neumann's seminal work on generating "reliable organisms from unreliable components" [9], [10], the significance of building dependable embedded systems has grown dramatically by utilizing the theories and practices of fault tolerant computing, reliability engineering, robust VLSI design, and consistent process fabrications. As Moore's law hits the technology and scaling wall, device reliability (and system reliability) has become more dependent on architectural features rather than circuit design and process fabrication processes. A relatively new emerging field for the realization of self-healing, reconfigurable digital systems is bio-inspired system design. It attempts to go beyond traditional approaches of fault tolerant computing and robust design to learn from processes and characteristics of living things, such as self-replication and self-healing properties, and adapting them to digital reconfigurable hardware platform [11], [12]. Ideally, we seek to develop new bio-inspired computing paradigms that strive to achieve self-healing and resilience properties against different types of faults during the execution of the application from a hardware perspective- that is, solutions targeted for programmable hardware architectures being used in safety-critical applications of next generation industrial automation.

1.2 Domain of Applicability: Manufacturing, Extreme Environments, Healthcare

We envision such self-healing reconfigurable machines being employed in a wide variety of applications, namely; cyber physical production systems, highly reliable advanced manufacturing, and safety-critical or life-critical applications. Much digital electronic instrumentation and control (I&C) systems embedded in the aerospace, medical healthcare, automotive manufacturing, and nuclear industry have obvious consequences of failure. A failure of these safety-critical systems could result in death, serious injury to people, or severe damage to the environment [13], [14].

1.3 Challenges

The current state of the practices for self-healing and self-protection systems is mostly achieved by the application of robust control system laws, Diversity and Defense in Depth (D3) protection techniques, and modular fault tolerance capabilities [13], [14], [15], [11]. These traditional techniques have worked well, but as the shift and demand for more dependable, distributed, and autonomous operations grows – these traditional approaches do not scale well with the high level of reliability required for safety-related systems, become costly to replicate the whole design with its hardware and software resources, and add complexity to the overall system design. Specifically, the main challenges can be identified as follows:

- How do we determine best self-healing strategies that can combine biologically-inspired self-healing models, architectural design principles and reconfigurable computing concepts to achieve high levels of fault tolerance and resilience without adding a significant burden of complexity, area overhead and verification?

- Can these bio-inspired self-healing designs be congruent to current programming practices used by industrial automation community, that is, can it be used by Programmable Logic Control (PLC) engineers?

- For highly safety-critical systems, we need designs that are “resilient”, “verifiable” and can be verified using model based formal verification and fault injection methods. How do we verify certain properties of the architecture will depend on tradeoffs between complexity of the self-healing mechanisms and the classes of faults intended to be tolerated?

- How do we organize the computing resources to achieve a high level of resilience and fault tolerance with minimal amount of hardware/software resources?

- What are the tradeoff spaces between realization, effectiveness, scalability, and verifiability?

- When is it necessary to monitor the behavior of the system, detect the fault occurrence that deviates the system from its desired behavior, and take a recovery action to restore the system into a safe state?

1.4 Reslience and Self-Healing Concepts

Definitions of resilience have evolved over the years since the inception of resilience by Holling [16], for ecological systems. Since then, concepts and definitions have emerged across many disciplines from systems engineering to economics to social systems [17]. Unfortunately, the number of definitions for "resilience" has increased significantly over the last decade, making it difficult to find a universal understanding of the term "resilience". However, a common denominator in most resilience formulations and definitions is the notion of *coping*. Coping is the ability to do something related to the system's capacity to manage an external (often unexpected) factor unto the system. The external factor is a generalization of a fault, disturbance, attack or stress that threatens the system functionality (see Figure 1-1).

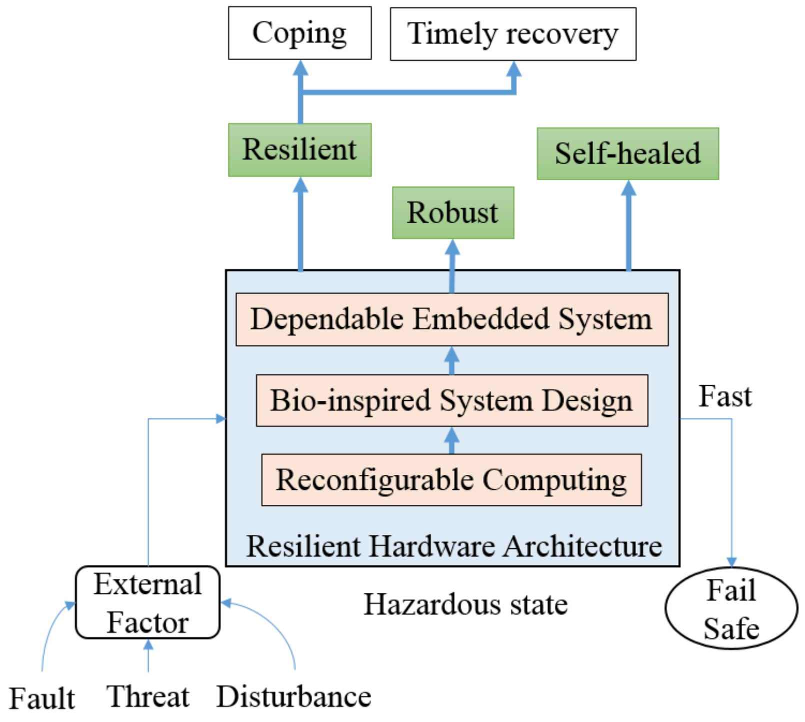


Figure 1-1: Attributes of a resilient hardware architecture.

A second important principle of resilience (noted in most definitions) is the notion of timely recovery by the system from the external threat or fault. For Cyber Physical Systems we refine our resilience notions to be more specific.

- A *resilient cyber physical system* is one that maintains state awareness and an accepted level of operational normalcy in response to disturbances, faults, including threats of an unexpected and malicious nature [18]. A cyber resilient system recovers fast enough to mitigate or absorb the effects of the disturbing event. In doing so, a resilient system may downgrade its functionality in a predictable manner. Resilient CPS should always produce a stable, resilient and safe control environment.

- *A Self-healing and self-protection* capabilities are essential requirements in high integrity applications of CPS. The attributes of self-healing and self-protection provide innate abilities to withstand disturbances, faults, failures and recovery quickly from these disturbances to a guaranteed level of service as the CPS interacts with the physical environment. A Self-healing System is defined as the system that is capable of monitoring its desired behavior, detecting the fault occurrence, diagnosing the fault type and the level of damage, and recovering from the hazardous states in which the system may cause catastrophic consequences in the environment or lead to the loss of human life and without human intervention [12], [11].

- A few remarks should be noted on resilience with respect to robustness. Robustness is the ability of a system to cope with a given set of disturbances and maintain its stability. Robustness and resilience belong to two different design philosophies. Robustness is concerned with strength, whereas resilience is concerned with flexibility. A robust system may be very hardened against one set of disturbances; however, it may be more fragile when faced with a different set of disturbances. Robust solutions are more defensive in nature, trying to detect then deflect attacks. In contrast, resilience solutions often involve continuous monitoring or self-awareness to detect changes and support for rapid reconfiguration and self-healing to allow continued operation with contained consequences.

1.5 Objectives of the Research

The primary objectives of this research are to rigorously investigate the design, assessment, and implementation of new biologically-inspired self-healing, resilient hardware architectures that are germane to safety critical CPS domain. Specifically I address the following objectives:

1. Investigate and develop new biological self-healing architectures that are novel and extend the current bio-inspired autonomic designs, and are relevant to safety-related digital control applications in industrial automation.

2. Investigate and develop methods to achieve high levels of resilience and fault detection coverage against different fault classes: transient, permanent, and hardware common cause failures while the resilience processes allow functions to execute for safety-critical applications.

3. Combine concepts from biology, traditional fault tolerance techniques, and IEC 61131-3 operational schematics to facilitate adoption in the automation industry.

4. Apply formal methods to critical components to enhance the trustworthiness and validation (T&V) of key concept properties.

Graphical programming has been widely adopted across many domains. The key point is that device use and programmability can vary from one domain to the next and is largely driven by the industry or engineering practices or standards. As example, all IEC 1131-3 compliant Programmable Logic Controller devices (PLCs) often use function block programing. PLCs and networked sensors are the prevalent technology in automation. Additionally, graphical programming as means to facilitate design and analysis in model based engineering tools is rather widespread – in such tools like Xilinx Vivado, Altera/Intel Quartus, National Instruments Labview, and MathWorks Simulink.

In this research, we have chosen to adopt graphical "function block" programming concepts in effort to make our design and innovations more accessible with the automation community. In this case, this research is inspired by the IEC 1131-3 semantic and operational requirement.

1.6 Expected Contributions to the Field

This research on bio-inspired self-healing digital systems is expected to be noteworthy to a number of stakeholders in the industrial automation and informatics area, extreme environment digital Instrumentation and Control (I&C) systems, and those areas concerned with safety-related Cyber Physical Systems (CPSs). As these different applications domain areas become pervasive, more and more CPS applications will be deemed critical for public services – such as, smart energy management, smart traffic automation, and smart cities. Digital embedded systems operating in these diverse application areas may experience harsh operating conditions and environmental changes where disturbances from random events such as High-Intensity Radiated Electromagnetic (EM) Fields (HIRF), extreme temperatures, radiation, or cosmic particle strikes are at an increased threat. In all these cases, the occurrence of multiple faults occurring, simultaneously affecting digital embedded devices or nodes, is a significant concern [19], [20]. As a consequence, the ability to detect, ride through, and repair the experienced failures (from a variety of different fault types) is important. In order to support all the objectives mentioned above, the work presented in this dissertation can be categorized in five major contributions and three minor contributions as described below:

Major contributions:-

1- A new Hierarchical Fault Tolerant (HFT) architecture with self-healing capabilities is designed in chapter 3 and analyzed in chapter 4.

2- This research extends the state-of-the-art in resilient Very-Large-Scale Integration (VLSI) design by developing new Fault Management Approaches (FMA) to support resiliency not previously reported in the literature. To the best of our knowledge, the majority of work on bio-inspired digital systems achieve the self-healing objective at decentralized level by embedding self-diagnostic modules inside the functional cells. These models are used to configure an internal control unit or notify a neighboring spare cell as a recovery mechanism. In our approach, we instead have local/internal and global/external function migration layers.

3- The HFT architecture is a notational (translatable) architecture that can be designed as an overlay on existing FPGA Fabric or as an ASIC.

4- Efficacy has been demonstrated on several digital I&C applications in chapter 4 and self-healing concepts of this research have been demonstrated to date with confirmatory evidence that the approach is feasible and is practical.

5- Different issues relevant to architectural concepts such as Reconfiguration by Graceful Degradation, Healing against hardware common mode failures that can defect a group of functional cells simultaneously, tolerating transient faults in input and output registers, detecting permanent faults in the functional execution units, and safety assurance is successfully addressed in the dissertation.

Minor contributions:-

1- To date, most of the traditional biologically-inspired self-healing approaches have not seriously addressed industrial automation or safety related Instrumentation and Control (I&C) applications. Specifically, we found the “programmability” aspects of the previous designs to be too low level for application engineers to understand (e.g., Very-High-Speed Integrated Circuit Hardware Description Language (VHDL) and bit level implementations), or they lacked the computing capacity required for control applications (i.e., simple logic). For example, As a result, a unique hierarchal self-healing architecture is designed in that resilience principles are derived from a heterogeneous perspective-combining concepts from biological systems (immune system, stem cells, living cell cycle, and genetic expression) and computer organization to provide a well-formed self-healing hardware architecture.

2- To the best of our knowledge, we believe this architecture is the first to employ PLC programming semantics accompanied by traditional fault tolerance techniques in a Bio-Inspired self-healing architecture. Programming semantics for PLC controllers typically specifies the standards for PLC software and these standards can define the PLC configuration, programming, and data storage. PLC vendors typically specifies five basic programming languages that support the IEC 61131 standard and these programming languages are Functional Block Diagram (FBD), Ladder Diagram (LD), Sequential Functional Chart (SFC), Structured Text (ST), and Instruction List (IL).

3- The proposed architecture can be *scalable* due to that the internal structure of bio-inspired cell contains a well-organized interface between the data flow and the control flow divisions. Whenever the size of the configuration memory is increased, more data outputs will be routed to the input lines of each functional block and more amount of functional cells can be connected to execute the application function.

4- Failure rates and repair rates were calculated for different components in the architecture and were used in solving Markov chain models using WinSTEM analysis program.

5- A comprehensive survey paper on Self-Healing Bio-Inspired Hardware Systems: A Survey, Current Trends and Challenges is submitted for publication.

1.7 Outline of The Dissertation

The reminder of the dissertation is closely organized as follows: In chapter two, background on essential concepts of dependable computing and self-healing process characterization is presented. Also, a brief related work on bio-inspired self-healing digital systems are reviewed in this chapter. Following the overview of the proposed self-healing architecture with its embedded layers in chapter three, two fault injection based case studies and two different industrial control applications used as a demonstration example to verify the correct operation of the self-healing properties are presented in chapter four, respectively. Additionally, in chapter four, a comparison between the proposed architecture and the existing self-repairing systems is being made in terms of self-healing capacity coverage and hardware area overhead. Finally, chapter five summarizes the proposed system and describes the feature work.

CHAPTER 2

BACKGROUND AND LITERATURE REVIEW

2.1Essential Concepts of Dependable Computing

This section provides informative essential background material on the concepts of dependability and fault tolerance as to provide clear definitions, terminology and concepts for the reader. In this thesis I adopt dependable systems concepts developed by Avizienis, et. al., [21] which are widely recognized as the gold standard for understanding the concepts of critical system attributes.

To begin with, a *dependable real-time system* has the ability to provide its intended, expected and agreed upon functions, behavior, and operations in a correct timely manner [21]. In dependability theory, a *system* is defined as an entity that interacts with other entities, i.e. other systems, including hardware, software, humans, and the physical world. These interactions with other systems are the environment or the context of a given system. The border between the system and its environment is the system boundary or common interface. The *service* delivered by a system is its timely behavior as it is perceived by its user(s).

2.1.1 Attributes

The *attributes* of dependable systems are the primary means by which the quantitative and qualitative requirements of a system are specified. As example, the ABS braking system on an automobile is specified to have less than 10^{-8}/unsafe failures per year. This reliability/safety requirement drives the architecture design and the design assurance practice for the ABS braking system. The following are some basic terms and concepts related to dependable system attributes as used in this thesis:

Definition : *Reliability*, a conditional probability that the system will perform correctly throughout the interval [*t0*, *t*], given the system was performing correctly at time *t0* [22], which is related to the continuity of service.

Definition : *Availability*, a probability that a system is operating correctly and is available to perform its functions at the instant time, *t* [22], which is related to readiness for usage.

Definition: *Safety***,** a probability that a system will either perform its functions correctly or will discontinue its functions in a manner that does not disrupt the operation of other systems or compromise the safety of any people associated with the system [22], which is related to the non-occurrence of catastrophic consequences on the environment.

2.1.2 Impairments to Dependability

Faults, errors, and failures affect the ability of a system to deliver its dependability attributes (e.g., safety, reliability, performance, etc.). Hence, they are called the *impairments* of dependability. Correct service is delivered when the service accurately reflects the system function. A *service failure*, abbreviated here to *failure*, is an event that occurs when the delivered service deviates from correct service. Since delivered service is a sequence of the system states, a service failure means that at least one (or more) states of the system deviates from the correct service state. This deviation is called an *error*. *Error propagation* occurs when an error is successively transformed into other errors through execution of functions on the digital system, (e.g., errors from component A propagate to component B when it receives information from component A). The cause of an error is called a *fault*. Faults can be internal or external to a system. A fault is active when it produces an error; otherwise, it is dormant. An active fault is either 1) an internal fault that was previously dormant and that has been activated by the computation process and/or environmental conditions, or 2) an external fault. *Fault activation* is the application of an input pattern to a component that causes a dormant fault to become active.

Implicit in the definitions of the above terms is a cause-effect relationship. The well-known 3-universe model depicted in Figure 2-1 shows the relationship between faults, errors, and failures [22]. Faults cause errors, and errors may propagate to the system interface to cause service failures.

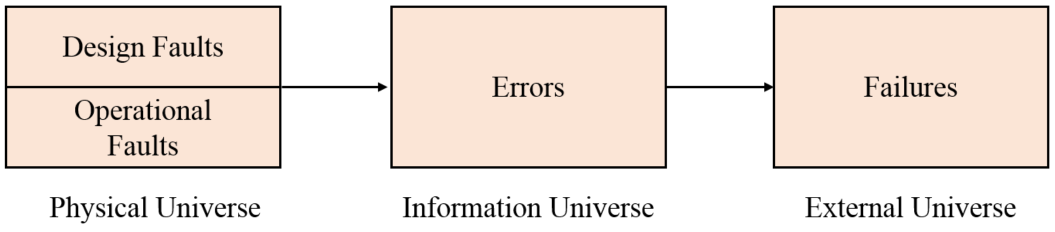


Figure 2-1: Cause-effects relationship among faults, errors, and failures using the three-universe model.

Definition: A *fault* is a physical defect, imperfection, or flaw that occurs within some hardware or software component [22]. A **fault** is the cause for an error.

Definition: An *error* is the manifestation of a fault. Specifically, an error is a deviation from accuracy or correctness [22]. Alternatively, an **error** is a design flaw or deviation from a desired or intended state.

Definition: A *failure* is the nonperformance or inability of the system or component to perform its intended function for a specified time under specified environmental conditions [22].

Essentially, the relationship in Figure 2-1 shows that failures are caused by errors, which are caused by faults. Associated with each term is a domain of effect, for example, faults are associated with the physical universe. Two categories of faults are possible in the physical universe: *operational faults and design faults*. Operational Faults include faults associated with semiconductor devices, mechanical elements, power supplies, and other physical entities that make up a system. These types of faults usually include fault types such as signal crosstalk in IC's, transistor failures in IC's, wear-out, oxidation, etc… Operational faults are often classified into three classes by their temporal behavior:

- **Transient faults**. A transient fault occurs once and subsequently disappears. These faults can appear due to electromagnetic (EM) interference, Ion particle disruption, power glitches, vibration, etc. In digital devices or elements transient faults typically lead to *bit-flips* in information storage and flow.
- **Intermittent faults**. An intermittent fault occurs time and time again - and disappears. These faults can happen when a component is on the verge of breaking down or, operating outside or near its maximum thermal specification.
- **Permanent faults**. A permanent fault that occurs stays until repaired. A permanent fault can be a damaged transistor in an IC, electrical bus driver circuit, damaged memory cell, etc. It should be noted that design flaws or faults are permanent faults as well, because they are present from the outset of operation. However, they are not usually classified in this manner.

On the other hand, *design faults* include hardware and software flaws are usually due to the inability to anticipate or fully consider certain interactions in hardware and software during system specification, design and implementation. Design flaws or faults are activated when specific input stimulus and computer states are present, e.g. a unique execution path is taken by the computer element. Design flaws are not randomly occurring actions per se; they are deterministic events since they occur every time the same input and state conditions happens.

2.1.3 Responding to Active Faults

In dealing with active faults, a designer has choices related to strategies at various levels or abstractions of the architecture to enhance system safety or reliability. These concepts explained below are useful to establish context for detection, self-healing, and reconfiguration methods discussed later in this chapter.

- ***Do nothing option* -** The error manifests into a failure at the module, and the larger system or subsystem of which it is a component inherits the responsibilities of handling the failure service.
- ***Fail-fast* -** The module reports a failure at its interface that something has gone wrong. This response notifies downstream or the next higher-level system devices that there is a failure, where it is at. The module may not be able to stop the failure from propagating.
- ***Fail-silent*** – When a module fails, it is designed to fail in such a way that its output goes into a quiescent state. The absence of active output is perceived to be a failure.
- ***Fail –symmetric*** – When a device fails its errors are seen consistently the same by connected down-stream devices.
- ***Fail-unsymmetrical*** – When a device fails its errors are seen inconsistently by connected down-stream devices.
- ***Fail-degraded* -** The module continues to operate correctly with respect to some predictably degraded subset of its service behavior from specifications, perhaps with some features missing or with a lower performance. The module typically notifies downstream devices of its degraded state, if it can.

- ***Fail Tolerant or Masking* -** Any value or values that are incorrect are made right and the module meets it specification as if the error had not occurred. The downstream devices of the module see no observable differences in delivered service behavior.

As a general rule, one can design systems to cope only with specific, anticipated faults. This is known as fault class identification for the system. Fault classes are usually derived from historical data, generally known threats, or threats related to a specific context or environment. Further, a system can be expected to cope only with faults that are actually detected. Faults (errors) can be classified into detectable and non-detectable. As example, a 7-3 hamming code will detect all single bit errors, but not all double bit errors.

Definition: A *detectable error* is one that can be detected reliably for a given fault class. If a detection procedure exists, and is in place and the error occurs, the system discovers the error with near certainty and it becomes a *detected* error. Conversely, an undetectable error is one that cannot be detected for the system error detection procedures.

Definition: *A Tolerable or maskable error* is one for which it is possible to devise a procedure to recover correctness. If a correction process exists, and is in place and the error occurs it is detected and is corrected, the error is said to be *tolerated*. Conversely, an *untolerated* error is one that is undetectable, undetected, and unmasked.

Often the term *coverage* is used to reflect the systems or modules ability to detect and/or tolerate faults/errors. Coverage is a metric for quantifying error detection capability and tolerance capability with respect to a given fault class. Fault coverage is sometime defined in terms of a conditional probability, especially when the detection mechanisms are imperfect or require testing to establish effectiveness. Coverage embodies the principle that the error detection action, fault tolerance is being performed correctly and on time.

2.1.4 Important Principles of Safety Critical Systems

For safety critical systems, failures associated with digital devices or systems that lead to hazardous conditions must be extremely improbable[1]. Computer systems without any additional protection against failures are most likely to fail in ways that are unacceptable to the overall safety of the system. Thus, a

[1] The term "extremely improbable" is dependent on the application requirements which vary from application to application. For example, the requirement for digital flight control systems in civilian air transports is 10^{-9} failures per 10 hour flight –which is extremely improbable.

defensive strategy must be employed in computer systems to do one or two actions upon the occurrence of erroneous behavior associated with the computer systems:

1. **A fail-safe action**: Upon detection of an error that could lead to failure, *force* (by some defensive measure) the computer device to discontinue its current operation in manner that does not compromise the safety requirements of the overall system.
2. **A fail-operational action:** Upon detection of error that could lead to failure, *overcome* (by some corrective action) the erroneous condition such that the operation of the computer device is able to continue according to its specified safety and functional requirements.

Fail-safe and fail-operational actions are primary tenets for safety critical systems. These actions are not native in most computer devices; the computer must be designed to behave as a fail-safe or fail-operational device. Traditionally, fault detection and/or fault tolerance mechanisms are added to the fault intolerant design to detect and possibly correct the effects of faults during the operation of the system. It's important to note, that some application may not have a "natural" fail safe state – examples are rotorcraft, magnetically levitated trains. In these cases, fail-operational design is the only choice for achieving safety enhancement.

2.2 Discussion on Bio-inspired Self-Healing Digital Systems

In this section, to provide context for the reader on HW based methods that are closely related to our research work and can be realized in FPGA architectures, a brief discussion on the three main methodologies that have been often used in achieving high levels of resilience and self-healing properties by utilizing the power of reconfigurable hardware computing is presented. At the broadest stance, hardware based techniques to achieve self-healing in digital systems can be classified into three distinct areas.

1. *Evolvable hardware (EHW)*: - is a research topical area that brings together reconfigurable hardware, artificial intelligence, fault tolerance, and autonomous systems. It uses a simulated evolution performed by stochastic search algorithms such as genetic algorithms and evolutionary programming strategies to search for a new hardware configuration. The purpose of this evolution is to reach the best performance by finding the best hardware architecture for the given application through implementing the search algorithm on a reconfigurable device like the FPGA. The resulted EHW allows to self-adapt to compensate for failures or unanticipated changes in the environment [4], [23].

2. *Embryonic hardware*: - includes a two dimensional (2D) array of embryonic cells implemented in a reconfigurable hardware platform. Each one of the embryonic cells stores a group of configuration genetic codes in a memory that describe the functionality of the cells and mimic the biological genome stored inside the living biological cell. Development of the system is achieved through differentiating one genetic string for each embryonic cell in the system. This differentiation process is based on the location of the cell within the embryonic array as it is shown in Figure 2-2 [23], [24], [25].

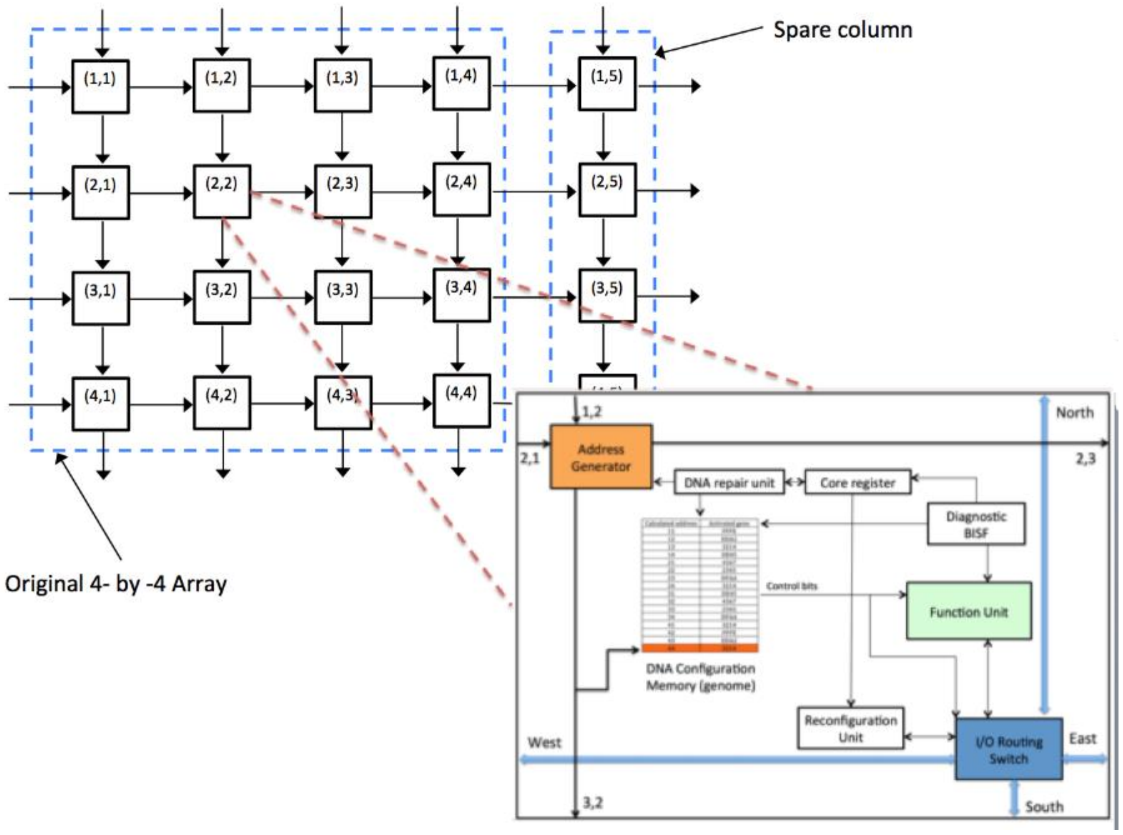


Figure 2-2: A 2D array of embryonic hardware architecture.

3. *Traditional self-healing hardware*: - it typically contains three types of cells: functional cell, spare cell, and routing cells organized as a two dimensional array of bio-inspired cells. Once the functional cell goes faulty, the resulted error is detected by the embedded Built-In Self-Test (BIST) inside the same cell, the output of function cell is isolated from the design, and the faulty cell is replaced by one of its neighboring spare cells [26].

There are some designs found in the literature that try to combine some attributes from these three different domains. However, in this research work, we are concentrating on concepts from two domains: Embryonic hardware and traditional self-healing hardware methods that are closest to our work.

2.3 Fundamental Background on Field Programmable Gate Arrays (FPGAs)

Field programmable gates arrays (FPGAs) are a specific family of digital integrated circuits that are used for implementing custom digital embedded devices. These programmable devices have the capability of being configured as many times as the user wishes in order to achieve a desired result and perform a specified application [5], [2], [27], [28]. Furthermore, in SRAM-based FPGAs the configuration data is stored in a volatile memory and downloaded into the FPGA fabric to configure the embedded functional and routing resources. Reconfiguring the FPGA fabric inside the chip means that the FPGA changes its functionality on the fly to support a new application.

A Stratix IV FPGA architecture from Altera which provides advanced features with efficient logic usage is illustrated in Figure 2-3 [29]. The main building blocks of this FPGA chip are *Adaptive Logic Module (ALM)* which contains a variety of LUT-based logical circuits, *Input/Output banks (IOB)* which interface the internal logic of the FPGA chip to a pin in the FPGA package, and communication resources that allows the arbitrary connection of ALMs and IOBs. Altera FPGAs usually have additional resources such as Digital Signal Processing (DSP) blocks, clock managers, high-speed transceivers, and sometimes hard Intellectual Property (IP) cores [29].

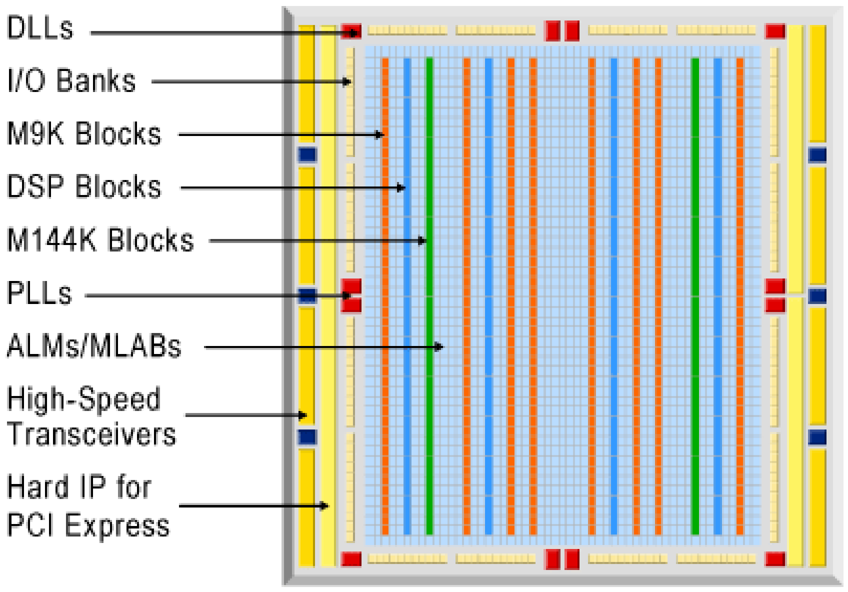


Figure 2-3: Stratix® IV FPGA architecture

The Stratix IV FPGA core fabric contains an array of the Logic Array Blocks (LABs) that are made up of adaptive logic modules (ALMs) (see Figure 2-4) that can be configured to implement logic functions, arithmetic functions, and register functions. LABs and ALMs are the basic building blocks of the Stratix IV FPGA device [29]. The ALM module has 8 inputs with a fracturable look-up table (LUT) that can be divided into two adaptive LUTs (ALUTs) using Altera's patented LUT technology. With up to eight inputs for the two combinational ALUTs, one ALM can implement various combinations of two functions. This adaptability allows an ALM to be completely backward-compatible with four-input LUT architectures. Each ALM is capable of:

- A full 6-input LUT or select 7-input LUT.
- Two independent outputs of multiple combinations of smaller LUT sizes for efficient logic packing.
- Implementing complex logic-arithmetic functions without additional resources.

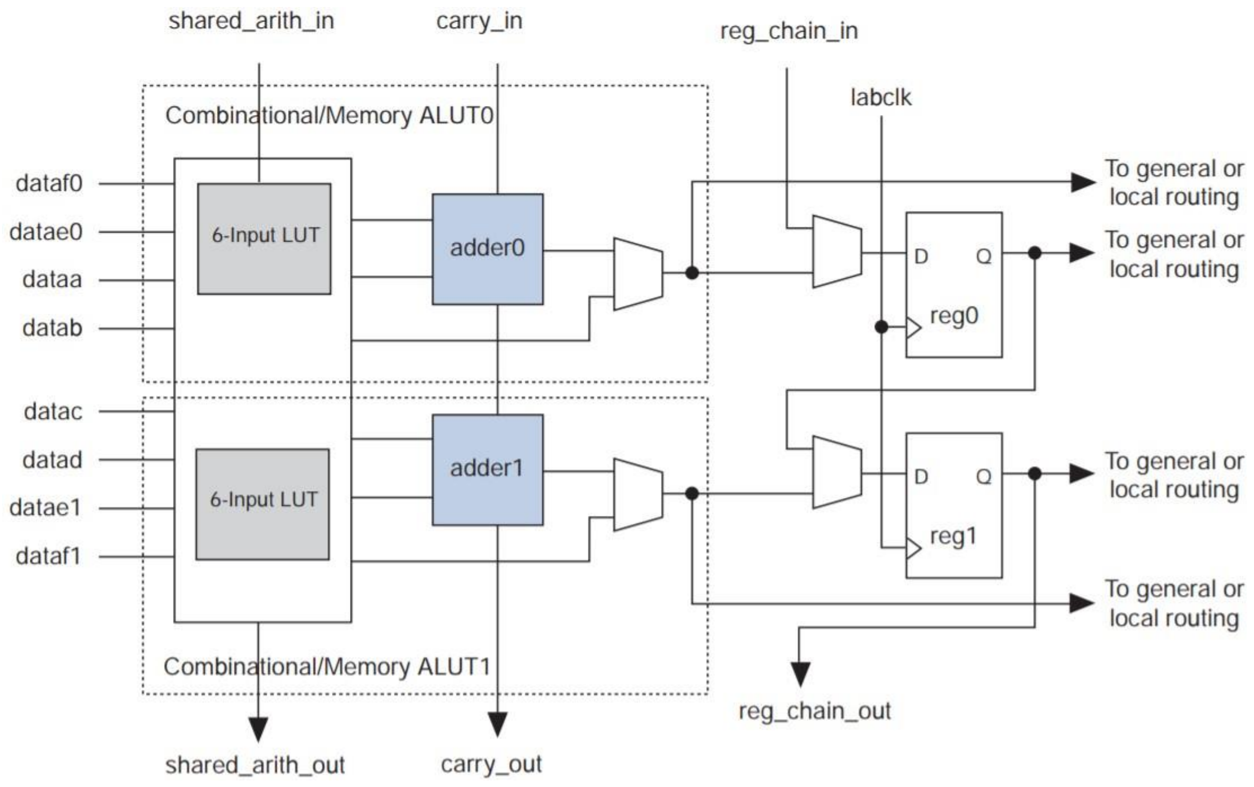


Figure 2-4: High-level block diagram of the Stratix IV ALM

2.4 Self-Healing Process Characterization

Over the last two decades, new research on self-healing digital systems inspired by biology has been growing steadily – to enhance fault tolerance, resilience, and survivability properties that go beyond traditional space and time redundancy methods [30], [24], [31], [32], [33], [34], [35]. Figure 2-5 presents the five key attributes of self-healing digital systems.

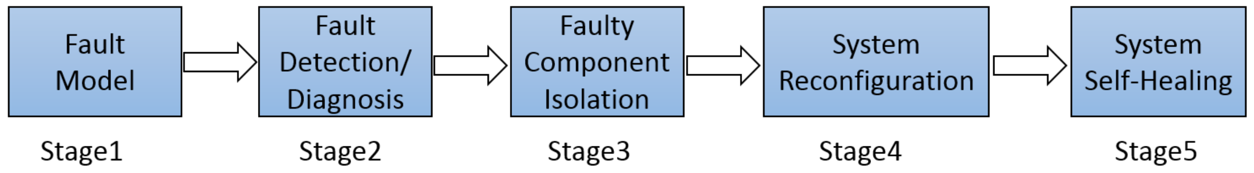


Figure 2-5: The key attributes of self-healing in digital systems.

These attributes are partitioned into five stages that are defined as follows:

- The fault model stage defines the classes of perceived faults a system is expected to encounter in its operational lifecycle. This fault model can include various classes of faults such as *logic faults* (stuck-at faults, delay faults) or faults defined based on its temporal behavior such as, transient, intermittent, permanent. The fault model may also include systemic faults from design errors. Defining the fault model provides a basis for the justification of the fault detection mechanisms.

- The fault detection/diagnosis stage is concerned about detecting the manifestation of different faults, given the occurrence of those faults. Design issues include reliable detection, imperfect detection, the bounds on the number of faults detectable, and resources needed to detect faults. Additionally, identification of the faulty module responsible for the generated error is essential in these kinds of systems.

- The faulty component isolation stage is defined as prevention of fault propagation across defined boundaries in the system. For example, if we have two digital devices: a transmitter and a receiver that are interfaced by a communication protocol. If one of these two devices defected by a fault, Isolation of a failed digital device from its neighboring healthy device will prevent the spread of the effects of the same fault into the receiver device. Furthermore, in fault isolation we are trying to limit the physical pathways of a fault from propagating via Fault Containment Regions (FCRs).

- The system reconfiguration stage is the process of removing a misbehaving component and replacing it with a healthy component at the component level. Reconfiguration depends on the ability of the system to effectively identify, isolate and replace faulty modules within some bounded time. There are many approaches to reconfiguration [36], for an in-depth survey.

- The system self-healing stage is the process of modifying the behavior of the digital embedded system automatically to compensate for failures or environmental changes. These modifications take the state of the defective system into a healthy state that is acceptable for continued operation and can be considered effective if they are taking place within specified timeframes [12], [15].

2.5 Taxonomy of HW based Self-Healing Strategies

Most of the previous work in hardware based bio-inspired systems has focused on the later stages of Figure 2-5, in particular stage 4 of these five stages, which includes the reconfiguration strategy of the system and how its structure is reconfigured in order to adapt to different failure modes. However, our aim in this research is to be comprehensive in our treatment of design - we address all of these stages (1, 2, 3, 4, and 5) to an effort to architect a highly resilient and survivable system.

Different strategies that have been found in the literature to date for realizing self-healing digital systems are presented in Figure 2-6. Most of these strategies aim to achieve high levels of fault tolerance and resilience against different failure modes through being inspired by the embryonic development of living organisms and its self-healing properties.

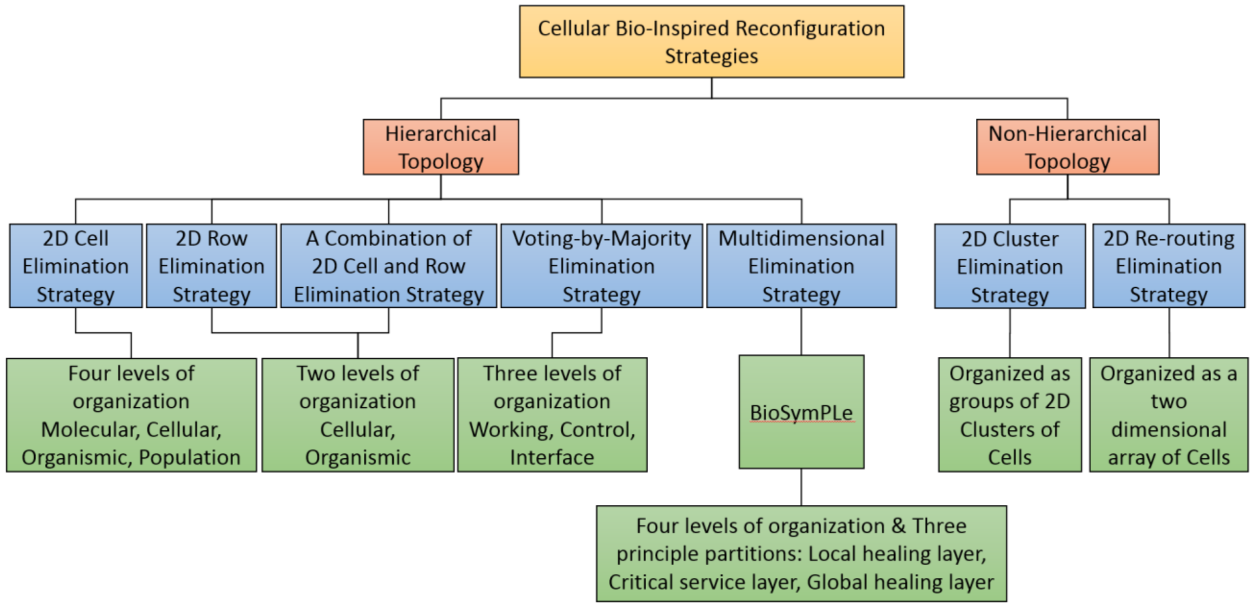

Cellular Bio-Inspired Reconfiguration Strategies
Hierarchical Topology
Non-Hierarchical Topology
2D Cell Elimination Strategy
2D Row Elimination Strategy
A Combination of 2D Cell and Row Elimination Strategy
Voting-by-Majority Elimination Strategy
Multidimensional Elimination Strategy
2D Cluster Elimination Strategy
2D Re-routing Elimination Strategy
Four levels of organization Molecular, Cellular, Organismic, Population
Two levels of organization Cellular, Organismic
Three levels of organization Working, Control, Interface
BioSymPLe
Organized as groups of 2D Clusters of Cells
Organized as a two dimensional array of Cells
Four levels of organization & Three principle partitions: Local healing layer, Critical service layer, Global healing layer

Figure 2-6: Cellular bio-inspired reconfiguration methods.

Reconfiguration strategies for Bio-Inspired Cellular architectures are basically organized in two broad domains: hierarchical approaches and non-hierarchical approach. Six basic strategies are presented in Figure 2-6. These strategies are cell elimination, row elimination, a combination of cell elimination and row elimination, majority-by-voting elimination, cluster elimination, and re-routing elimination. All these strategies will be presented briefly in the following sections.

2.5.1 Cell Elimination Strategy

In [31], [37], [38], [39], a new programmable cellular architecture that performs logic and arithmetic operations for a self-repairing FPGA has been designed by Mange et al. This architecture includes four hierarchal levels of organization: molecular, cellular, organismic, and population. At the molecular and cellular levels, two fault tolerant techniques based on time redundancy were used detect the fault occurrence inside the cell within the array at the decentralized level. This proposed approach for a self-repairing reconfiguration strategy is called cell elimination as shown in Figure 2-7. The aim of using this strategy is to detect the fault occurrence in any cell embedded in a two dimensional array of functional cells and recognize that fault if it is transient or permanent. If the fault is determined to be permanent the faulty cell will be replaced with the nearest spare cell in such a way that the functionality of the faulty cell and the functionality of all the cells on the right side of the faulty cell are shifted to the right to use the spare cells available as backup resources.

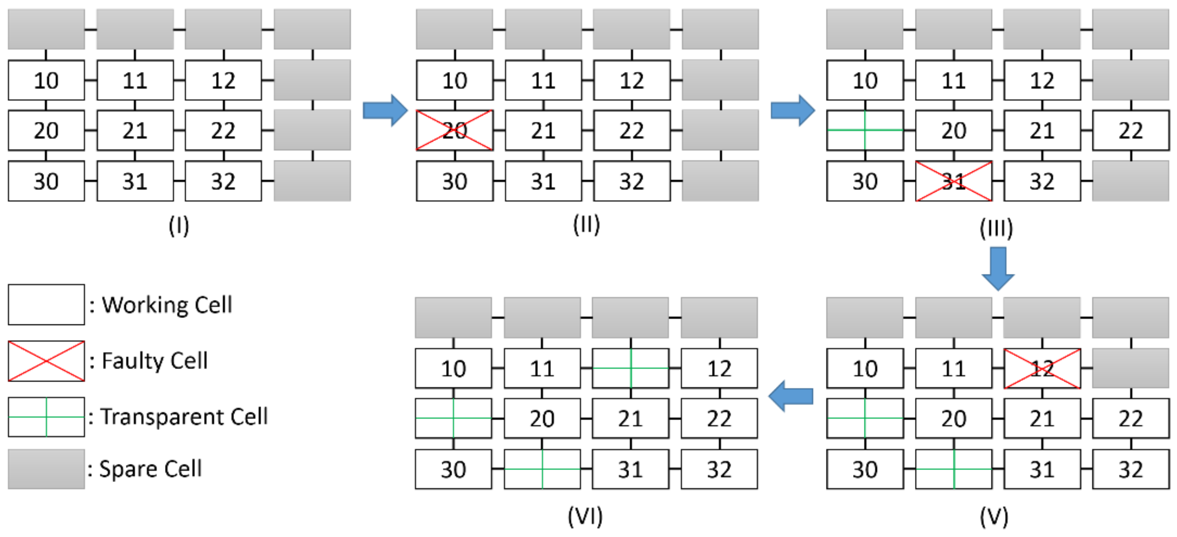


Figure 2-7: Cell elimination as a self-repairing reconfiguration strategy.

The advantage of using this strategy as a self-healing mechanism is to make the design of the embryonic array more reliable and capable of tolerating more failed-cells compared to the row elimination strategy that will be explained in section 2.3.2. This advantage in terms of efficiency could be achieved due to that the cell elimination mechanism would eventually use more available spare functional cells in the embedded digital system to mitigate the effect of failures. However, the disadvantage of using this reconfiguration strategy is that more hardware area overhead is required as a comparison with the cell elimination strategy. The reason is that more complex routing switches and larger configuration memories embedded inside the functional cell will be needed. Furthermore, each spare functional cell in a row is required to store the configuration genetic codes for all the functional cells in the two dimensional electronic array.

2.5.2 Row Elimination Strategy

In [30], [40], Tyrrell et al. have presented a different architecture which embeds a logic block performing the functions by a 2-1 multiplexer and a D flip-flop. This approach is a two-level hierarchical architecture consisting of a cellular level and organism level. Two modes, operation and configuration were proposed to control the operation of the organism in a fault tolerant manner. This second approach uses a concept of row elimination strategy as shown in Figure 2-8. Once a cell in a row gets defective, that entire row is removed by being transparent and replaced with its neighboring spare row from the top. The advantage of this strategy is decreasing the level of complexity in the routing switches because each faulty cell will be required to transfer the functionality to its neighboring cell from the top. However, the disadvantage of using this reconfiguration strategy is that the number of spare cells in each row must be always equal to or more than the number of faulty cells in the same row at one instant of time. As a consequence, this mechanism lead to a poor hardware area overhead because it makes the embryonic electronic array wasteful of available resources.

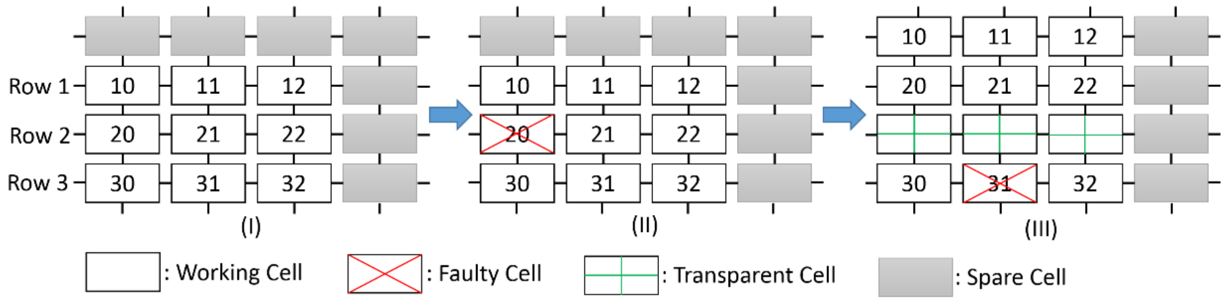


Figure 2-8: Row elimination as a self-repairing reconfiguration strategy.

The bio-inspired cell architecture that has been designed at the University of York, U.K, is developed by Zhang et al, at the University of West of England in [24]. Their architecture also works at two levels: cellular and organism, but they have increased the level of fault coverage by adding more levels of reliability to detect the transient faults in the configuration memory besides the detection of permanent faults. Whenever an error occurs in the memory, a Deoxyribonucleic acid (DNA) repair unit sets an error flag in a core register to reset the contents of that gene. However, a backup gene is used at a second fault tolerant level to replace the faulty gene if the error was diagnosed as a permanent error (mutation). In [25], Wang et al. have designed a different approach to realize the self-healing cellular architecture. Their design was based on using a Look Up Table (LUT) as a building block for the functional unit. Column elimination is used as a self-healing mechanism by making all the cells within the same column of the faulty cell transparent cells that perform no function.

2.5.3 A Combination of Cell and Row Elimination Strategy

The third approach used as a self-repairing reconfiguration strategy in an embryonic electronic array is called a combination of cell elimination and row elimination as shown in Figure 2-9. In this approach, the spare functional cells are distributed along the right side and top side of the two dimensional array. The reconfiguration process is performed in two phases. The first phase, once the faulty functional cell is detected through a Built-In Self-Test (BIST) digital circuit embedded in the same functional cell, this cell is killed and becomes a transparent cell, and its functionality is replaced by a spare cell available on the same row. However, in the second phase, the entire row is eliminated and all the cells of that row are shifted to the upward when the number of spare cells becomes less than the number of faulty cells [41], [38]. The disadvantage of using this reconfiguration strategy is that the number of spare cells in each row must always be equal to or more than the number of faulty cells in the same row at one instant of time. In addition, this method suffers from the same two problems presented previously in section 2.2 and section 2.3, which are shifting the local addresses of the faulty cells and deactivation process of a complete row of cells. Consequently, these problems can cause an increased level of complexity that increase the recovery time required to achieve the self-healing objectives.

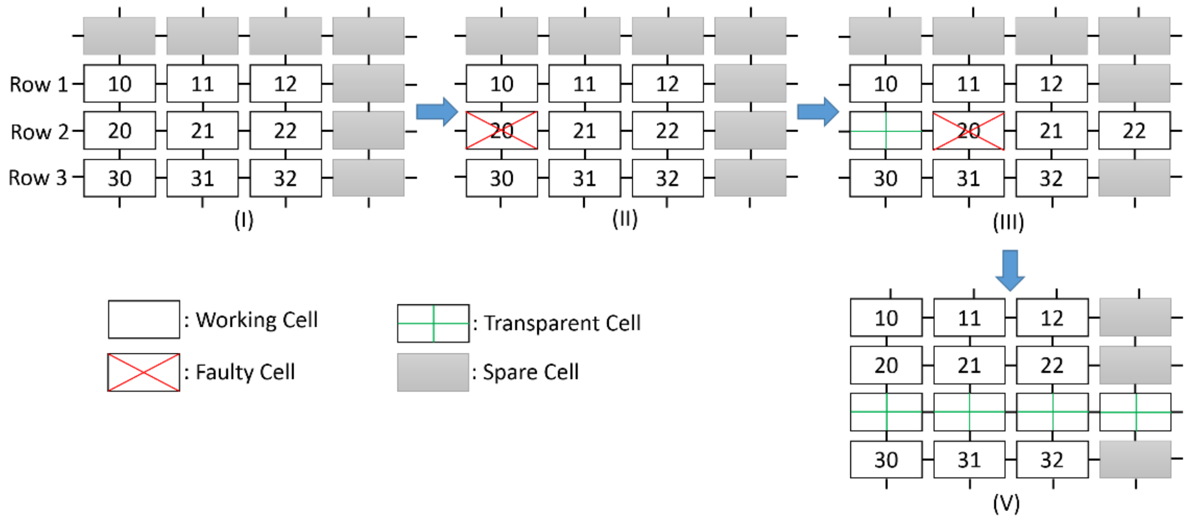


Figure 2-9: A combination of cell and row elimination strategies.

2.5.4 Cluster Elimination Strategy

In [41], [42], a group of researchers have developed a new bio-inspired cluster-computing elimination strategy comprised of nine FPGA boards. Each FPGA board includes a two-dimensional network of cells that are connected to perform the global function of application. Furthermore, each cluster embeds nine cells, five of them are active cells with a white background and four of them are backup cells with a gray background (see Figure 2-10). The function of each active cell is specified through the gene expression for the genome stored inside the cell. For example, each cell has five genes (A, B, C, D, and E) and one gene (with underline) is only activated within each cell based on its location in the array. Once one of the active cells goes faulty, a neighboring backup cell takes over the functionality of that faulty cell [42]. The disadvantage of this approach is that each spare cell stores the genetic codes for all functional cells in the cluster, which lead to a high hardware area overhead. Additionally, this strategy is not able to provide self-healing properties against the occurrence of five concurrent common cause failures.

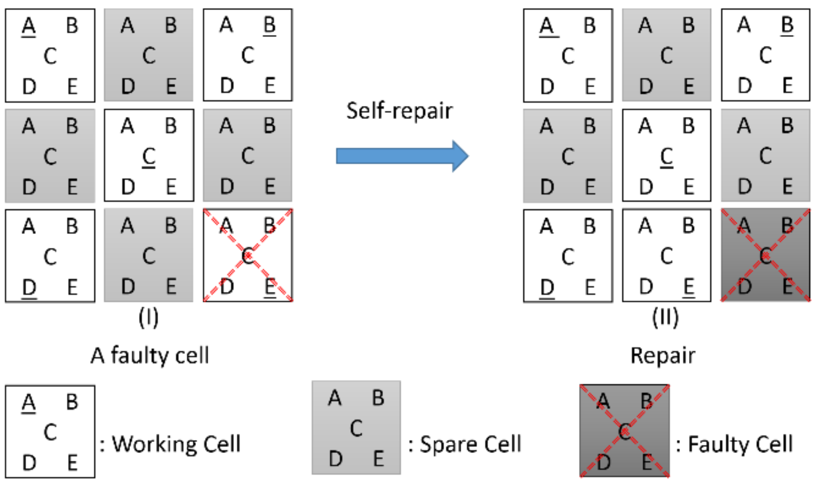


Figure 2-10: Cluster architecture strategy and process of self-repair.

2.5.5 Re-routing Elimination Strategy

In [26], the authors have designed a new architecture inspired by the biological process of the human immune system (see Figure 2-11). Their approach, which is called a re-routing architecture, contains three types of cells: functional cell, spare cell, and routing cells organized as a two dimensional array of bio-inspired cells. Two spare cells and two routing cells surround each functional cell except those at the periphery of the array. In other words, four functional cells surround each spare cell. Once the functional cell goes faulty, the resulted error is detected by the embedded BIST inside the same cell; the output of function cell is isolated from the design and placed in high impedance using a tristate buffer, and the faulty cell is replaced by one of its two neighboring spare cells. Consequently, each functional cell needs to store its own configuration code and each spare cell needs to store the configuration of its four neighboring functional cells to take over the functionality of any cell in case of the failure of one of the four faulty cells. Routing cells used to route the output of internal functional cell into output cells [26]. The advantages of their work includes the capability of tolerating the transient faults inside the configuration memory and the direct connection between the functional cells, which eliminates the need for complex routing units embedded inside the cell. However, that has increased the level of complexity in mapping the logic expressions into the functional cells in the array.

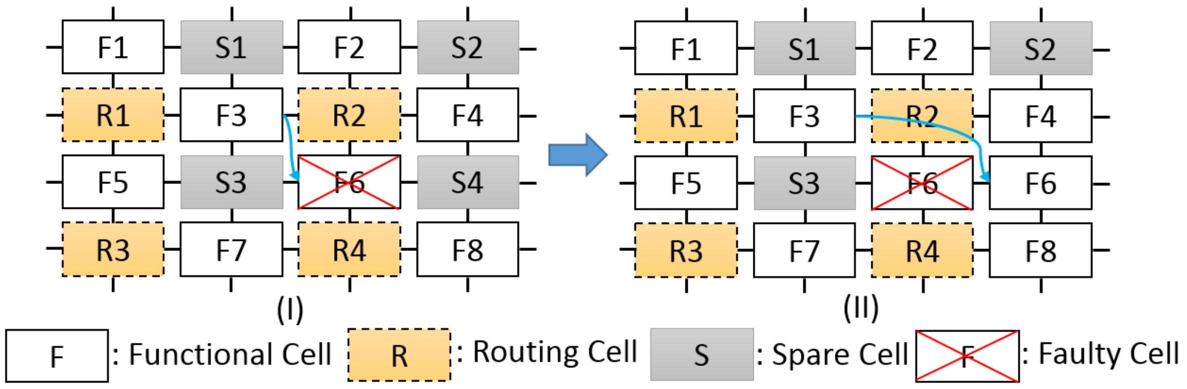


Figure 2-11: Re-routing architecture strategy and process of self-repair.

2.5.6 Voting-by-Majority Elimination Strategy

In [32], a novel self-repairing hardware architecture inspired by paralogous gene regulatory circuits in molecular biology was designed to achieve fast fault recovery with an efficient use of hardware resources. When the living organism has two genes and one of these genes die, that dead gene will not eventually lead to the death of the whole organism. In this biological feature, similar genes are assumed to have similar functions with their paralogous partners in the architecture. However, the living organism contains these regulatory circuits will not survive if both genetic partners die. As a consequence, three layers of organizations: a working layer, a control layer, and an interface layer, have been used to realize this multi-layered architecture. Three types of cells: working, redundant, and spare are being embedded in the working layer. The control layer include a two-dimensional (2D) array of control modules that embed four processing units used to detect the failure inside the working layer. A group of these processing units are using a concept of voting-by-majority to detect the failures in one working module. Ultimately, the interface layer is responsible for receiving information about the faulty cells and downloading the configuration file required from a hosting computer to differentiate the spare cells. The fast recovery was achieved by using a group of programmed hot spare cells that quickly replace the faulty cells through the partial reconfiguration technology available in FPGA. On the other hand, the main limitations of this approach are that it is highly vulnerable to the effects of common mode failures that can affect the two replicated units of the duplication with comparison unit. Also, different components like system bus, self-test module, and the configuration memory are not protected against any class of faults.

2.5.7 Unitronics Elimination Strategy

Figure 2-12 shows the architecture of a novel fault-tolerant architecture that is inspired by the biological process of unitronic creatures such as prokaryotic bacteria [43]. This architecture consists of two types of cells: core cells located in the middle of a two-dimensional array and connected to perform specific tasks, and peripheral cells distributed around the core cells to implement the replacement process for faulty cells. Furthermore, the peripheral cells manage the flow of information between the internal structure and the output interface. The self-healing mechanism is achieved by transferring the genetic code horizontally between the cells through a specialized local bus [44].

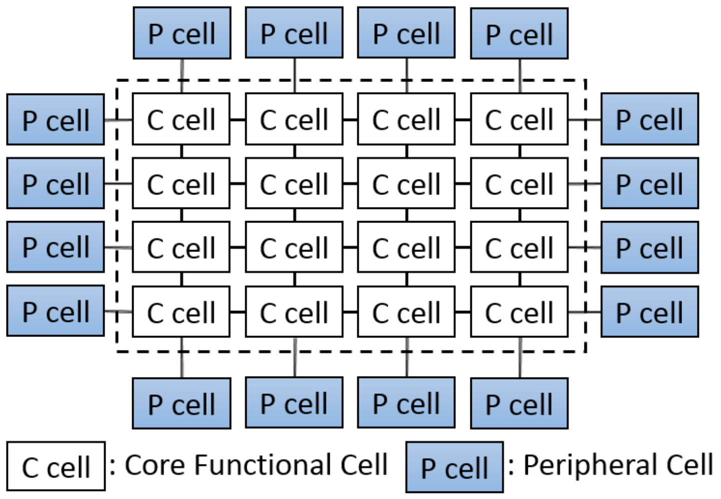


Figure 2-12: Unitronics architecture strategy.

2.6 Proposed Elimination Strategies

This section briefly presents the proposed self-healing elimination strategies that aim to build a biologically-inspired reconfigurable digital hardware device that is inherently resilient. This hardware device tries to fill the gaps that have not been addressed completely in the field of bio-inspired self-healing multicellular design. Some of the gaps include the incorporation of more realistic fault occurrence models embedded in the digital system.

Hierarchical coordinated fault management to provide resilience to multiple classes of faults was designed. In addition, using concepts of formal design assurance to verify safety and functional properties for the critical aspects of the design can qualify the novel self-healing architecture to be used in different safety-critical applications. These applications typically require high levels of verification and validation (V&V) by meeting the design requirements and specifications with its hardware implementation.

The proposed architectures will be described in detail in chapter 3 and the self-healing elimination strategies which are shown in Figure 2-13 and in Figure 2-14. The first elimination strategy which is shown in Figure 2-13 is comprised of nine bio-inspired functional cells organized as three rows [45]. Four of them highlighted with a white color background are working collaboratively in row1 to carry out the functionality of one task of the critical application. However, the other four neighboring spare cells located in row2 and highlighted with the gray color background are used as a self-healing elimination strategy against the death of the working cells defected by permanent faults. In addition, the ninth functional cell located in row3 and highlighted with the black color background was added as second line of defense to repair the failure occurrence in any one of the four pre-generated spare cells in row2.

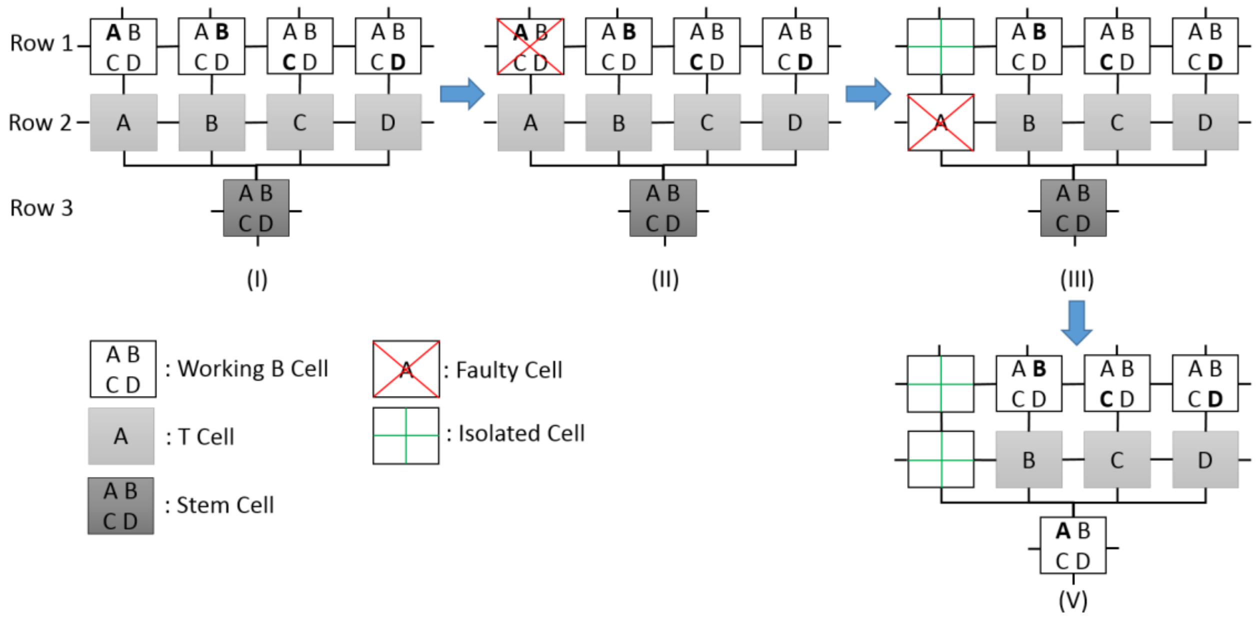


Figure 2-13: The first proposed elimination strategy and process of self-repair.

The second elimination strategy which is shown in Figure 2-14 is comprised of twenty bio-inspired functional cells organized as four rows [46]. Eight of them highlighted with a white color background are working collaboratively in row1 and row2 to carry out the functionality of one task of the critical application. However, the other eight neighboring spare cells located in row1, row2, row3, and row4 and highlighted with the gray color background are used as a self-healing elimination strategy against the death of the working cells defected by permanent faults. In addition, the other four functional cells located in row1 and in row4 and highlighted with the black color background was added as second line of defense to repair the failure occurrence in any one of the eight pre-generated spare cells.

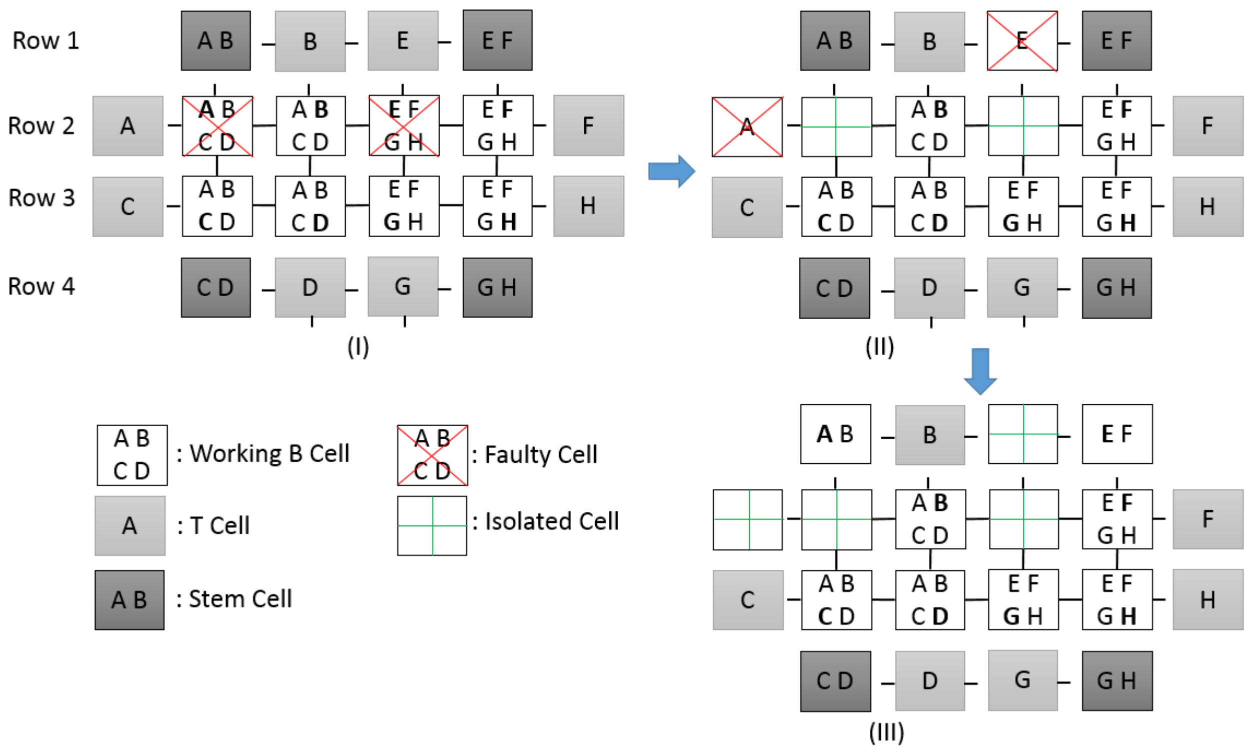


Figure 2-14: The second proposed elimination strategy and process of self-repair.

CHAPTER 3

THE BIOLOGICALLY-INSPIRED SELF-HEALING ARCHITECTURE - BIOSYMPLE

In this chapter, the design objectives for the proposed biologically-inspired self-healing hardware architecture called BioSymPLe, its architectural design principles, and how its abstraction layers are working cooperatively to perform functional and safety objectives for the critical application are presented.

3.1 Design Objectives for BioSymPLe

BioSymPLe is motivated by the challenge of developing evolvable self-healing HW architectures that are inherently resilient, accessible by the process control industry, and capable of being verified for highly safety critical applications. The nexus of these domains creates design opportunities, challenges, and conflicting design spaces, and has been up to now largely uncharted. Below we list out the main design objectives for our proposed approach.

- *Multilayered Approach to Self-Healing*– BioSymPLe aims to use a hierarchical multi-layered self-healing approach to model the dependable behavior of the critical application. This approach will make the critical system consists of a number of layers working cooperatively to perform the functional and safety objectives. For example, one layer of the architecture is used to execute the critical functions of the application and another layer is monitoring the correct operation and triggering self-healing mechanisms when the functions layer fails. This interacting process between different layers will lead to achieving high levels of reliability and robustness.

- *Tolerance of Realistic Fault models*– The process of building a realistic fault model for BioSymPLe is a complex process that requires assuming a complete list of fault classes. These different classes of faults can cause a functional failure in the system which can lead to a catastrophic incident in the environment, where the proposed architecture is operated. To make BioSymPLe resilient against the realistic fault model, we have to develop and adopt fault models appropriate for most industrial applications. Also, all the sources for the physical and logical faults such as power fluctuations, Electromagnetic Interference (EMI), and latch-up have to be identified.

- *Compositional Resiliency*– Our aim with the BioSymPle architecture is to provide basic building blocks and well-defined self-healing functions and services. From these building blocks and self-healing services we can allow the designer to compose application functionality and support resilience at the task and system levels-where it is most needed.

- *Engineer Accessible*: Safety Critical Automation Industry has reluctant to consider new technologies such as field programmable gate arrays (FPGAs) for industrial process control applications. The primary reason is that typically, manufacturing plant engineers have no training in advanced digital system design or the tools for FPGA system design. One of our goals for BioSymPLe, is to provide an "engineer accessible" Function Block programming architecture that allows plant engineers to program it. Thus it needs no extensive training for plant engineers, as this configured FPGA emulates a conventional PLC with added advantages of resiliency and self-healing.

- *Verifiable Behavior–* The last design objective of developing BioSymPLe is to make the functional and safety properties verifiable against the design requirements. As a fully verifiable self-healing hardware architecture using mathematical formal methods, it will support the constrained execution of elementary bio-inspired functional cells.

3.2 Biologically-inspired Model for BioSymPLe

Our biologically-inspired model is guided by a combination of two principles: *biologically-inspired concepts* and *architectural design principles*. The three biological concepts we found to be promising in terms of complementary resilience are

1) The biological cell life cycle (see Figure 3-1).

2) The biological stem cell differentiation (see Figure 3-2).

3) The pathway from DNA to protein (see Figure 3-3).

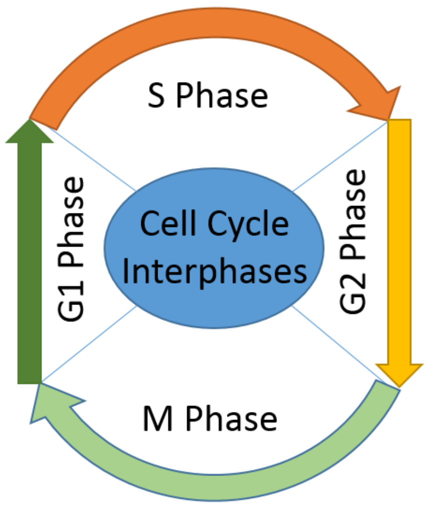


Figure 3-1: Biological cell life cycle.

Within each of these three biological techniques, there are diverse self-healing and self-diagnostic mechanisms used by the living organism to provide overall resilience [15], [47], [48]. For instance, the cell life cycle (see Figure 3-1) is considered to be a self-diagnostic mechanism used inside the living cell in order to make the cell capable of continuously monitoring its chemical internal state through four phases of testing integrity [47], [48].

If the living cell is attacked by an invader causing a weak mutation, the effects of this mutation will be tolerated using a special enzyme. However, if the mutation is strong such that it cannot be tolerated or repaired inside the cell, the cell cycle will lead to a cell death and it will notify the immune system to start working.

The immune system (see Figure 3-2), will generate two types of cells: adult stem cells which, are divided to produce more stem cells (self-renewal), and embryonic stem cells that can be divided in order to generate differentiated cells [47]. Adult stem cells can detect the infection using B-cells, and fix it or kill it using another type of cell called T-cells. In addition, differentiated embryonic stem cells can become any specific functional cell based on the gene expression that determines which gene is active or not on the DNA molecule (see Figure 3-3) [47], [48].

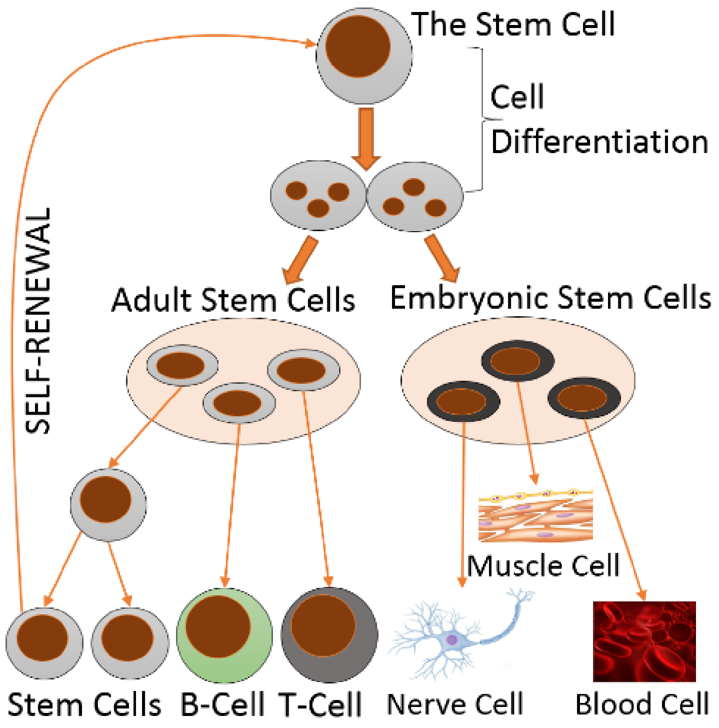


Figure 3-2: Biological stem cell differentiation.

The pathway from DNA to protein shown in Figure 3-3 where the genetic strands located on the DNA molecule are expressed through two biological processes: transcription and translation, to generate a collection of amino acids (proteins) that specify the function of the divided cell based on their shape and their position in the living organism [47]. All of these genetic codes are stored inside the genome that contains that entire organism's DNA. However, not all these genes are activated inside the cell at one time to determine the behavior of the organism. Some of these genes are on and some of them are off and that is based on the location of the biological cell in the living organism [47], [48].

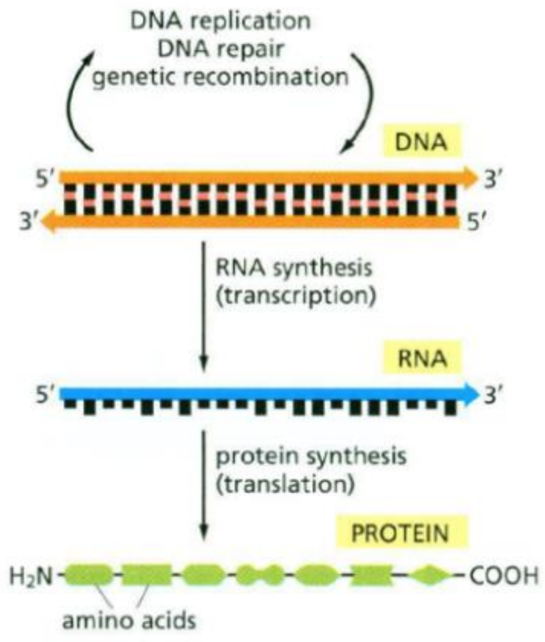


Figure 3-3: The pathway from DNA to protein.

3.3 Architectural Design Principles for BioSymPLe

Architectural design principles establish the basis of the formulation of operational rules, rules for organization, and rules for composition for any architecture. Jackson [49] lists 4 categories of biologically-inspired resilience attributes namely; *capacity*, *flexibility*, *tolerance*, and *inter-element collaboration* to achieve resilience in architecting systems [49], [37]. We extend the aforementioned attributes with 3 additional principles:-

1) Reorganization. Flexibility is defined as the ability of a system to undergo changes with relative ease in operation while the system is experiencing some external disturbances in its environment. In contrast, reorganization principle says that the system should be able to cope with uncertainty and be stable against some disruptions such as transient, intermittent, or permanent faults by detecting the failure in the unstable units, restructure itself and/or migrate its functionality to healthy units.

2) Separation of Concerns or Partitioning. Building self-contained functional units allows the disentangling of one huge homogenous system into a group of separate functional units. In turn, they can be grouped in self-contained architectural units to generate stable forms of functionality. Partitioning in the proposed architecture separates the healing layers and Input/Output (I/O) digital devices from computational activities carried out inside service layers.

3) Independence. Independence in dependable embedded systems means that the digital system should be capable of observing all of its functional and safety behaviors and be able to perform the necessary instrumentation and control actions autonomously. In this research, this principle has been achieved by reducing the architectural functional units of the proposed architecture to their minimum representation as required by the critical application. More specifically, this principle is a fundamental architectural concept, most notably illustrated in the local and global healing layers.

The design principles listed above are used throughout the design to achieve *resilience-aware deterministic hardware architecture*, and prevent the architecture from becoming complex and feature rich.

3.4 Overview of BioSymPLe High Level Perspective

The BioSymPLe architecture with its three abstraction layers shown in Figure 3-4 (a) and the four hierarchical sublayers (see Figure 3-4 (b)) embedded inside the critical service layer is based in part on the way biological organisms achieve resiliency, and our architectural design principles. The high level perspective of BioSymPLe (see Figure 3-4(a)) is comprised of three principle divisions; (1) the Local Function Migration Layer (corresponding to the life cycle of the cell), that continuously monitor the correct operation of the functions being executed in the Critical Service Layer (CSL) and trigger reconfiguration elimination strategies when faults and failures are detected; (2) the Critical Service Layer (corresponding to B cells and T cells in the immune system), which is responsible for hosting/executing the intended functions for digital I&C critical application, and (3) the Global Function Migration Layer (corresponding to embryonic stem cells), which is responsible for monitoring the behavior of the functions but also triggering the required repair mechanisms to heal any faulty T cells present in the critical service layer.

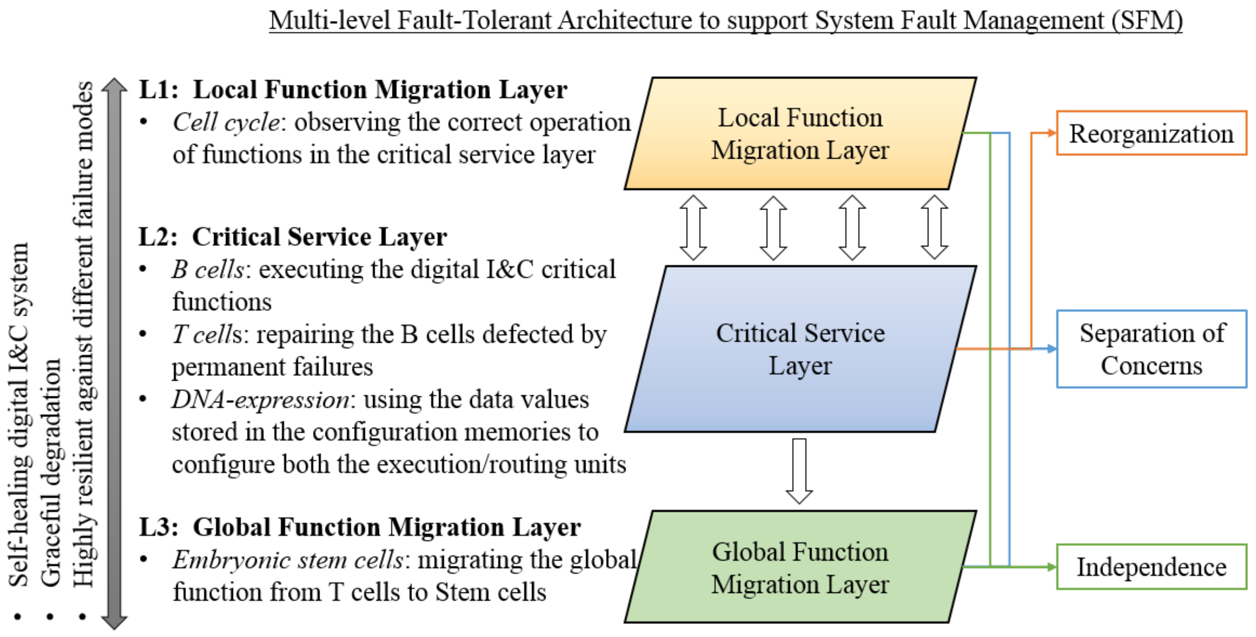

Multi-level Fault-Tolerant Architecture to support System Fault Management (SFM)
• Self-healing digital I&C system
• Graceful degradation
• Highly resilient against different failure modes
L1: Local Function Migration Layer
• Cell cycle: observing the correct operation of functions in the critical service layer
L2: Critical Service Layer
• B cells: executing the digital I&C critical functions
• T cells: repairing the B cells defected by permanent failures
• DNA-expression: using the data values stored in the configuration memories to configure both the execution/routing units
L3: Global Function Migration Layer
• Embryonic stem cells: migrating the global function from T cells to Stem cells
Local Function Migration Layer
Critical Service Layer
Global Function Migration Layer
Reorganization
Separation of Concerns
Independence

Figure 3-4 (a): High level perspective of the BioSymPLe architecture

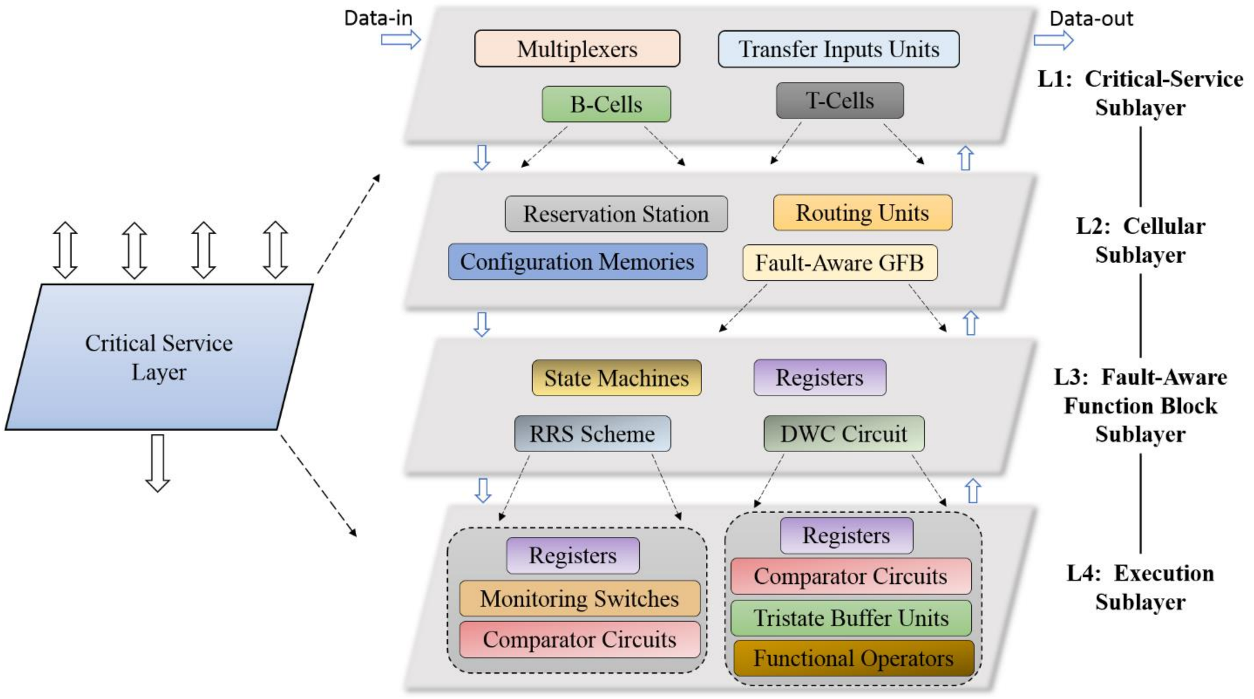


Figure 3-4 (b): The four organizational sublayers of the critical service layer

Figure 3-4 (b) presents the in internal structure of the critical service layer, which is organized in a top-down hierarchical approach and includes four sublayers: critical service sublayer from the top side, cellular sublayer, Fault-Aware function block sublayer, and execution sublayer from the bottom side. Each one of these four sublayers contains a group of hardware components that can be connected together to execute the application task. As example, the critical service layer contains a group of functional B cells and T cells, a transfer inputs unit, and multiplexers. The cellular layer represents the internal structure of each functional cell: B cell or T cell and contains routing units, a reservation station, a configuration memory, and a Fault-Aware function block. The Fault-Aware function block sublayer has a state machine, registers, Register Redundancy Scheme (RRS), and Duplication with Comparison (DWC) circuit. The last sublayer from the bottom embeds registers, monitoring switches, comparator circuits, tristate buffers, and functional operators.

3.4.1 Version1 Concept of BioSymPLe Architecture

In the first proposed architecture for BioSymPLe (see Figure 3-5), we utilize the three layers of abstraction presented in Figure 3-4(a) and the four sublayers of organization shown in Figure 3-4(b) to create interacting functional partitions to achieve overall system resilience. The *Critical Service Layer (CSL)* embeds four basic sublayers: critical service (L1), cellular (L2), Fault-Aware function block (L3), and execution (L4) (see Figure 3-4(b)) to execute the application functions. Specifically, it contains eight functional cells: *four* active B cells (designated *B* in Figure 3-5) used to execute the application based functions, and *four* passive pre-generated redundant T cells used as a healing mechanism for the faulty B cells (designated *T* in Figure 3-5).

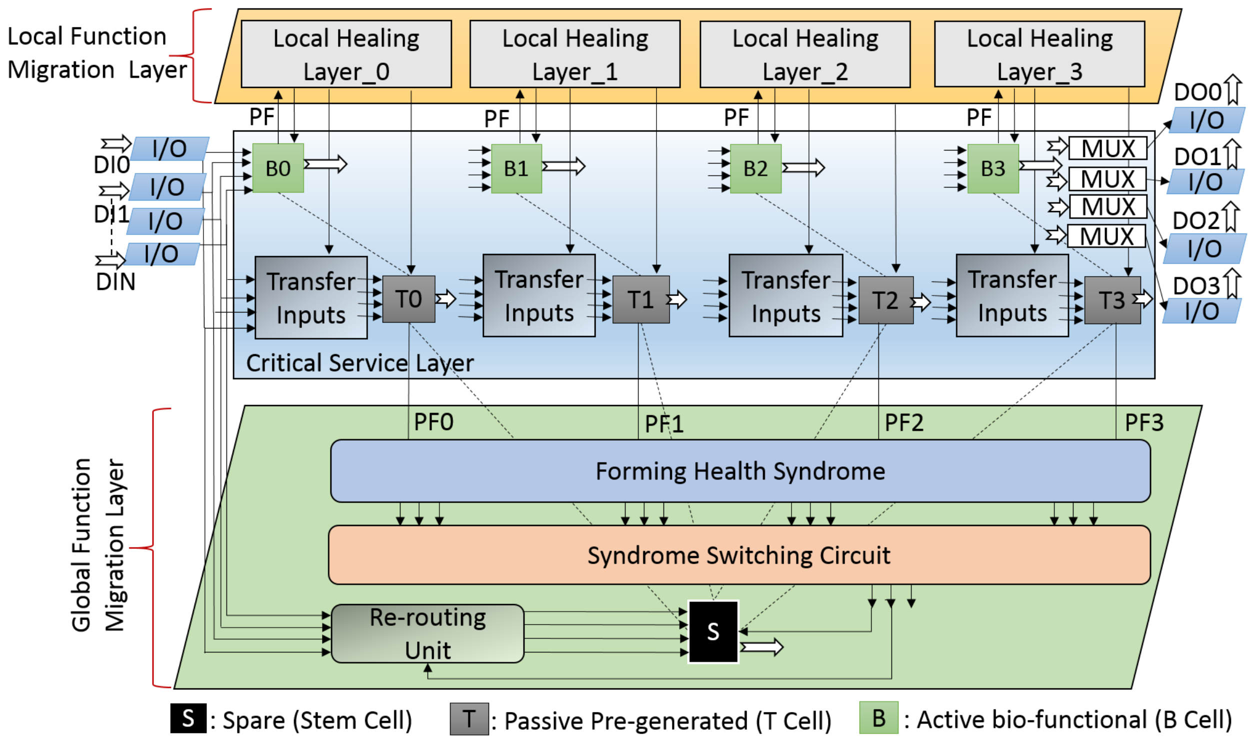


Figure 3-5: BioSymPLe architecture version1 concept

The correct execution of each B cell is monitored continuously by its neighboring local healing layer (designated *PF* in Figure 3-5), and once a fault is detected and determined to be transient, it is masked/tolerated using a Register Redundancy Scheme (RRS) embedded inside the cell. The RRS unit represents the first line of defense against the discovered transient faults defined as temporary deviations in the register values stored inside each cell [45].

3.4.2 Operational Detail of BioSymPLe

The BioSymPLe architecture is designed to realize the functionality of critical systems operating in harsh environments and radiation-induced transient faults that can occur at unpredictable times are the most prevalent fault type [50], [19], [20]. The process of detecting/correcting the transient faults at the register level inside the functional block is most effective and represents the first line of defense for BioSymPLe. If they are not tolerated at the register level in the functional block, their wrong values will be sensitized at the output signals of the bio-inspired cell, which is directly connected to the I/O ports in the external world and can impact the safety of the public or the environment. However, once a permanent fault is detected in any one of the four B cells, its neighboring *local healing layer* is notified and generates a health syndrome, which does three things:

(1) Deactivates the output of the faulty B cell (cell death).

(2) Reroutes digital input data from the faulty B cell and makes it available at the input of the healthy T cell (reorganization).

(3) Selects a genetic code stored in a configuration memory in the T cell so that the functionality of the defected cell is healed and performed by the healthy T cell (restoration).

The *local function migration layer* in Figure 3-5 represents the second line of defense against the permanent faults affecting one or all of the four B cells. As a third line of defense against the occurrence of additional permanent faults, the *global function migration layer* is responsible for fault management for the entire critical service layer and this layer consists of three components:

1) An embryonic stem cell - can be differentiated to repair any type of T cells in the critical layer (designated T0, T1, T2, and T3 in Figure 3-5).

2) A health syndrome unit - which continuously reads the error signals from the critical service layer.

3) A switching circuit - selects which one of the four syndromes, generated by the health unit, will be chosen to differentiate the embryonic stem cell.

The global layer fault management process in BioSymPLe is aimed at managing the processing capacity of the critical service layer. If too many faults occur, then we have too few resources to maintain operations. As such, the global layer monitors all of the cells concurrently based on the fault status and repair history of the critical layer. The global function migration layer employs *stem cells* to replace faulty/offline B cells, and T cells. The stem cell is only used when most of the self-healing resources (T cells) of the critical layer have been used up due to fault repair actions. As an example, a stem cell can be differentiated either in the case of the failure one of the four local healing layers to repair the B cell with its neighboring T cell, or in the case of the occurrence of two permanent faults in a B cell and a T cell sequentially. This process is imitating how the embryonic stem cell is differentiated in the case of the failure of the immune system in generating the T cells as a first line of defense against the invader. The global function migration layer can produce twelve control signals for the recovery of each faulty T cell and the syndrome switching circuit selects which one of these three signals is used to configure the stem cell.

This configuration process is based on the location of fault occurrence in BioSymPLe architecture. BioSymPLe is an architectural concept that leverages existing programmable and configurable hardware technologies such as Field Programmable Gate Array (FPGA) technology, and Application Specific Integrated Circuits (ASICs). Many studies (and marketplace products) have confirmed that FPGA technology is a viable and competitive option for various application domains: aerospace, nuclear industry, and industrial control systems due to their characteristics with regard to re-programmability, high performance, concurrency, and low-cost development [51], [52], [8]. As a consequence, hardware implementation of this proposed architecture is realized on FPGAs.

3.5 The Biologically-inspired Cell

As it is shown in the second cellular sublayer (L2) of the BioSymPLe architecture (see Figure 3-4 (b)), each one of the functional cells: B cell, T cell, and Stem cell has a similar internal structure that contains: Fault-Aware GFB, configuration memories, reservation stations, and routing units (see Figure 3-6). However, their functionalities are different and based on the expression for the genetic codes stored inside the configuration memories. Each cell has its own configuration memory, highlighted with the blue color in (Figure 3-7), and imitates the operation of the genome in cellular biology. It typically contains a number of genetic codes represented by hexadecimal numbers, equal to the number of different functional cells that are connected to implement the critical application on BioSymPLe architecture. The amount of the configuration genetic codes stored in the configuration memories of the B cells is different from the amount of the genetic codes stored in the configuration memories of the T cells and stem cells.

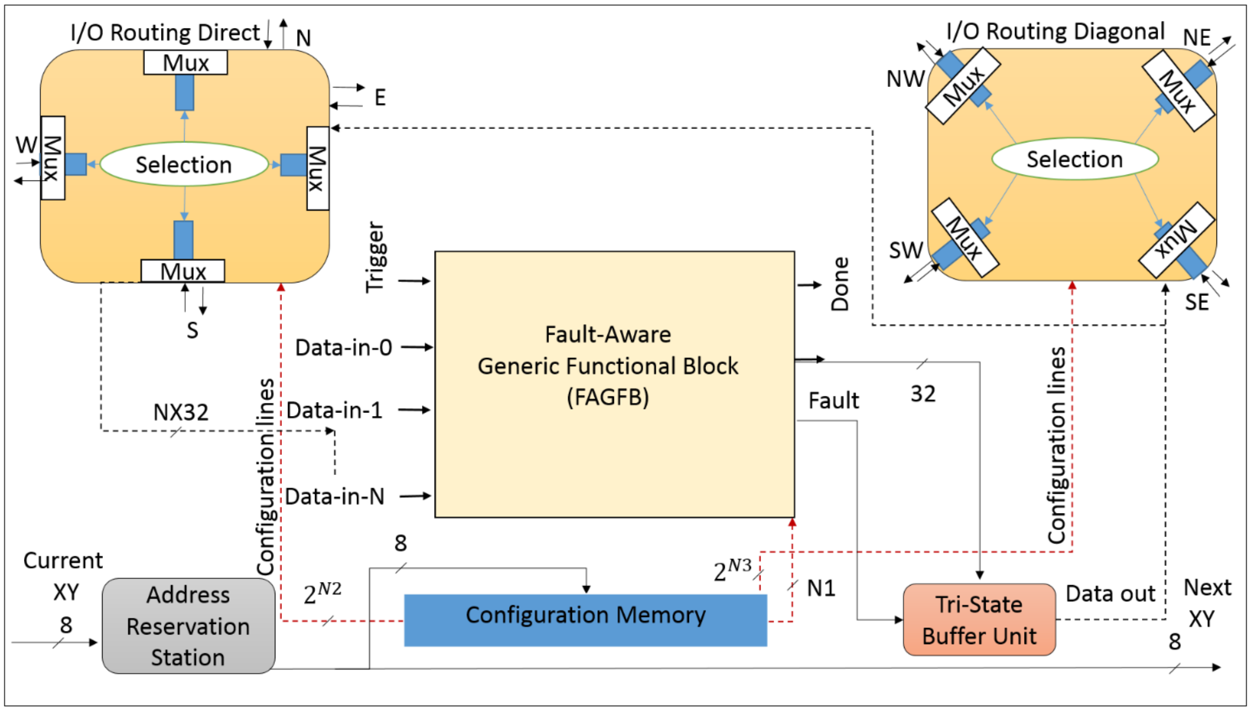
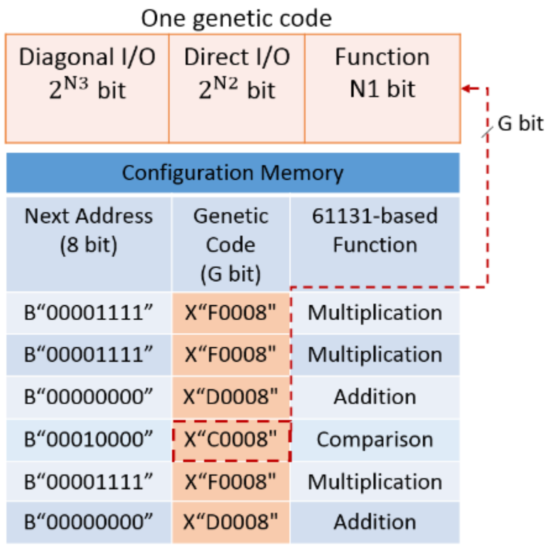


Figure 3-6: The internal structure of the biological-inspired cell



Diagonal I/O 2^{N3} bit	Direct I/O 2^{N2} bit	Function N1 bit

Configuration Memory		
Next Address (8 bit)	Genetic Code (G bit)	61131-based Function
B"00001111"	X"F0008"	Multiplication
B"00001111"	X"F0008"	Multiplication
B"00000000"	X"D0008"	Addition
B"00010000"	X"C0008"	Comparison
B"00001111"	X"F0008"	Multiplication
B"00000000"	X"D0008"	Addition

Figure 3-7: The internal structure of the configuration memory

The basic structure of each cell is comprised of two partitions. The first partition is the data flow, which includes the data path for the Fault-Aware Generic Functional Block (FAGFB) that will be explained in section 3.6, direct I/O routing unit, and diagonal I/O routing unit to make each cell capable of being connected to its neighboring cells from north, south, east, and west. The second partition is the control flow, which embeds a reservation station to store the address of the cell, a configuration memory to store the genetic codes that configure both the FAGFB execution unit and the two routing units, and a tristate buffer unit used to disconnect the faulty cell from its neighboring cells. The cell address is used to select the genetic code in the configuration memory.

Furthermore, the N input registers located inside the FAGFB unit are enabled to read the input digital data through the I/O routing units, executing the functionality of the task, writing the resulted data into the output register, and sending a 'done' signal to a control unit. Since the global function of the critical application requires different types of functions (logical, arithmetic, bitwise, selection, and relational operators), different genetic codes are stored inside the configuration memory (genome) of each cell in the system.

3.6 Fault-Aware Generic Functional Block (FAGFB) Semantics

BioSymPLe was designed to be accessible by the industrial automation community and one of the means this is achieved is by its adopting PLC programming constructs. International Electrotechnical Commission IEC 61131-3 is an international standard for Programmable Logic Controllers with respect to Programming languages. The main purpose of IEC 61131-3 is to define a consistent and complete notational syntax with semantics for a suite of PLC programming languages [53], [54]. One of the more widely used programming methods in PLCs is Function Block Diagram Programming. Function Block (FB) programming provides a visual and graphical syntax to describe the logic and functionality of PLC program – via interconnected elementary function blocks. This paradigm seemed appropriate for BioSymPle, as we envisioned a library of PLC functions that a designer could use to create an application in the critical service layer. Because IEC 61131-3 is a software based standard – it does not exactly translate to hardware execution architectures [55].

In the third Fault-Aware function block sublayer (L3) of the BioSymPLe architecture (see Figure 3-4 (b)), Our approach to creating a function block programing context for BioSymPle was driven by looking at the specification of function blocks for both IEC 61131-3 and IEC 61499. From this activity, we created a “pseudo” 61131 function block design we call the Fault-Aware Generic Function Block (FAGFB) illustrated in Figure 3-8.

Referring to Figure 3-8, a FAGFB is divided into two divisions: the state machine-based control flow and the data flow. The data flow division is where traditional PLC Function block units reside and execute– like AND, OR, COMP, LESS THAN, etc (see Table 3-1)… According to the two mentioned standards, the FAGFB can only be activated with the occurrence of a “Trigger” input event signal and then the control flow division activates the data flow (executes the function).

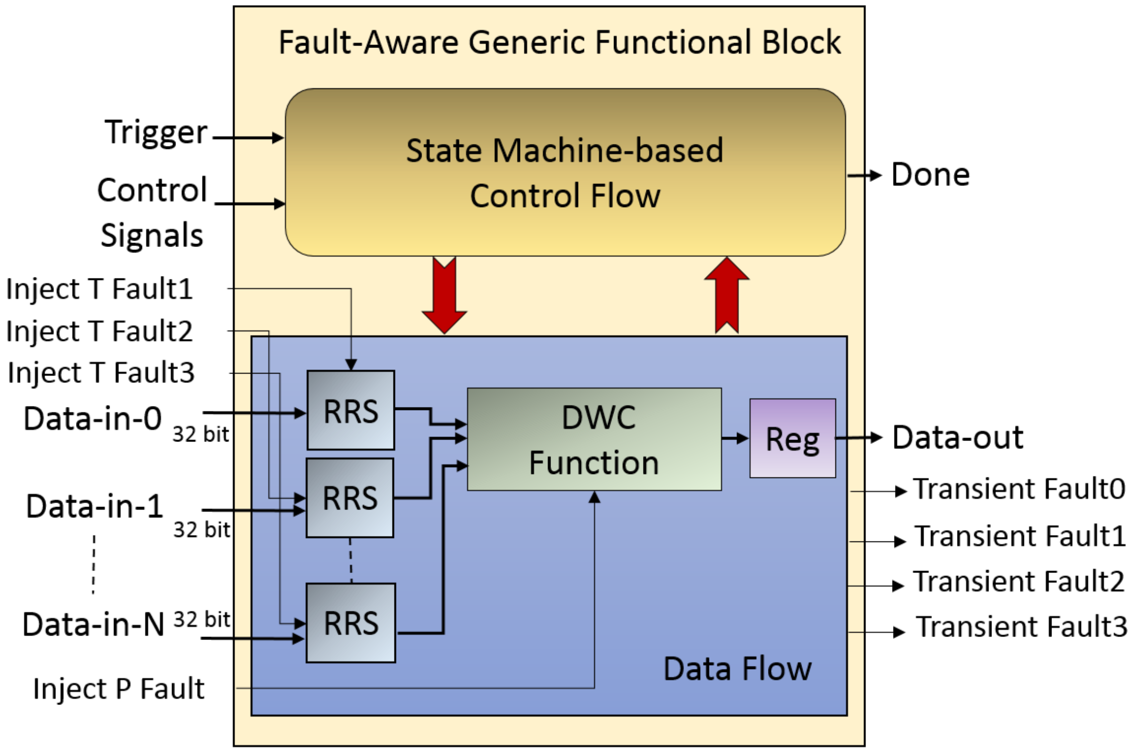


Figure 3-8: The Fault-aware Generic functional block

TABLE 3-1
FUNCTIONALITY EXECUTED IN THE DATA FLOW DIVISION

INSTRUCTION	DESCRIPTION
AND, OR, NAND, NOR, XOR, XNOR, NOT	Logical Operators
ADDITION, SUBTRACTION, MULTIPLICATION, DIVISION	Arithmetic Operators
MAX, MIN MUX	Selection Operators
LT, LE, GT, GE, EQ	Comparison Operators

The data flow division is comprised of N input registers, a functional execution unit, and one output register. The functional execution unit is capable of implementing one function of up to N variables. Where N represents the maximum number of input data lines that can be connected to the data flow unit inside the functional cell. Each local function implemented in each cell is assumed to be a part of a larger function. To describe the relations between the genetic code size, input data lines, and selection lines for the routing units inside the functional cell, in a notational form, we assume that:

G: represents the size of each genetic code stored in the configuration memory. Basically, the size of the configuration memory depends on the number of three parts: the input data lines, the number of selection lines for the four multiplexers of the direct routing unit, and the number of selection lines for the four multiplexers of the diagonal routing unit.

N: represents the maximum number of input data lines that can be connected to the FAGFB unit inside the functional cell. This number of data lines depend on the number of selection lines for the four multiplexers embedded inside the routing units due to that increasing the number of selection lines will enable the functional cell to be connected to more functional cells in the system. All the outputs of these functional cells must be routed to the FAGFB execution unit through the direct and diagonal routing units.

These calculations can be seen in equation (3-1) and (3-2), respectively.

$$G = N1 + N2 * 4 + \mathrm{N3} * 4 \qquad (3 - 1)$$

$$N = 2N2 + 2N3 \qquad (3 - 2)$$

Where:-

N1: represents the number of selection lines being connected to the PLC-based FAGFB unit to select the operational function. The selection lines of the FAGFB unit is connected to the lowest part of the output for the configuration memory.

N2: represents the number of selection lines for each multiplexer inside the direct I/O routing unit.

N3: represents the number of selection lines for each multiplexer inside the diagonal I/O routing unit.

3.7 Fault-Tolerance Techniques for FAGFB

Fault tolerance techniques based on redundancy are typically used to detect and correct fault occurrences in critical systems. A key decision early-on is identifying the appropriate fault detection strategy which can be designed using one of the redundancy techniques. The redundancy technique is defined as an addition of the software, hardware resources, spatial, informational or time beyond what is need for the normal operation of the critical system [22], [56]. The physical replication of hardware resources is one of the most common methods of redundancy that have been used in critical system due to the cheap price and small size of semiconductor components being used in the VLSI circuits technology [22].

Hardware redundancy can be classified into three basic forms: passive, active, and hybrid. Passive redundancy techniques use the concept of fault masking to tolerate the effects of the fault occurrence and prevent the fault from generating an error [22]. The second approach which is called active or dynamic redundancy is used to detect the fault occurrence by using some techniques such as spare resources, faulty component replacement-based reconfiguration, and restoration the system into a fully operational state [22]. Hybrid redundancy techniques combine the two approaches: passive and active redundancy techniques to achieve a maximum fault tolerance within a system with a minimum amount of hardware resources. However, this approach is the most expensive form of redundancy techniques that can be used in safety critical systems [22], [57].

Different fault classes can occur in VLSI circuits like transient faults that may occur in sequential logic circuits, or permanent faults occurring in combinational logic circuits [22], [58], [3]. Fault tolerance involves design of a system or a subsystem in such a way that, even if certain types of faults occur, a fault will not propagate through the system and induce a failure in the system.

In the fourth execution sublayer (L4) of the BioSymPLe architecture (see Figure 3-4 (b)), the Fault-Aware Generic Functional Block (FAGFB) combines two fault tolerance techniques: *Active and Passive redundancy*, to address transient and permanent faults [3]. In BioSymPLe, two fault tolerance mechanisms are embedded in the design of the B cell and T cell to enhance the reliability. Firstly, we employ Duplication with a Comparison (DWC) unit shown in Figure 3-9, which imitates the operation of B cells in the immune system - that attack the antigens in their local locations to detect the occurrence of permanent faults inside the 61131 functional units. This unit is comprised of two duplicated 61131 function units, a comparison circuit, a register and a tri state buffer unit as is shown in Figure 3-9 and in Figure 3-4 (b).

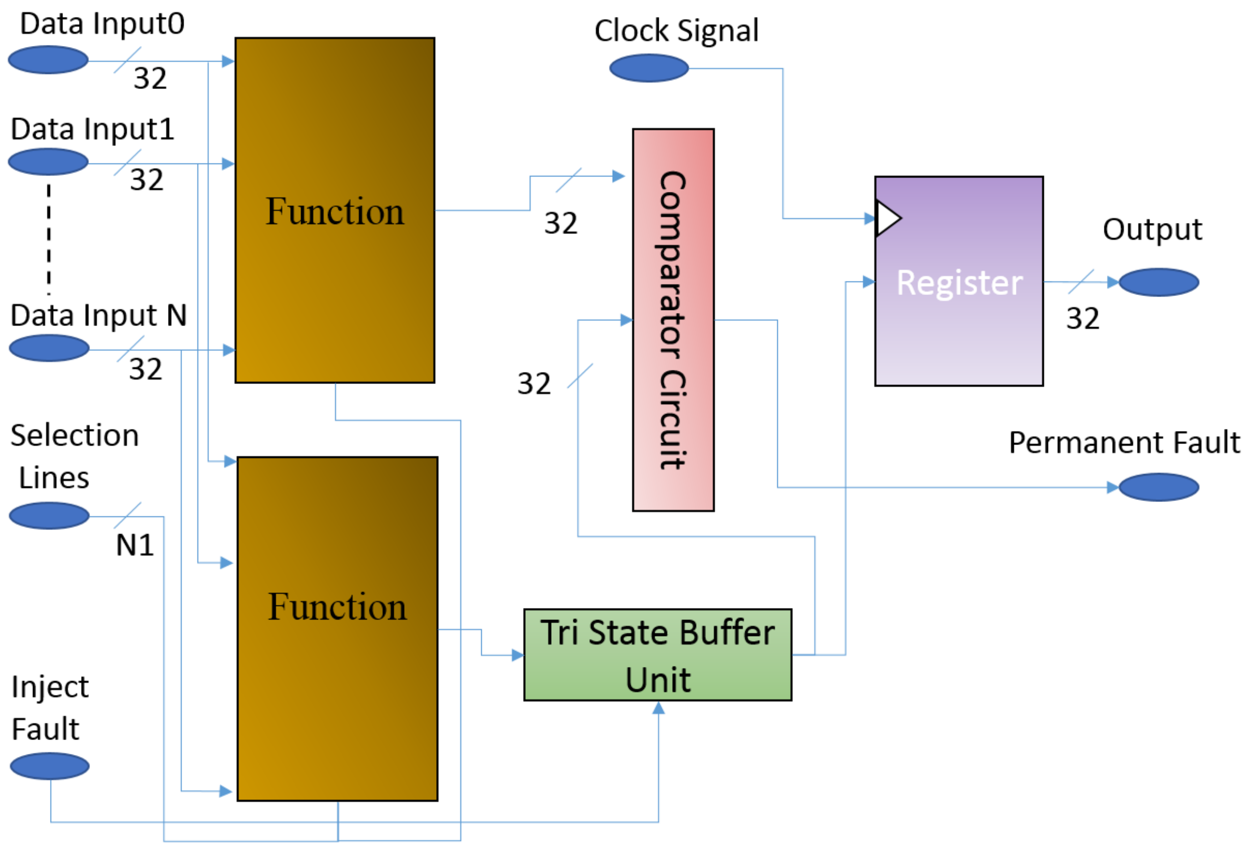


Figure 3-9: Duplication with comparison functional unit

Secondly, as it can be seen in Figure 3-10 and in Figure 3-4 (b), a register redundancy scheme was designed as an active redundancy technique to tolerate the effects of transient faults that may defect the input registers for the FAGFBs. The DWC functional unit and the register redundancy scheme are embedded inside each functional cell [59]. Each register redundancy module consists of eight hardware components: three registers, three error detection units, a monitoring switch, and a comparator circuit. A transient fault was simultaneously injected into four input registers of the four register redundancy scheme (RRSs) embedded inside the FAGFB in which transient faults can be tolerated sequentially for an unlimited number of times using a self-monitoring switching unit.

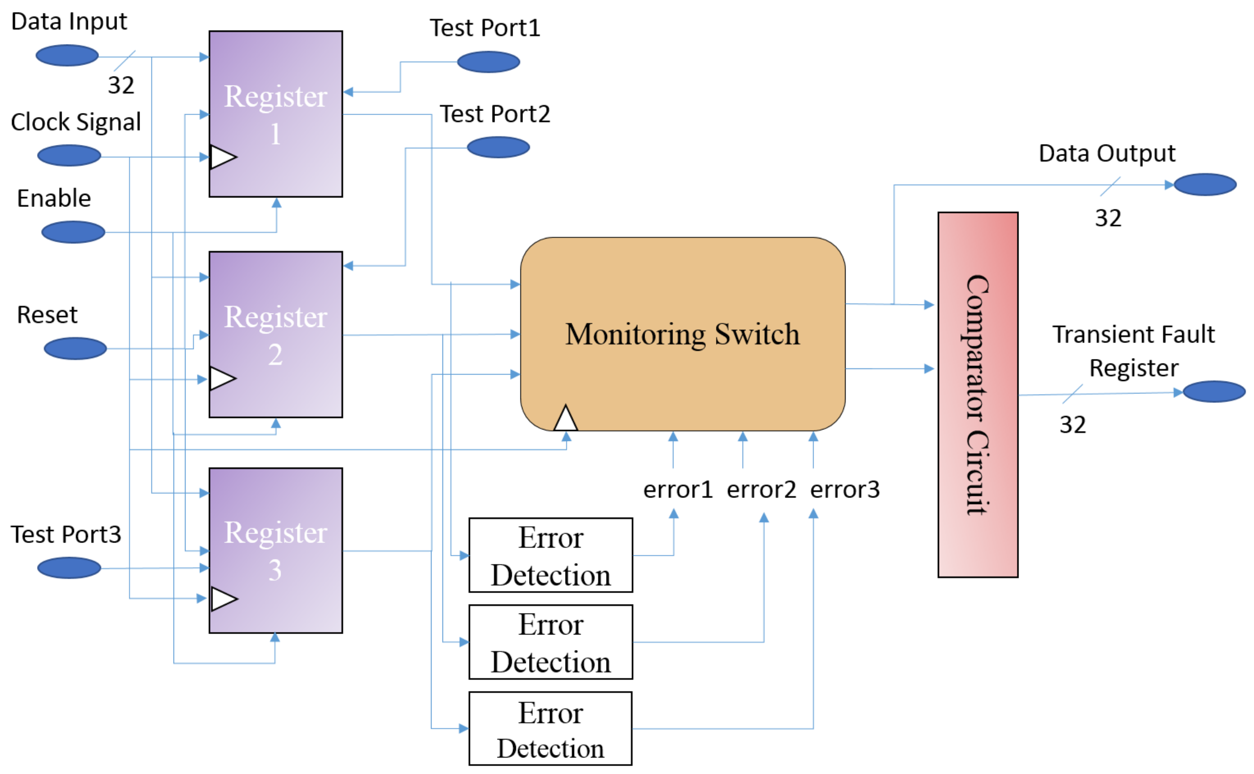


Figure 3-10: Register redundancy scheme

This switching machine continuously reads the status of the error signals produced by three triplicated duplication with comparison (DWC) register based models. The concept of register redundancy scheme is based on the operation of active redundancy in which a hot spare unit is taking over the functionality of the defective unit. Normally, the combinational logical circuits are vulnerable to be defected by permanent failure modes (Hard errors) and the sequential logical circuits are susceptible to transient faults (soft errors). Once the trigger signal connected to the FAGFB unit is activated, this unit goes into a (start state) which takes one clock cycle to complete the initialization process of the FAGFB. After that, one clock cycle is taken to activate the four parallel input registers (Read state), and then the (execution state) takes one cycle for executing the functionality by the FAGFB. In (done state), the output is produced at the output register. That means four clock cycles are required at least for producing the output and raising the done signal after the trigger is activated.

3.8 Reconfiguration by Graceful Degradation

To enhance the reliability of the proposed architecture and increase the fault detection coverage, more levels of fault-tolerance techniques have been added at the critical service layer before notifying the neighboring healing layers of the BioSymPLe architecture. As it is shown in Figure 3-11, three groups of functional bio-inspired cells (GCs) have been embedded in the critical service layer and each group contains three functional B-cells which were connected to perform a voting by majority.

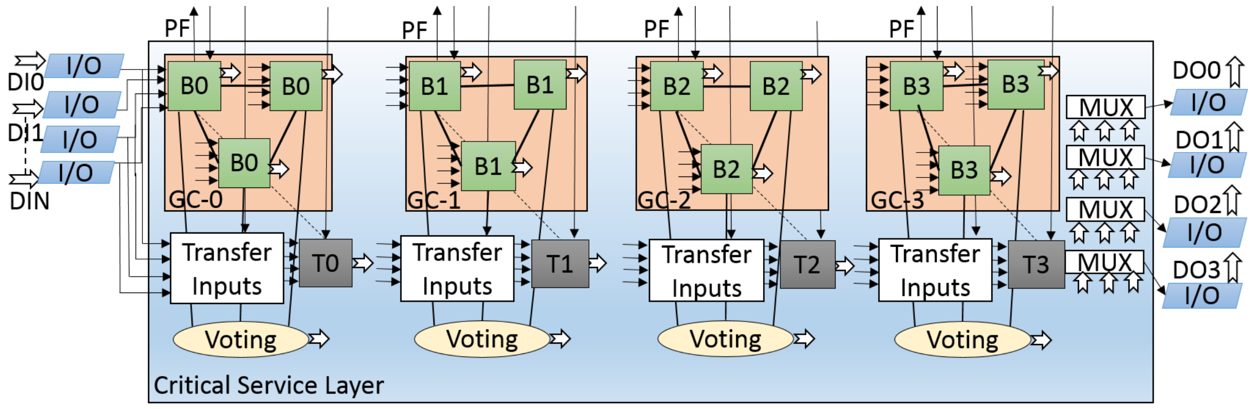


Figure 3-11: Reconfiguration by graceful degradation

Voting by Majority circuit typically determines which input value appears more often than all of the others. The Voting by Majority circuit that was designed (see Figure 3-12) consists of three groups of hardware components: three replicated functional cells (FC-A, FC-B, FC-C) executing the same function at one time and requires cells to reach agreement on which cell goes faulty and has to be removed, a state machine based control unit that reads the data outputs of the three functional cells (V1, V2, V3) through a switching circuit, and a decoding circuit that reads three control signals (A, B, C) and sends back to the cells a syndrome of feedback signals connected as enabling signals for a network of tristate buffers embedded inside the switching circuit. In the BioSymPLe architecture, the system begins with all three functional cells operational (assume identical cells) in each GC group. On the first fault detected by the voter, the system analyzes the errors from the voter and diagnoses which cell is faulty. As an example, the state machine based control unit continuously reads a vector of three error signals (error1, error2, error3) produced by the faulty cells. The switching circuit disconnect the faulty cells from the voting process whenever a permanent fault is detected inside the cell. Table 3-2 summarizes how the state machine was designed as a controlling unit for the voting by majority circuit. In this table, the graceful degradation: degrading (from Triple Modular Redundancy TMR to Duplex to Simplex) can be achieved based on the status of a vector of three error signals. For example, in the simplex mode in which only one data input value is connected to the output data, an event of two sequential permanent faults defecting two cells at different times have been assumed as a condition to reach this mode.

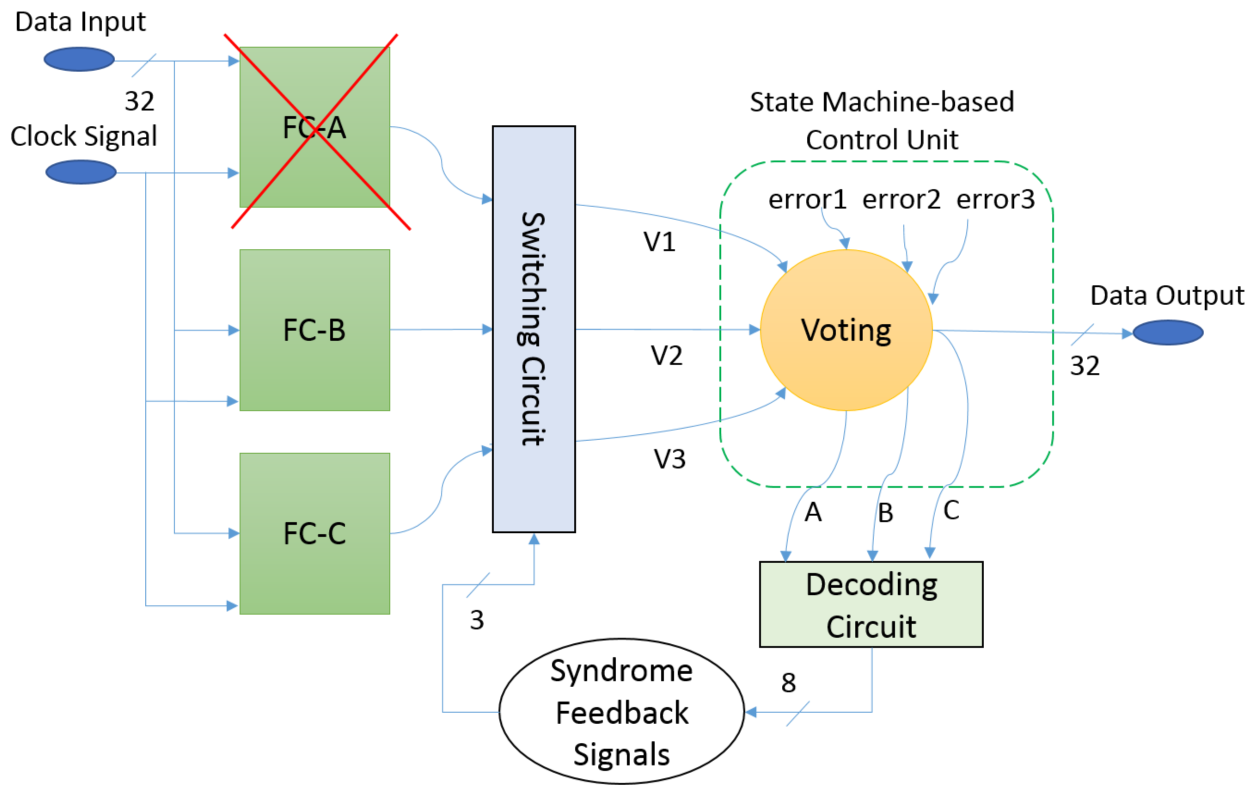


Figure 3-12: Voting on one lane of BioSymPLe

TABLE 3-2
STATE MACHINE BASED TRUTH TABLE

Error signals (Error3, Error2, Error1)			DATA INPUT			DATA OUTPUT =	STATE	MODE
0	0	0	V3	V2	V1	V1 = V2 = V3	S_1	Triple Modular Redundancy
0	0	1	V3	V2	V1	V2 = V3	S_2	Duplication with Comparison
0	1	0	V3	V2	V1	V1 = V3	S_3	
1	0	0	V3	V2	V1	V1 = V2	S_4	
0	1	1	V3	V2	V1	V3	S_21	Simplex
0	1	1	V3	V2	V1	V3	S_31	
1	1	0	V3	V2	V1	V1	S_41	
1	0	1	V3	V2	V1	V2	S_22	
1	1	0	V3	V2	V1	V1	S_32	
1	0	1	V3	V2	V1	V2	S_42	

On the second permanent fault, the same process occurs but the system degrades from a duplex to a simplex in a way that the voter and the remaining good functional cells collaborate to remove the faulty cell from the system, which simulates the process of Reconfiguration by Degradation (RD). Consequently, the main benefits of this proposed fault tolerance approach is that the faulty cells cannot participate in the consensus process among the healthy cells. For example, the faulty cells will be producing unpredictable values if it is left un-reconfigurable and these wrong values can impact the agreement process executing inside the voter. Secondly, instead of healing only against four sequential permanent faults (4 PFs) inside the critical service layer (CSL) of BioSymPLe, Grouping three functional cells (3 FCs) as fault confinement regions that don’t propagate the internal failures to their neighboring groups of cells (GCs) makes the proposed architecture more resilient against more number of transient and permanent failure modes.

3.9 Scan Time

The elapsed time from a change in an input I/O routing unit to the resulting change in the output I/O routing unit is dictated by the B cell scan time. The biologically-inspired functional B cell has multiple embedded hardware components such as Fault-Aware Generic Functional Block (FAGFB), direct I/O routing unit, diagonal I/O routing unit, Configuration Memory (CM), Reservation Unit (RU), and Buffer Unit (BU); therefore, an organized operation must be embedded inside the functional cell so that the 1131 execution on the four 32 bit digital data inputs is performed correctly. The data that can be interfaced with the input ports can be constants or signals of type signed 32-bit integer variables. Figure 3-13 illustrates the four phases of the B cell scan cycle.

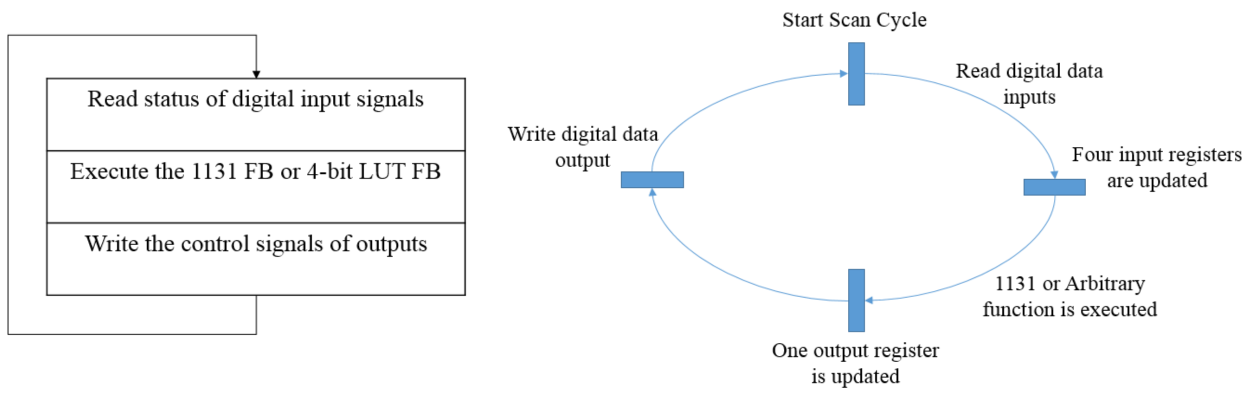

Read status of digital input signals
Execute the 1131 FB or 4-bit LUT FB
Write the control signals of outputs
Start Scan Cycle
Read digital data inputs
Four input registers are updated
1131 or Arbitrary function is executed
One output register is updated
Write digital data output

Figure 3-13: The functional B cell scan cycle.

3.10 Execution of Functional Cells

The association of both the data and the event digital signals can be either in sequential execution or in parallel execution and that that is based on the data dependency among the functional cells and the functional requirements for the critical application. In sequential case as shown in Figure 3-14, the done signal of FC1 drives the trigger signal of FC2 and the done signal of FC2 drives the trigger signal of FC3, and so on. However, in the parallel case, the trigger signals of the four functional cells FC1, FC2, FC3, and FC4 are being activated at the same instance of time. To develop a sequence of functional cell diagram (FCD) program that can solve a simple control problem using the proposed BioSymPLe PLC architecture, one output valve and five digital sensors are assumed to be interfaced with the architecture as it is shown in Figure 3-15:

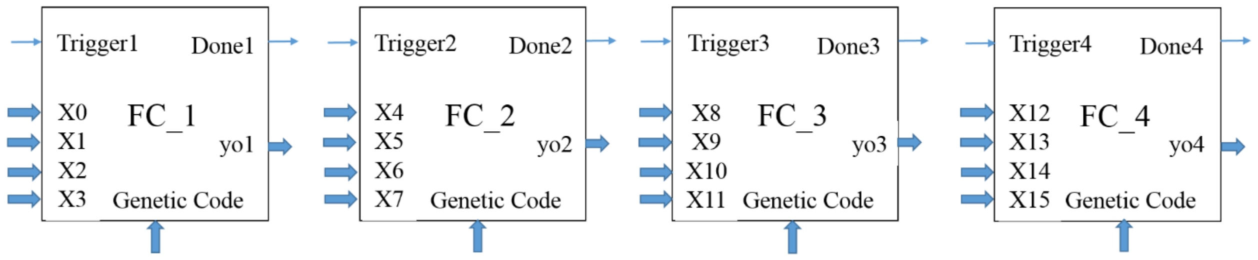


Figure 3-14: Event and data interface of four sequential B cells.

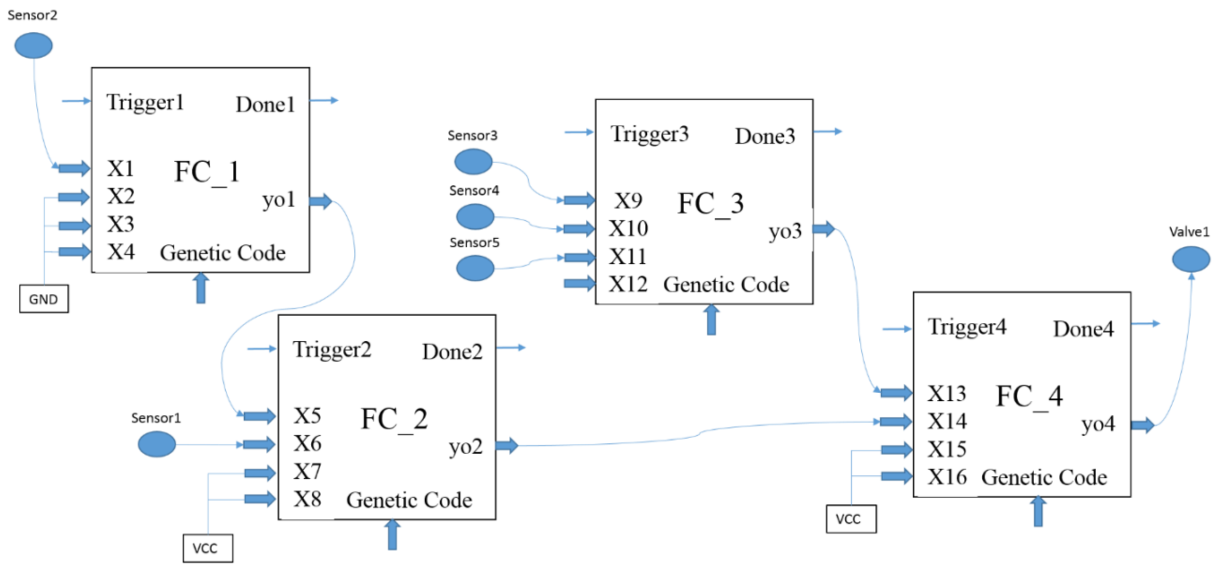


Figure 3-15: Event and data interface of four parallel B cells.

3.11 Communication Models

To define the communication models, this section specifies concurrency and interaction between concurrent and sequential tasks implemented on BioSymPLe Machines (BSMs). Concurrency can be expressed at various levels of granularity from circuit level to system level. Each BioSymPLe Machine, which contains one local function migration layer, one critical service layer, and one global function migration layer in the field of distributed dependable embedded systems can either operate under a centralized control of a single control unit or work independently.

3.11.1 Single Control Multiple Data (SCMD)

If there is a single Global Control Unit (GCU) that gives the same Writeable Control Register (WCR) to various BSMs that work on different data, the model is referred to as Single Control stream Multiple Data stream (SCMD) as it is shown in Figure 3-16.

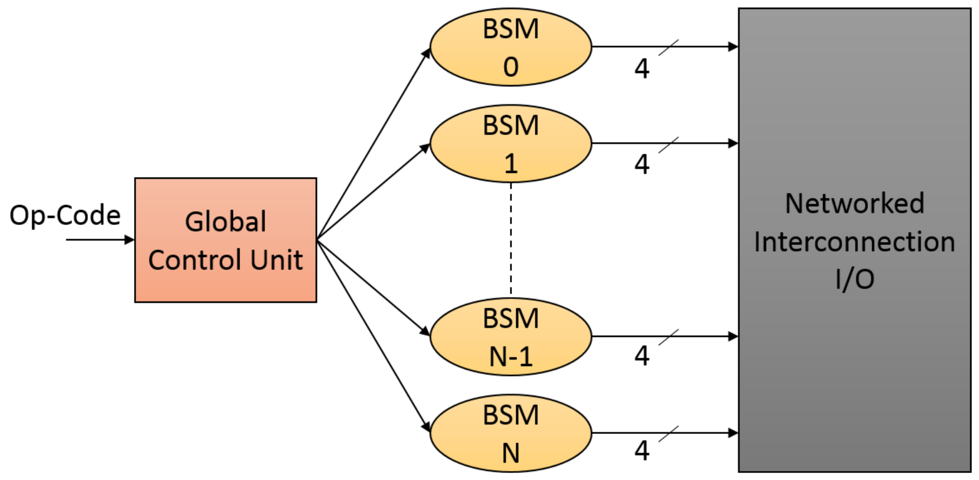


Figure 3-16: Single control multiple data (SCMD) communication model

3.11.2 Multiple Control Multiple Data (MCMD)

However, this model is called Multiple Control stream, Multiple Data stream (MCMD) (see Figure 3-17), if each BSM machine has its own Local Control Unit (LCU) and each BSM machine can execute different functions on different data items.

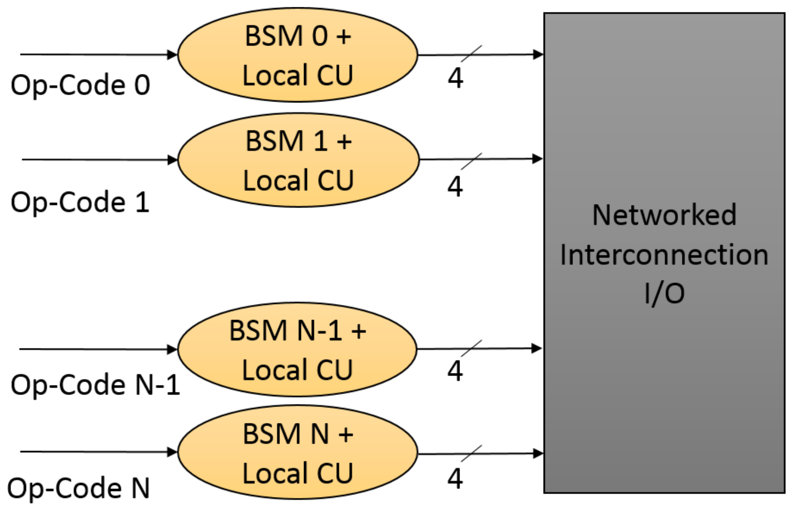


Figure 3-17: Multiple control multiple data (MCMD) communication model

3.11.3 Control Registers

Two types of registers will be embedded inside each BSM machine in order to facilitate the communication process between different tasks. These registers are writeable control register (WCR) and Readable Status Register (RSR). WCR shown in Figure 3-18 (a), is a write-only register which is used to program the four functional cells (FCs) embedded inside the critical service layer. However, RSR shown in Figure 3-18 (b), contains the current state of one BioSymPLe machine and the outcome of the I/O transaction. Where R/B is defined as Ready/Busy flag that indicates whether the BSU is busy servicing I/O transaction or is ready to receive the next transaction. This is the flag used when performing I/O transaction in busy-wait mode. If R/B is equal to ‘0’, then Busy mode is active and if R/B is equal to ‘1’, then Ready mode is active. Where: - CXY0-3: specifies the local address of each FC in CSL.

CXY0	CXY1	CXY2	CXY3	Trigger0	Trigger1	Trigger2	Trigger3

Figure 3-18 (a): Writeable control register

R/B 0	R/B 1	R/B 2	R/B 3	DO0	DO1	DO2	DO3

Figure 3-18 (b): Readable status register

3.11.4 Communication Bus

Since BioSymPLe is a new self-healing hardware architecture, we will investigate using the CAN bus or the TCP/IP bus communication protocols in order to connect a network of BioSymPLe machines (BSMs). Currently, we are proposing a single communication bus (see Figure 3-19) to connect the major components of two BioSymPLe machines, combining the functions of a data bus to carry information, an address bus to determine where the resulted data should be sent, and a control bus to determine the operation of each BSM. To ensure that the communication bus is fault tolerant against the hard errors we can connect three redundant channels and each channel contains three lines: address, data, and control. Each one of these three channels needs to be connected to a voting by majority circuit that can tolerate the first fault on the bus and detect the second sequential fault. Furthermore, the Error Correcting Code (ECC) concept can be used as an additional of defense against the soft errors that can occur in the data lines.

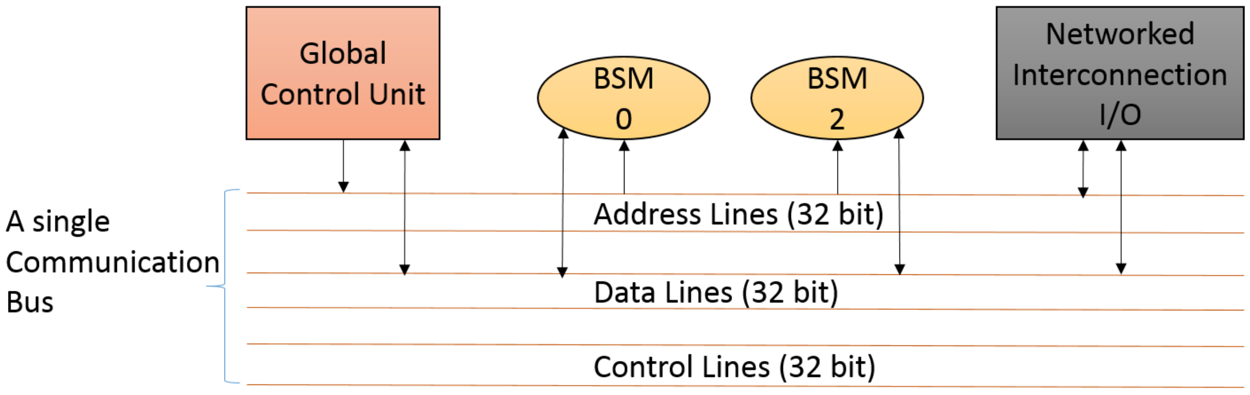


Figure 3-19: State machine based control unit

CHAPTER 4

PRELIMINARY APPLICATIONS AND SIMULATION RESULTS

4.1 Case Studies and Applications applied to BioSymPLe

BioSymPle is an evolving and active research project where refinements and new ideas are driven by mapping the architecture into different problem domains. To date, we have successfully mapped two application problems into the BioSymPle architecture: Emergency Diesel Generator (EDG) Startup for Nuclear Power back energy and an Automotive Cruise Control. These example applications are modest steps toward a planned at-scale-application related to complex distributed industrial control. In addition, two case studies are presented in the section to verify and demonstrate the correct operation of the self-healing mechanisms occurring at different layers of the proposed architecture. Quartus Prime 15.1 Lite Edition software from Altera was used as a design tool that supports several FPGA device families and the system was embedded in a digital platform the Altera Cyclone V (5CGXFC7C7F23C8) FPGA.

4.2 Fault Injection Results for the First Case Study

The first case study presents the hardware implementation and the timing diagram simulation results for the high-level perspective of two BioSymPLe architectures connected together. A block diagram for the six main components: two critical service layers, two global function migration layers, and two local function migration layers is presented in Figure 4-1(a). The two critical service layers embed eight pairs of bio-inspired cells: functional B cell and spare T cell and each global function migration layer contains four hardware components: forming health syndrome unit, syndrome switching circuit, one re-routing unit and one spare stem cell. These hardware components are connected and interfaced with each other to realize two machines of BioSymPLe architecture presented in chapter 3. Figure 4-1(b) shows that two sequential permanent faults have been injected into the 1131-based FAGFB units of B cell and T cell located in pair0 of critical service layer_0 shown in Figure 4-1(a), faults are identified by embedded self-checking units, self-healing is activated, and the system is repaired. Regarding the B cell recovery, once the trigger signal of the B cell in pair0 unit is activated at time 40ns, it will take this cell four clock cycles to produce the 32bit hexadecimal output value “51FBBBBB” with the rising edge of the clock at time 180ns. This value represents the resulted output after executing a bitwise OR function on the four input signals values:”00FAAAAA”,”01010000”, ”50500000”, and “00001111”. Whenever a permanent fault is injected into the FAGFB of a cell, it will be detected immediately by a self-checking unit embedded in the same cell and the three healing mechanisms start working. For example, at time 240ns, a permanent fault is injected into the FAGFB of the B cell0 and the self-checking unit detects that fault and activates the tristate buffer unit embedded in the same cell.

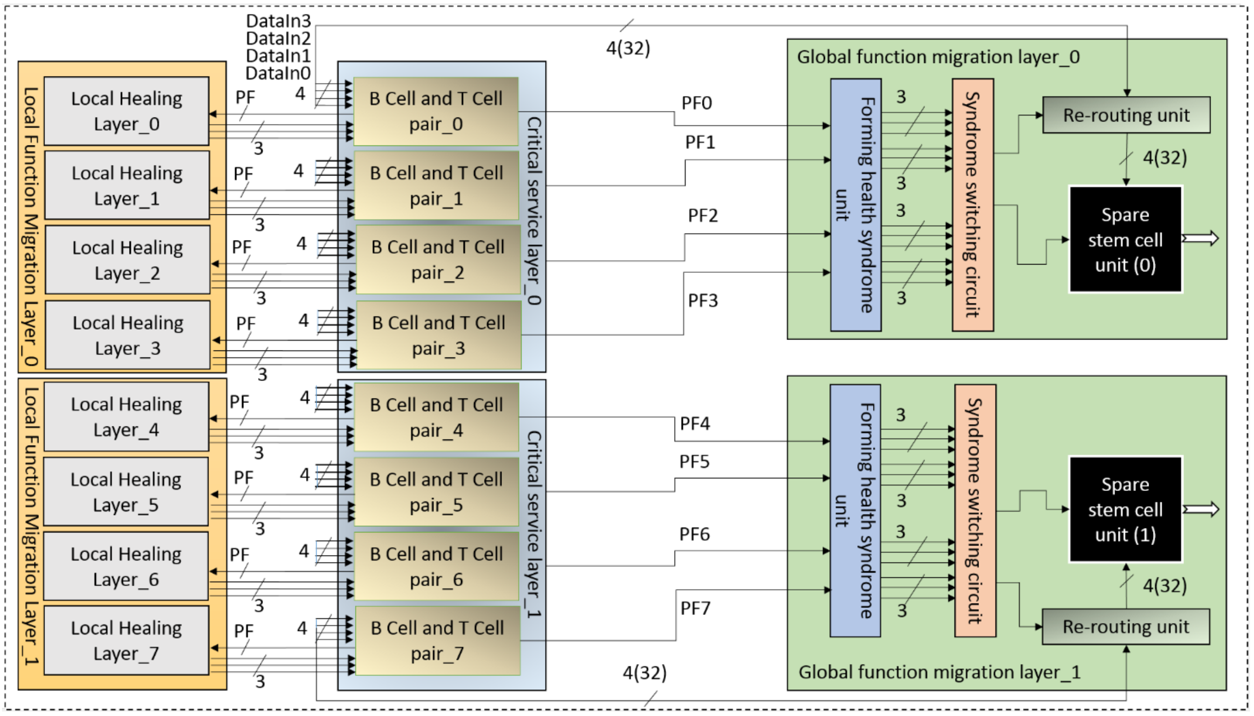


Figure 4-1 (a): Block diagram of the BioSymPLe architecture.

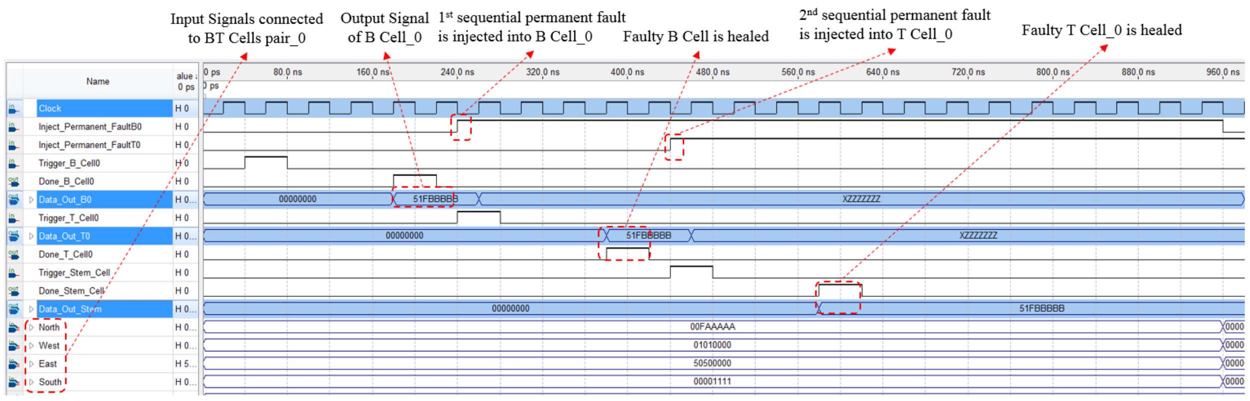


Figure 4-1 (b): Time diagram simulation results.

The buffer unit is working as a fault confinement region (FCR) for the erroneous value produced by the faulty cell and it causes the value of the “Data_Out_B0” to become high impedance with the rising edge of the next clock.

Second mechanism is transferring the four input signals of the faulty B cell to the inputs of the pre-generated T cell as a rerouting process and third mechanism is activating the same 66-bit genetic code stored in the neighboring pre-generated T cell. These are the three basic strategies that were proposed as a self-healing mechanism for the faulty B cell. However, the T cell cannot produce the correct output value until it receives its own trigger signal and generates the done signal. For instance, at the rising edge before time 400ns, the done signal of the T cell is generated and the cell produces the output value which represents the end of the healing mechanism for the faulty B cell. Additionally, regarding the T cell recovery, once the trigger signal of the T cell is activated, it produces the output value with the rising edge of the clock. Also, the self-checking unit can detect the second sequential fault being injected into the FAGFB of this cell and the healing mechanism starts working.

Ultimately, regarding the S cell recovery, there is no benefit from injecting a fault into the S cell and observing the healing mechanisms because the limitation of two machines of BioSymPLe is either to tolerate ten sequential permanent faults occurring in eight B cells and two T cell or to tolerate eight permanent faults occurring concurrently on the whole architecture presented in Figure 4-1(a) through using all backup resources: eight T cells and two embryonic stem cells.

4.3 Fault Injection Results for the Second Case Study

The second case study presents the hardware implementation and the timing diagram simulation results for the internal architecture of one pair of cells: one B cell and one T cell. Figure 4-2 (a) shows a block diagram for the basic three hardware components required to realize each BT cells pair of the BioSymPLe architecture: one biologically-inspired operational B cell (B0), one pre-generated T cell (T0), and a transfer inputs unit.

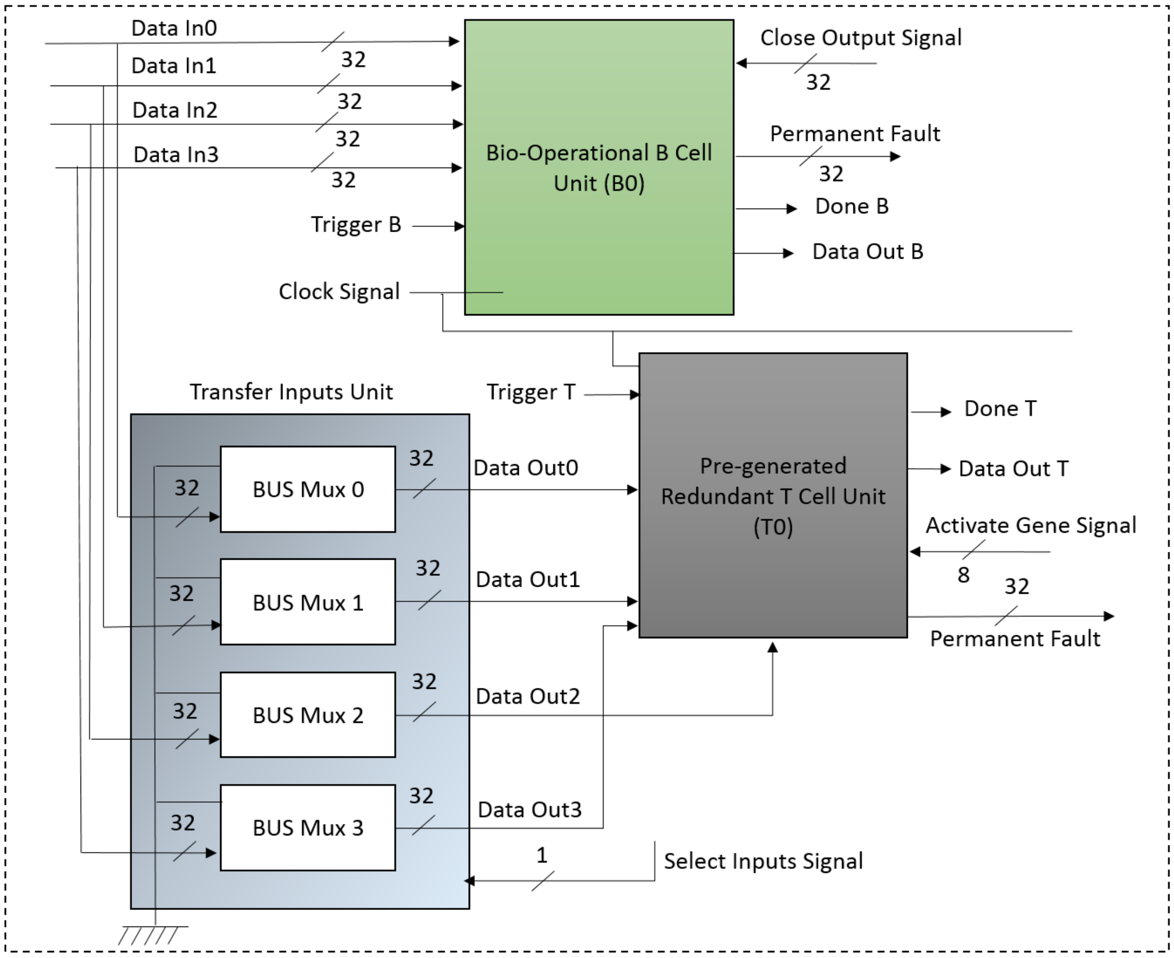


Figure 4-2 (a): Block diagram of one B cell and one T cell.

Figure 4-2 (b) shows that three sequential transient faults have been injected into the input registers of FAGFB unit for the biologically-inspired operational cell located in pair0, register redundancy units (RRUs) are tolerating their impacts quickly, and the output signal is generated without producing any erroneous value. Regarding the fault tolerance inside the biologically-inspired B cell using the RRS unit, once the trigger signal of the B cell0 is activated at time 40ns, it will take this cell four clock cycles to generate the 32bit hexadecimal output value "51FBBBBB" with the rising edge of the clock at time 180ns. This value represents the resulted output after executing a bitwise OR function on the four input signals values:"00FAAAAA","01010000", "50500000", and "00001111".

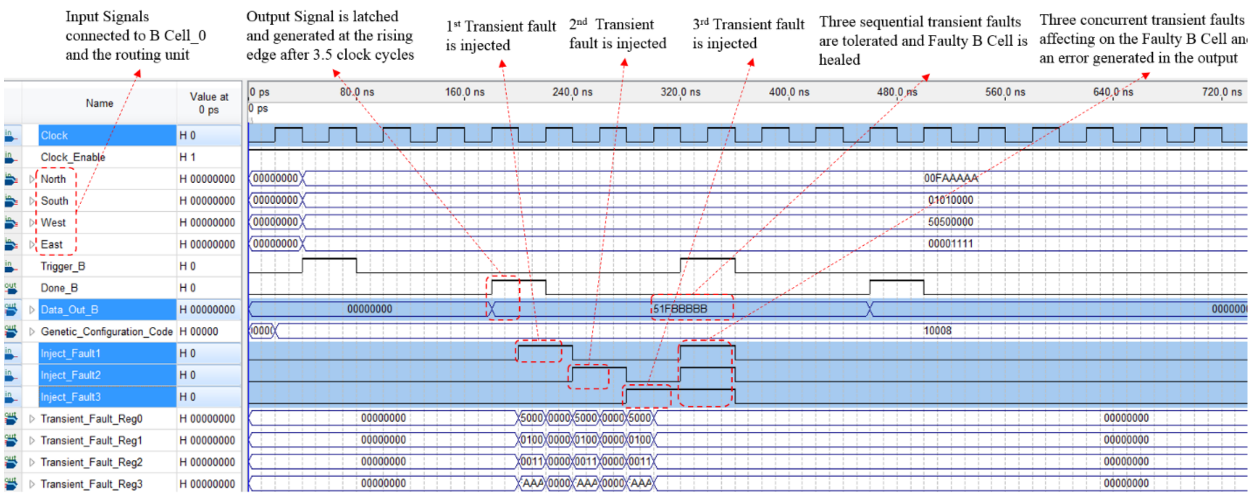

Input Signals connected to B Cell_0 and the routing unit
Output Signal is latched and generated at the rising edge after 3.5 clock cycles
1st Transient fault is injected
2nd Transient fault is injected
3rd Transient fault is injected
Three sequential transient faults are tolerated and Faulty B Cell is healed
Three concurrent transient faults affecting on the Faulty B Cell an
Name
Value at 0 ps
0 ps
80.0 ns
160.0 ns
240.0 ns
320.0 ns
400.0 ns
480.0 ns
560.0 ns
640.0 ns
720.0 ns
Clock
Clock_Enable
North
South
West
East
Trigger_B
Done_B
Data_Out_B
Genetic_Configuration_Code
Inject_Fault1
Inject_Fault2
Inject_Fault3
Transient_Fault_Reg0
Transient_Fault_Reg1
Transient_Fault_Reg2
Transient_Fault_Reg3
00FAAAAA
01010000
50500000
00001111
51FBBBBB
10008

Figure 4-2 (b): Time diagram simulation results.

Three sequential transient faults were injected into three input registers for the FAFB of the biologically-inspired B cell at times: 200ns, 240ns, and 280ns. As a consequence, the four output tag registers:" Transient_Fault_Reg0"," Transient_Fault_Reg1"," Transient_Fault_Reg2", and" Transient_Fault_Reg3", shown in Figure 4-2 (b) are generating an erroneous value only for half a clock cycle. However, the output signal "Data_Out_B" is not affected by these transient faults and continue to produce the correct value.

4.4 Emergency Diesel Generator (EDG)

The functional logic for the EDG, published in Electric Power Research Institute (EPRI) technical report is illustrated in Figure 4-3. The EDG receives a total of fourteen digital input signals and produces two output signals. The output signals are calculated from the input signals using basic combinational logic AND, OR, and NOT operations. The EDG digital control system within a Nuclear Power Plant (NPP) is a safety critical system required for reactor cooling and other safety functionalities. While the functionality of the EDG is rather simplistic, it is a highly critical system that must be fault tolerant.

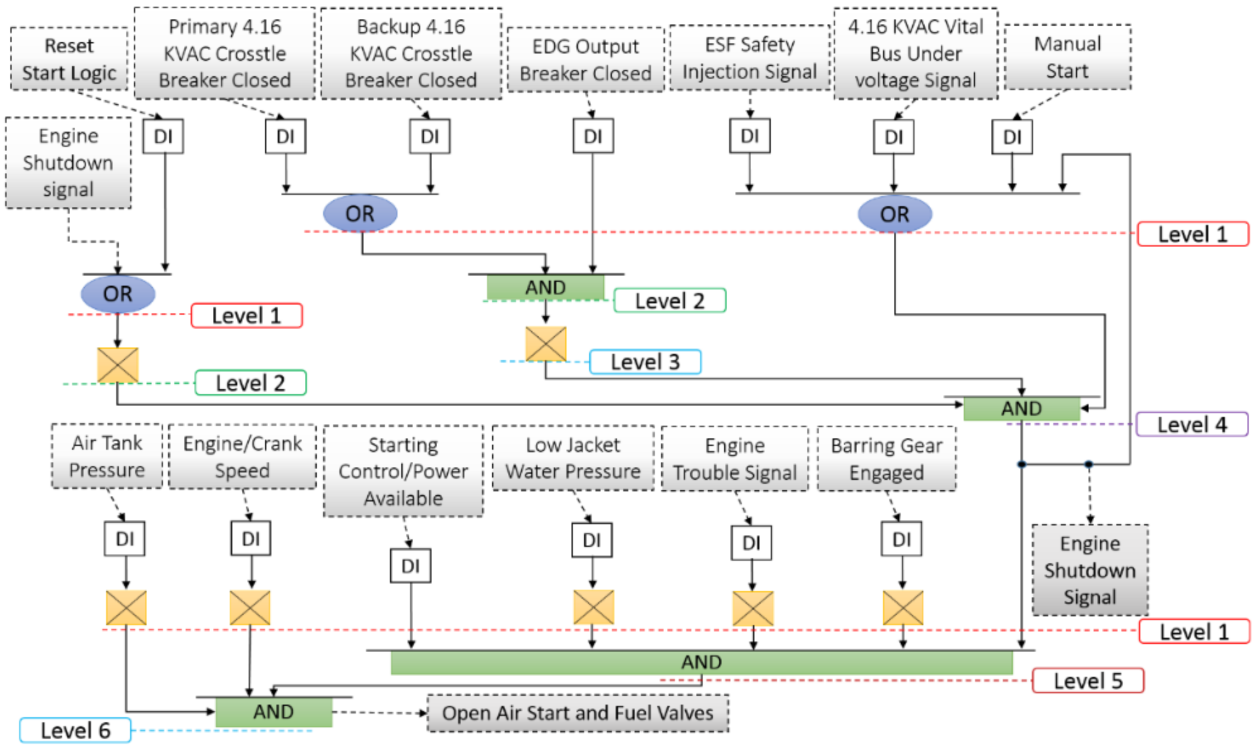

Reset Start Logic
Primary 4.16 KVAC Crosstle Breaker Closed
Backup 4.16 KVAC Crosstle Breaker Closed
EDG Output Breaker Closed
ESF Safety Injection Signal
4.16 KVAC Vital Bus Under voltage Signal
Manual Start
Engine Shutdown signal
DI
OR
AND
Level 1
Level 2
Level 3
Level 4
Level 5
Level 6
Air Tank Pressure
Engine/Crank Speed
Starting Control/Power Available
Low Jacket Water Pressure
Engine Trouble Signal
Barring Gear Engaged
Engine Shutdown Signal
Open Air Start and Fuel Valves

Figure 4-3: Logic diagram for starting the EDG.

To demonstrate the resilience properties of BioSymPLe, the EDG critical application has been implemented on the proposed architecture. This implementation required sixteen functional cells to be connected together (see Figure 4-4) in such a way that four critical service layers are interconnected with each other. Each one of the four functional cells (B cells) embedded inside the critical service layer has to 1) activate a different genetic code (DNA expression) based on the current address of the functional cell and 2) receive different digital input data through the I/O routing units connected to the input and output ports of the EDG application. The EDG functionality basically was divided into six levels of sequential execution (see Figure 4-3 and Figure 4-4) and at each level a limited number of functional cells are activated which requires 3.5 clock cycles execution time to produce its results to the next level.

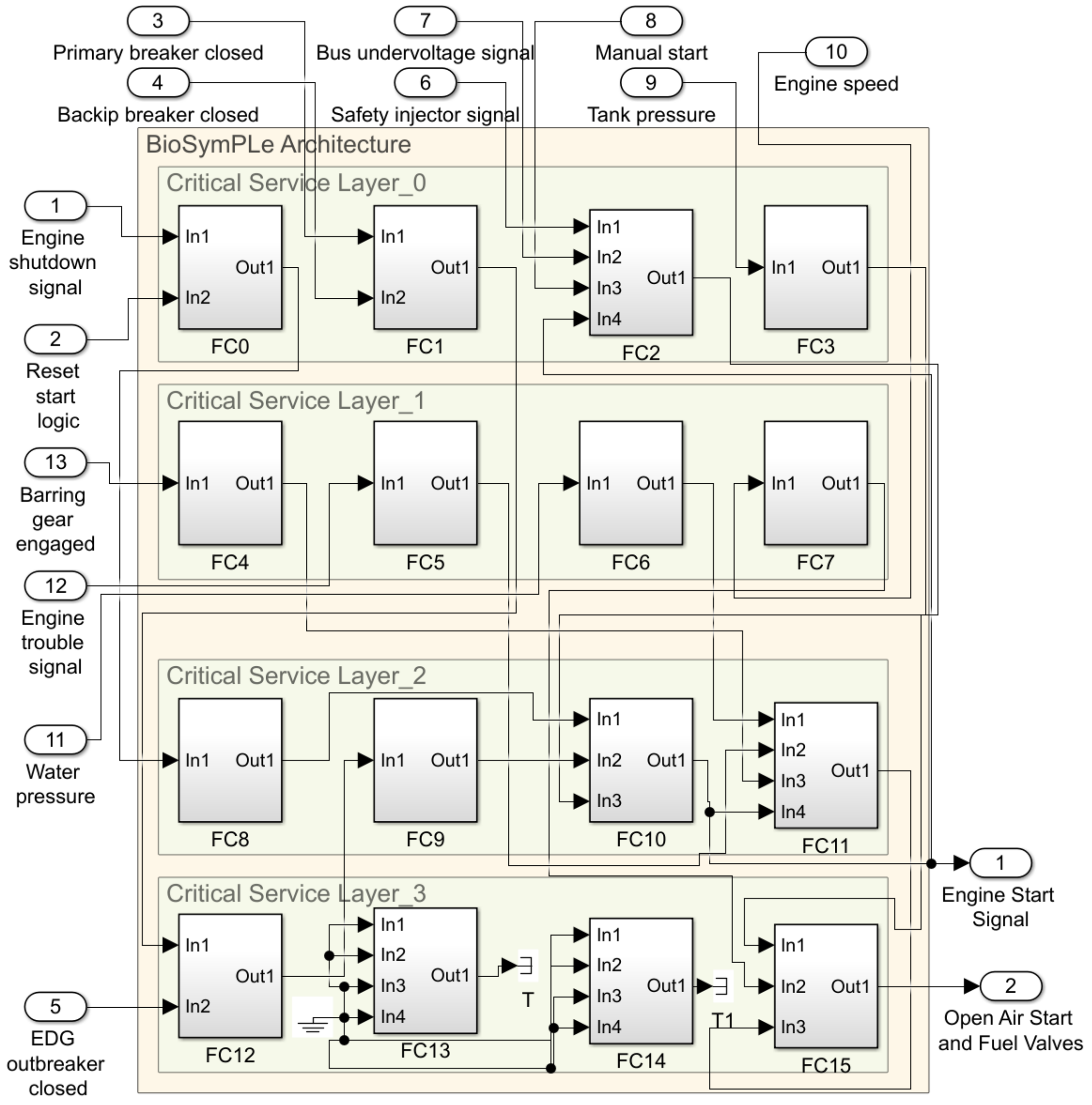


Figure 4-4: A simplified block diagram for EDG implemented on BioSymPLe.

As example, for level1, two critical service layers are required to execute the functionality of eight logic functions (OR, OR,….and NOT) by eight functional cells (FC0, FC1… and FC7). All the functional cells of this level are activated at the same instant of time T1 which is “Trigger_1” signal at time 10ns (see Figure 4-5 (a) and Figure 4-5 (b)). After that, the data will be available at the output ports of the eight cells at time 50ns with the rising edge of “Done” signal. Furthermore, for levels 2, 3, 4, 5, and 6, another two critical service layers are needed to perform the functionality of other six logic functions (NOT, NOT… AND) by the functional cells (FC8, FC9 …… and FC15). However, each one of these five levels of execution has to be activated at different instants of time (T2, T3…... and T6) as shown in Fig. 4.5 (a) and Fig. 4.5 (b). The reason is that the digital input data that is needed for each functional cell is not available at the input ports of each cell.

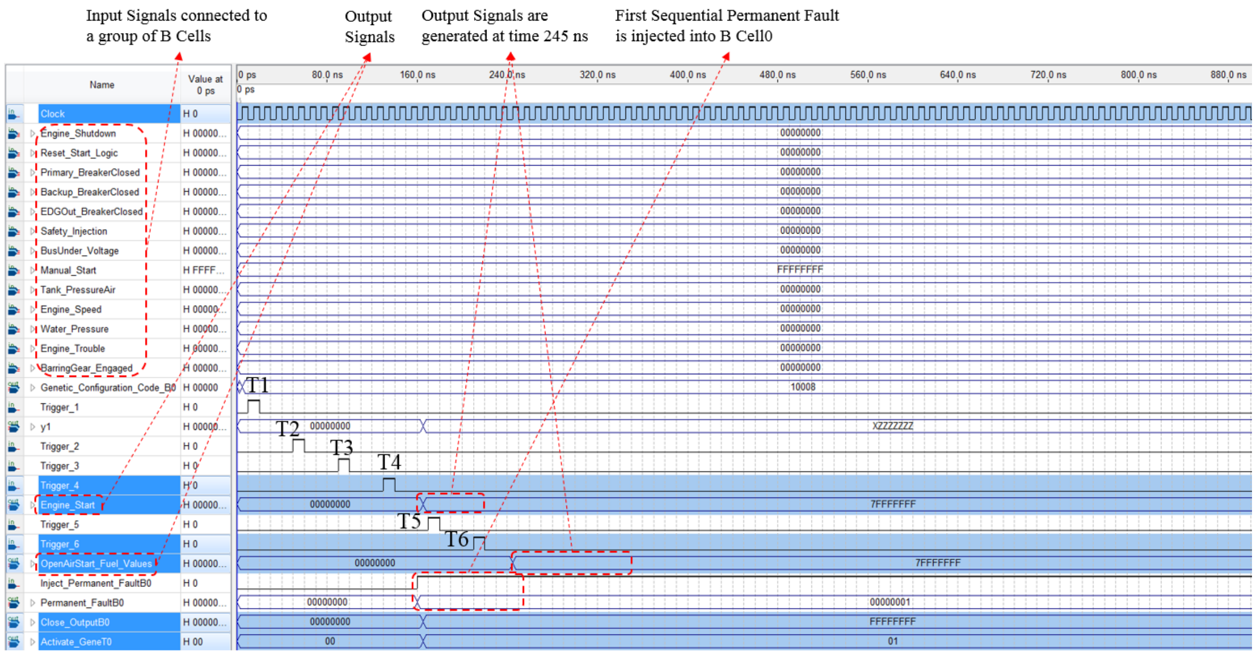


Figure 4-5 (a): Time Diagram Simulation Analysis of a first sequential fault injection in the EDG application implemented on BioSymPLe architecture by connecting four critical service layers.

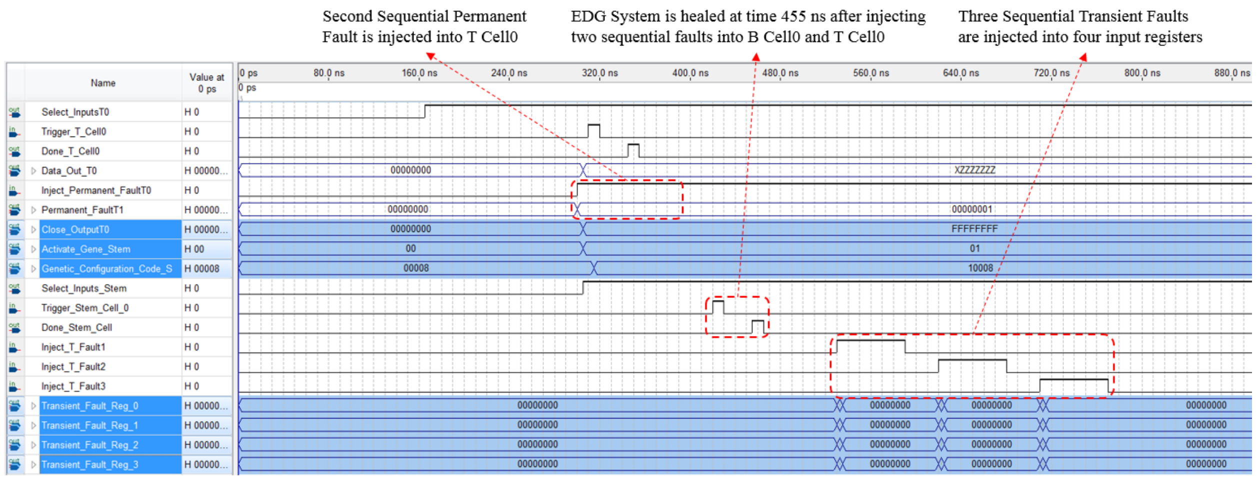


Figure 4-5 (b): Time Diagram Simulation Analysis of a second sequential fault injection in the EDG application implemented on BioSymPLe architecture by connecting four critical service layers.

When the EDG is subjected to two sequential permeant faults into the 1131 functional units of both B0 cell and T0 cell in the critical service layer, the EDG application is healed against the first fault by time 345ns and the second fault by time 455ns. This about 82 % increases in time delay to handle 2 sequential permanent faults – and this delay remains relatively constant as the number of handled faults increases.

In addition, Figure 4-6 (a), shows the simulation results for a case study in which, three sequential transient faults have been injected into the three input registers of FAGFB for the first biologically-inspired operational-cell (designated *B0* in Figure 3-5) in the architecture at times: 180ns, 240ns, and 300ns, register redundancy units are tolerating their impacts quickly, and the output signal is generated without producing any erroneous value at the “Dat_Out_B” digital output port. This signal represents the result of the GFB executing on the four digital input signals: “North”, “West”, “East”, and “South”.

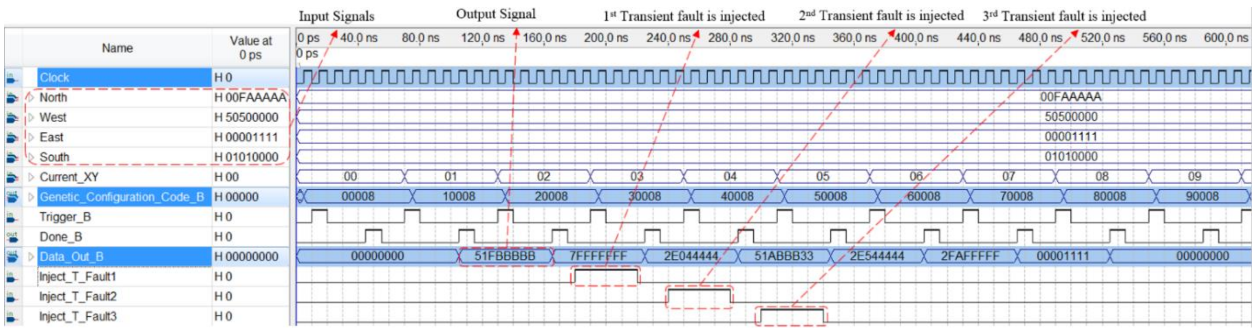


Figure 4-6 (a): Time diagram simulation of injecting three sequential transient faults in the input registers of Bio-Operational B Cell

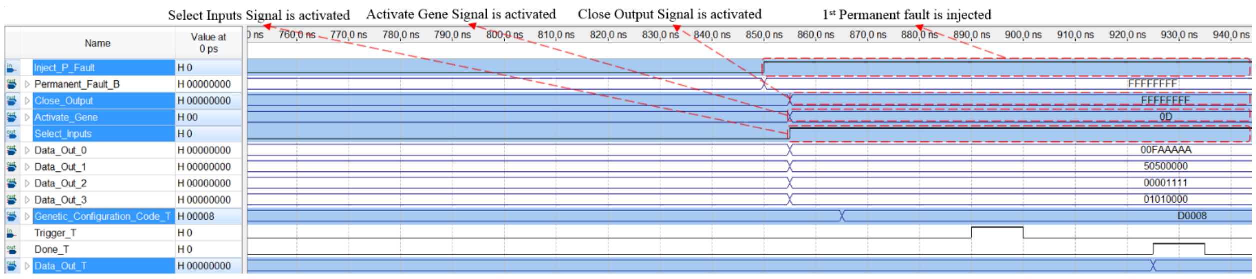


Figure 4-6 (b): Time diagram simulation of injecting one permanent fault in the 61131-Generic Functional Block.

Another case study is shown in Figure 4-6 (b), in which, whenever a permanent fault is injected into the FAGFB of the B cell (designated *B0* in Figure 3-5), it will be detected immediately by the embedded self-checking unit embedded in the same cell, and the three healing mechanisms start working in collaboration. For example, at time 850ns, a permanent fault is injected into the FAGFB of the first B cell (designated *B0* in Figure 3-5) and the self-checking unit detects that fault. At this point, the healing process against this fault starts reconfiguring the faulty B0-cell according to the three basic self-healing strategies that have been discussed previously. This reconfiguration requires the neighboring local healing layer to generate a health syndrome consisting of three control signals: “Close_OutputB0”, “Activate_GeneT0”, and “Genetic_Conficuration_Code_T0” shown in Figure 4-6 (b). These three signals activate three healing mechanisms to repair the faulty cell in such a way that “Select_InputsT0” signal is transferring the four input signals of the faulty B cell to the inputs of the pre-generated T0 cell by activating the selection lines of a network of multiplexers. “Close_OutputB0” signal’s value “FFFFFFFF” activates a network of tristate buffers to stop the faulty cell from generating any erroneous values.

“Activate_GeneT0” signal’s value “01” enables the neighboring T0 cell to activate the same genetic code which is equaled to “10008” to make the T0 cell capable of executing the same functionality. In addition, the occurrence of the second sequential fault in the T0 cell requires the same action, but involves the neighboring global function migration layer to differentiate the embryonic stem cell (S) that can be used to repair one of the four T cells (T0, T1, T2 or T3). This reconfiguration process also generates a health syndrome including the following three signals: “Close_OutputT0”, “Activate_Gene_Stem”, and “Genetic_Conficuration_Code_S”. This type of multiple fault scenario typically occurs in I&C systems when there is cascading disturbance effect due to a power fluctuations, Electromagnetic Interference (EMI), and latch-up.

The experimental results showed that the EDG application in a fault-free state requires an execution time of at least 245ns to produce the values of the two output ports: “EngineStart” and “OpenAirStartFuel_Valves”. However, when the EDG is subjected to multiple permanent and transient faults into the GFB functional units of both B cell and T cell of the first critical-service layer, the time delay is increased and the EDG application is healed by time 570ns. This is about a 230 % increase in time delay to handle 4 faults (three transient faults and one permanent fault) – and this delay remains relatively constant as the number of handled faults increases.

4.5 Cruise Control System (CCS)

A classic example that illustrates mode based control seen in process automation applications is the *automotive cruise control system (CCS)* illustrated in Figure 4-7 (a). The CCS is a closed loop control system that keeps the vehicle tracking at a constant speed without depressing the accelerator pedal in spite of the external disturbances. This can be achieved by measuring the vehicle speed, comparing it to the desired speed, and then adjusting the throttle output value based on specific control rules like the

Proportional Integral (PI) controller. The CCS receives a total of six digital input signals and produces two output signals. The output signals are calculated from the input signals using a combination of some digital control logic and a PI controller.

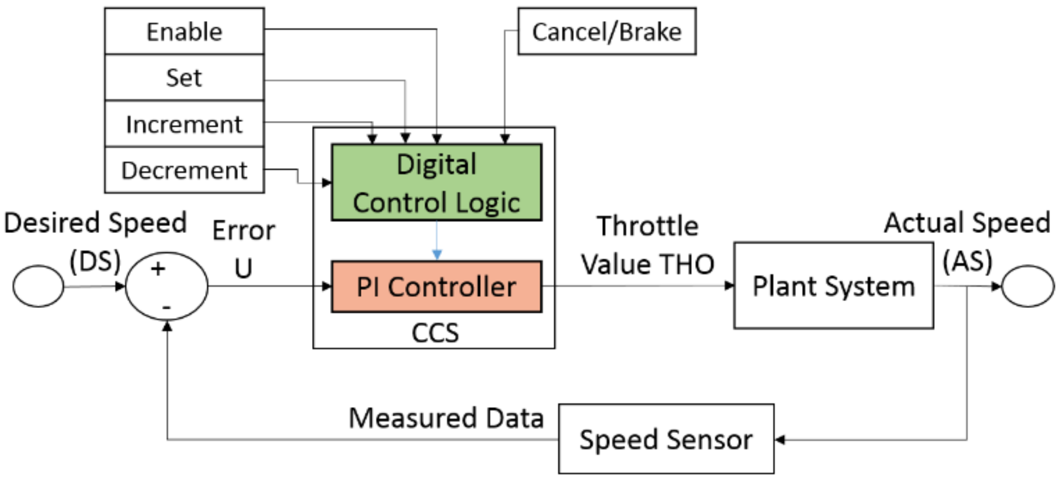


Figure 4-7 (a): A closed loop cruise control system CCS.

A block diagram for PI the controller that is used in many industrial control systems has been implemented on BioSymPLe architecture with modest investment in time. To implement the CCS application on BioSymPLe, this application has been partitioned into three levels: level1 (top control logic), level2 (PI controller), and level3 (bottom control logic) as shown in Figure 4-7 (b). Table 4-1 shows the different operations that are required to perform the mapping process of the CCS application and how they are distributed on 17 functional cells of BioSymPLe, and the four operational semantics of the designed CCS application are shown in Table 4-2. As a consequence, the functionalities of these three different levels were distributed among five critical-service layers of BioSymPLe illustrated in Figure 4-7 (b), and at each layer four functional cells (designated *B*) are triggered at a specified time. These five layers have been connected in a distributed way (see Figure 4-7 (b)) to execute two tasks: Task1 represents the top and bottom control logic and Task2 represents the PI controller in such a way that Task1 requires three critical service layers and Task2 requires only two critical service layers.

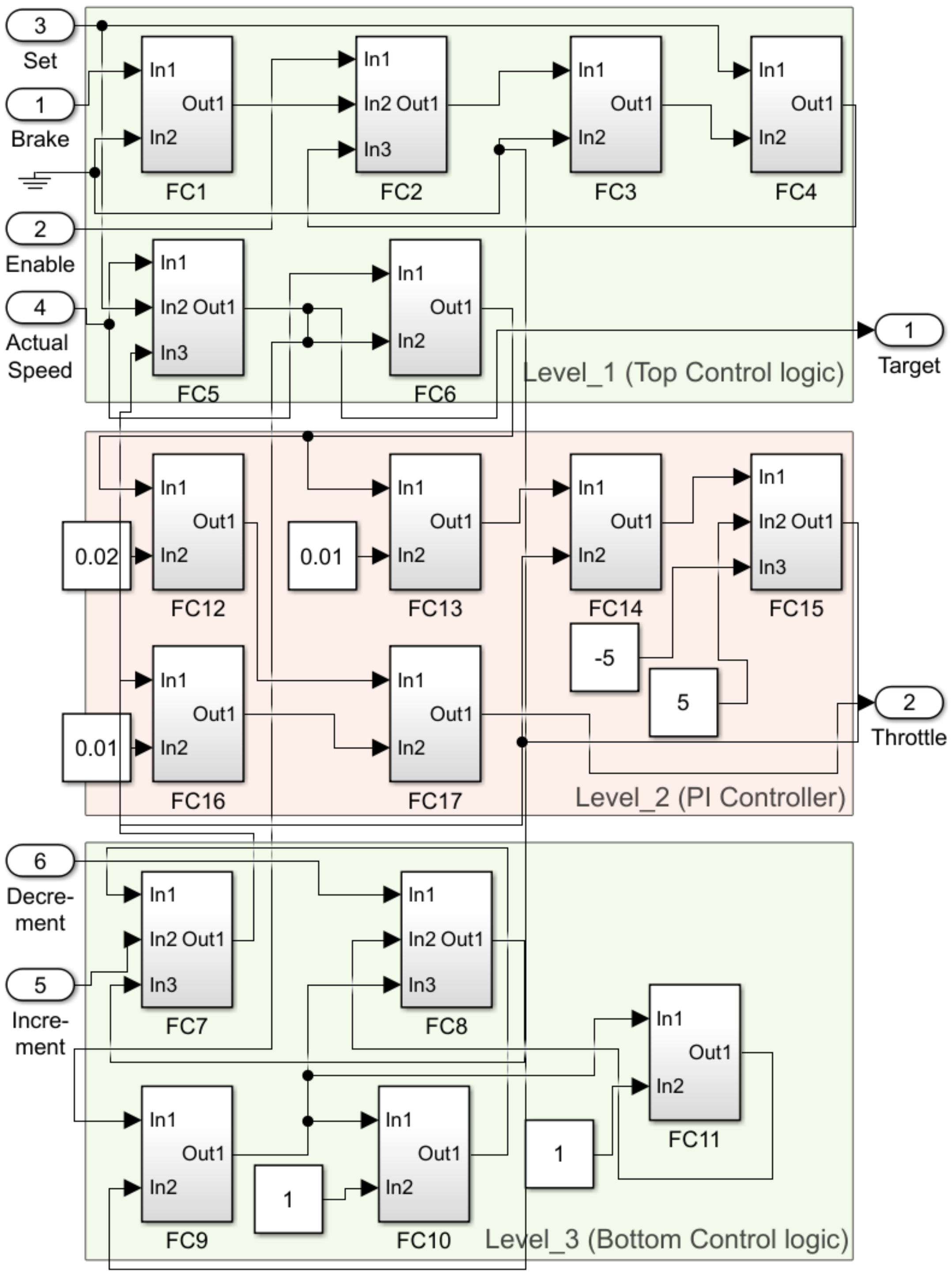


Figure 4-7 (b): A simplified block diagram for CCS implemented on BioSymPLe.

The experimental simulation results show that the CCS takes at least 35ns execution time to produce a value "50" for the "Target" output signal because the clock cycle time is 10ns. In addition, this signal is generated only by the functional cell FC5, in which the execution of control flow embedded inside the GFB needs only 3.5 clock cycles.

The simulation results show that the "Target" signal has six hexadecimal values over the 800ns execution time which are "0", "50", "49", "48", "49", and "50". The "0" value is produced by FC5 cell of BioSymPLe (see Figure 4-7 (b)) because the cell located in Level1 is not triggered to start processing the input data "actual speed". However, the same cell generates the "50" value once the cell is triggered at instance of time T1. After that, at the time 90ns, when the "Decrement" signal is enabled "-1" to decrease the plant speed, the FC11 cell, which is functioning as a subtractor circuit with constant value "1", starts working. FC8 cell, which is working as a multiplexing unit, is activated to pass the FC11 output to the "Target" port, and eight clock cycles are required as execution time to produce the decreased speed from "50" to "49" (level3 in Figure 4-7 (b)). However, at time 420ns, when the "increment" signal is activated "-1" to increase the plant speed, the FC10 cell, which is functioning as an adder circuit with constant value "1", starts working and an additional eight clock cycles are required as execution time to produce the increased speed from "48" to "49". Finally, the FC6 output, which represents the error signal based on the difference between the actual speed and target speed, is always connected as an input signal to the PI controller. This controller computes the "Throttle" output signal value based on the error value to generate the throttle output value at time 430ns.

TABLE 4-1
CCS FUNCTIONALITY IS MAPPING ON 17 FUNCTIONAL CELLS

Implementation Level	FUNCTIONAL CELL	OPERATION
Level_1	FC1	NOT
Top	FC2	Addition
Control logic	FC3	Delay
	FC4	OR
	FC5	Multiplexing
	FC6	Subtraction
Level_2	FC12, FC13	Multiplication
PI	FC14, FC17	Addition
Controller	FC15	Comparison
	FC16	Multiplication
Level_3	FC7, FC8	Multiplexing
Bottom	FC9	Delay
Control logic	FC10	Addition
	FC11	Subtraction

TABLE 4-2
OPERATIONAL SEMANTICS OF CCS APPLICATION

Condition	STATE	OPERATION
Set	Speed is set	Target speed = Actual speed
Decrement	Speed is decreased	Target speed = target speed -1
Increment	Speed is increased	Target speed = target speed +1
Cancel/Brake	Speed is cancelled	Target speed = 0

4.6 Example of Formal Verification of Properties in BioSymPLe

This section defines a notational formal approach for BioSymPLe in order to verify its correct operation in a mathematical method using Mathworks verification tools. From the outset, we adopted a model based design perspective for BioSymPLe. Model-based design is a design method that establishes a useful framework for the development and integration of formal executable models system and its environment early in the design cycle. We chose the MathWorks Simulink Toolchain for the BioSymPLe project. The primary reason for this selection was that Simulink offers an end-to-end solution in a single suite for design, simulation, implementation and verification. The main tools we used for design were:

- *Simulink and Simulink Stateflow.*
- *Simulink Design Verifier.*
- *Simulink HDL coder (automatic code generation).*
- *Altera Quartus for the analysis, simulation and synthesis of HDL designs.*

A critical tool in our verification scheme is the usage of the Simulink Design Verifier (DV) toolbox. Design Verifier is formal verification tool combining both model checking and limited automatic theorem proving. Simulink design verifier (DV) software is a plug-in to PROVER [60], which is a formal verification tool that performs reachability analysis using a combination of Bounded Model Checking (BMC) [61] and theorem induction based on the K-Induction rule (see [62] for more details). Model checking, as the name implies, given a model of a system checks to whether this model meets a given property specification. Usually this consists of exploring all states and transitions in the state model. While model checking is finite in nature, the number of states that can be efficiently searched is enormous – making it practical and applicable to real systems. If properties hold, the model checker outputs a confirmation. If a property fails to hold for some possible event sequences, the tool produces counterexamples, i.e., traces of event sequences that lead to failure of the property in question.

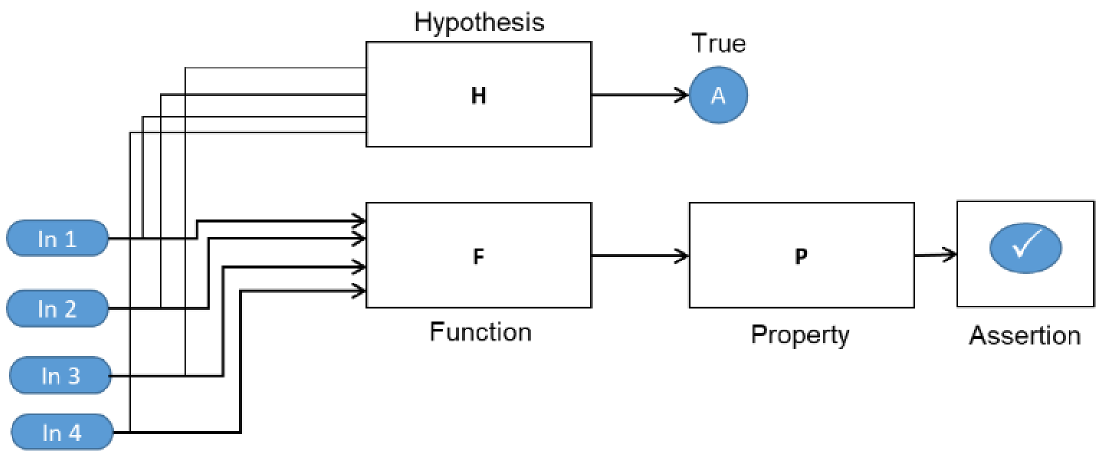


Figure 4-8 (a): Generic proof structure of Simulink DV

In Simulink DV, a proof objective is generally specified as illustrated in Figure 4-8 (a). We have a function F for which we would like to prove a certain property P. As shown in Figure 4-8 (a), the output of function F is specified as input to block P. Property P is a predicate, which should always return true when hypotheses H set on the input data flows of the model are satisfied. P is therefore connected to an *Assertion* block, while H is connected to a *Proof Assumption block*. Whenever an *Assertion* block is used, DV attempts to verify whether its specified input data flow is always true. *Proof Assumption* blocks have the purpose to constrain the input data flows of the model during proof construction. *Proof Assumptions* blocks are not always required, especially if input space does not need constrained.

Simulink DV has been used to check a number of properties for BioSymPLe and the applications. Due to space limitations, we show two examples of functional and safety properties proving. Figure 4-8 (b) shows the verification model for the FAGFB for one specific functional property proof.

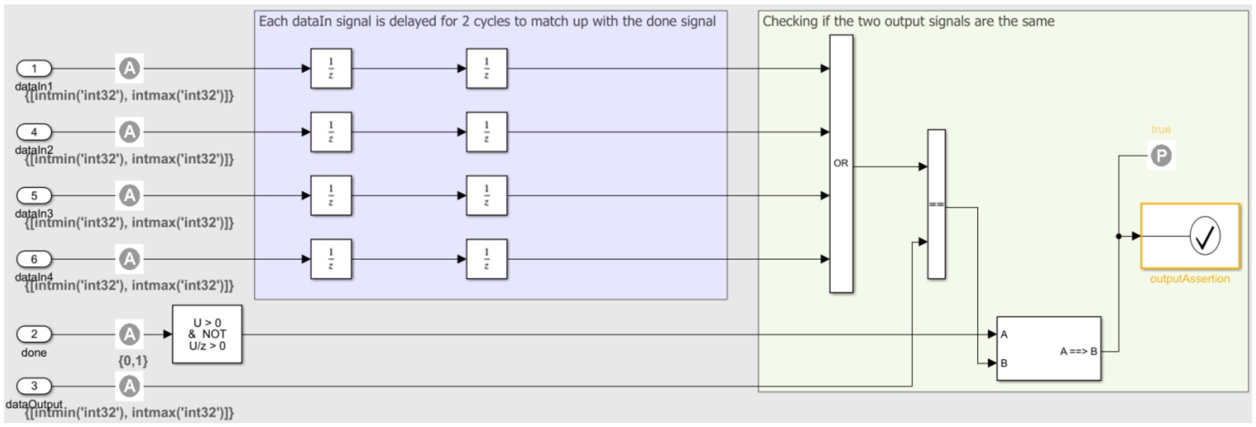

Each dataIn signal is delayed for 2 cycles to match up with the done signal
Checking if the two output signals are the same
dataIn1
{[intmin('int32'), intmax('int32')]}
dataIn2
dataIn3
dataIn4
done
{0,1}
U > 0 & NOT U/z > 0
dataOutput
OR
==
A ==> B
true
outputAssertion

Figure 4-8 (b): Functional property proving.

Firstly, a functional specification in English for proper sequencing of FAGFB is given as:

If four digital input data lines of the FAGFB are read at the same time in parallel while state machine-based control flow is triggered, the Data-out-B signal <u>will always</u> produce a correct value with the rising edge of the done signal.

This functional specification is transformed into DV property model as shown Figure 4-8 (b). The formal temporal logic expression of this requirement is

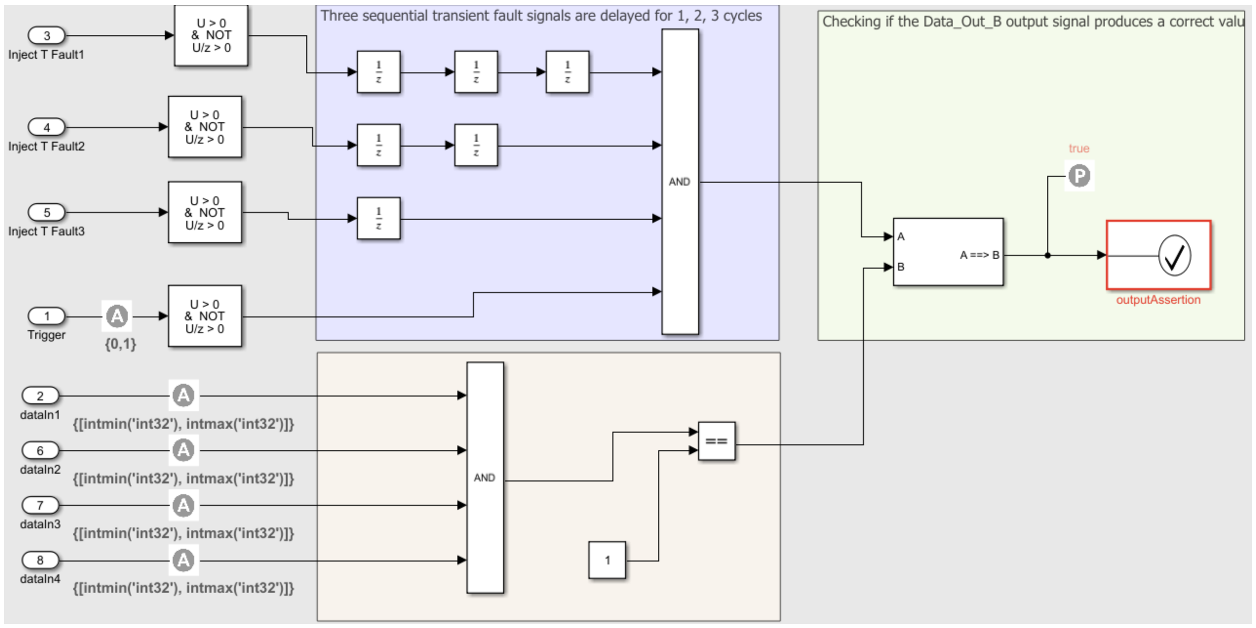
$$G\ (P1 \wedge P2 \wedge P3 \wedge P4 ==> F\ (Q))$$

Where: P1, P2, P3, P4= four digital input lines, G is universal quantification of the expression.

Secondly, a safety specification in English for proper transient faults property of FAGFB is given as:

"No matter if three sequential transient faults are injected into three input registers of 61499-based FAGFB at different times while state machine-based control flow is triggered, the Data-out-B signal will <u>never</u> produce erroneous value".

This safety specification is transformed into DV property model as shown Figure 4-9 and the formal temporal logic expression of this requirement is defined as G (P1 ^ P2 ^ P3 ==> F (Q)). Where: P1, P2, P3 = three transient faults are injected, G is universal quantification of the expression and the data output will never produce erroneous value.



Three sequential transient fault signals are delayed for 1, 2, 3 cycles
Checking if the Data_Out_B output signal produces a correct valu
Inject T Fault1
Inject T Fault2
Inject T Fault3
Trigger
{0,1}
U > 0 & NOT U/z > 0
AND
A ==> B
true
outputAssertion
dataIn1
dataIn2
dataIn3
dataIn4
{[intmin('int32'), intmax('int32')]}
==

Figure 4-9: Safety1 property proving/ transient fault assertion.

4.7 Comparative Dependability Assessment Results

In this section we perform a qualitative comparison of the BioSimPLe architecture and comparable systems found in the literature. These reference cases include a voting-by-majority Triple Modular Redundancy (TMR) self-healing architecture, the re-routing self-healing architecture by Lala et al., and the self-healing architecture inspired by the endocrine cellular communication by Yang, I et al [26], [33], [41], [23], [32].

Table 4-3 summarizes the comparison results when implementing the EDG application with an array of N*N functional cells. In the comparison table (see Table 4-3), all the functional cells that are used for either the rerouting purposes or the recovery processes were considered as an additional hardware overhead.

In addition, the maximum number of defective functional cells that can be healed against a number of sequential or concurrent permanent faults has been used in the calculation of the self-healing capacity coverage (C). Each of the four self-repairing strategies has a different cell replacement and rerouting process in such a way that each one has different advantages and disadvantages which are presented in Table 4-3. The self-healing capacity coverage (C), is calculated based on the following formula:

TABLE 4-3
SELF-HEALING CAPACITY COVERAGE AND AREA OVERHEAD COMPARISONS IN THE EDG IMPLEMENTATION WITH N*N ARRAY OF CELLS

Architecture	Biological concept	Functionality	Advantages & Disadvantages	No. of F Cells	No. of Spare Cells	No. of Re-routing Cells	Self-healing Capacity Coverage	Area Over-head
BioSymPLe Elimination Strategy	Embryonic cells+ immune system+ DNA expression	61131-based unit capable of implementing one function of up to N variables	1) Tolerating transient faults in the registers and the defected cell can be used for unlimited number of times 2) Healing against permeant faults in 1131-based unit 3) Recovery time is demonstrated	N*N / 2	(N*N) / 2 + (N*N) / 8	----	0.833	125 %
Re-routing Self-healing (Lala) Elimination Strategy	Immune system	LUT-based can implement any function of up to 3 variables	1) Healing against transient faults in the contents of RAMs 2) Cells can tolerate only one fault 3) System cannot use the defected cell for the second time 4) (tolerates only one fault for each f cell) 100% if no. of faults=no. of spares=8 5) 66% if no. of faults= 12 > no. of spares 6) Recovery time isn't verified	N*N / 2	(N*N) / 4	(N*N) / 2	0.333	150 %
Gene Control (Yang) Elimination Strategy	Endocrine cellular communication	LUT-based	1) Monitoring and detecting the soft errors only inside the gene memory 2) re-routing with, but not after cell replacement 3) proposed to be used in outer space or deep sea 4) four sequential permanent faults for one working cell and two simultaneous faults in two cells	N*N / 2	(N*N) / 2	----	0.666	100%
Voting-by-Majority TMR Elimination Strategy	Paralogous gene regulatory circuits	LUT-based	1) five permanent faults and unlimited number of transient faults in a single working cell with time delay reconfiguring four spare cells and one redundant cell	N*N / 2	(N*N) / 2 + (N*N) / 8	----	0.833	125%

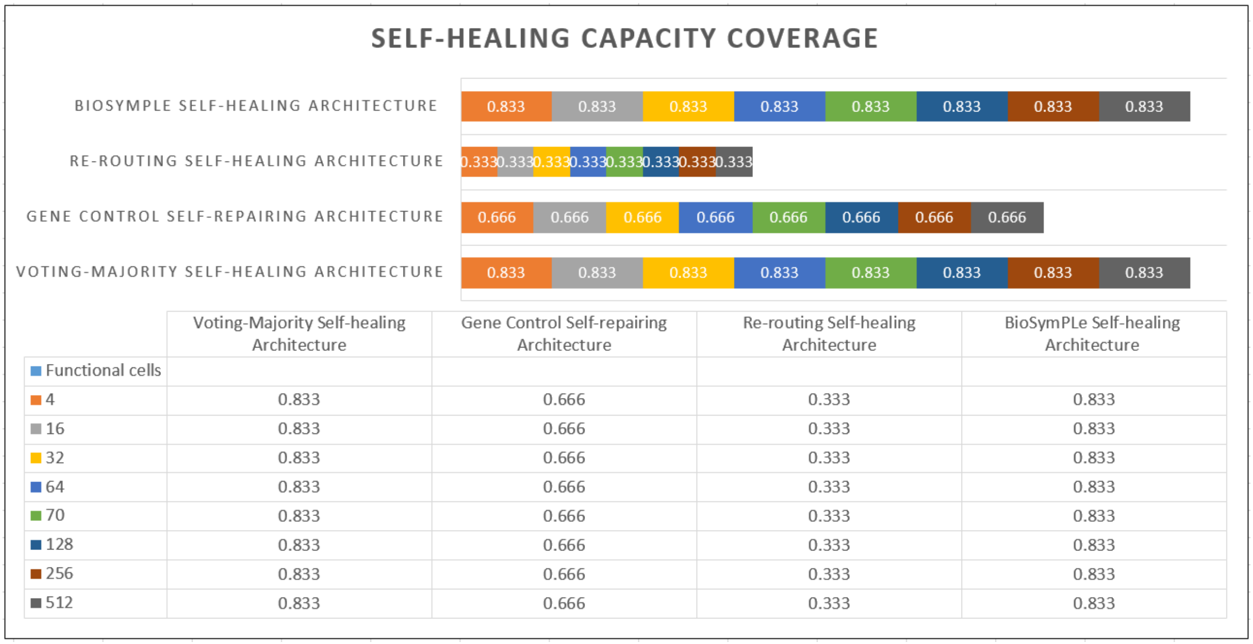
$$Self healing\ capacity\ coverage\ (C) = \frac{SCs}{SPF} \qquad (4\text{-}5)$$

Where:-

SC_S : represents the total or available number of spare functional cells that can be used for self-healing at an instant in given time.

SPF : represents the maximum number of sequential or concurrent permanent faults occurrences that may defect the self-healing architecture at a given time.

In relation to the self-healing capacity coverage, we assume up to a maximum of 12 fault occurrences of fault type permanent (SPF=12) that can impact the self-healing architecture sequentially at different times or concurrently at one instant of time. The self-healing capacity coverage (C) has been calculated for BioSymPLe and compared to the other three architectures as it is shown in Figure 4-10 (a).



	Voting-Majority Self-healing Architecture	Gene Control Self-repairing Architecture	Re-routing Self-healing Architecture	BioSymPLe Self-healing Architecture
🟦 Functional cells				
🟧 4	0.833	0.666	0.333	0.833
⬜ 16	0.833	0.666	0.333	0.833
🟨 32	0.833	0.666	0.333	0.833
🟦 64	0.833	0.666	0.333	0.833
🟩 70	0.833	0.666	0.333	0.833
🟦 128	0.833	0.666	0.333	0.833
🟫 256	0.833	0.666	0.333	0.833
⬛ 512	0.833	0.666	0.333	0.833

Figure 4-10 (a): Self-healing capacity coverage between different architectures as system size increases

These results show that BioSymPLe architecture has the potential to achieve high self-healing capacity coverage (C) (approaching 1), and as much as the coverage of both self-repairing architectures the voting-by-majority and the gene control architecture by Yang as the system size increases. In addition, our proposed architecture requires 125% hardware overhead and this overhead is approximately equal to the overhead was consumed by the voting-by-majority elimination architecture. The gene control self-repairing architecture requires 100% hardware area overhead and the re-routing self-healing architecture consumes 150 %, as shown in Figure 4-10 (b). The hardware area overhead that is shown in Table 4-3 was calculated based on the equation (4-6) *is considered low* for our architecture when compared to the re-routing self-healing architecture approach and the self-healing capacity coverage is considered high.

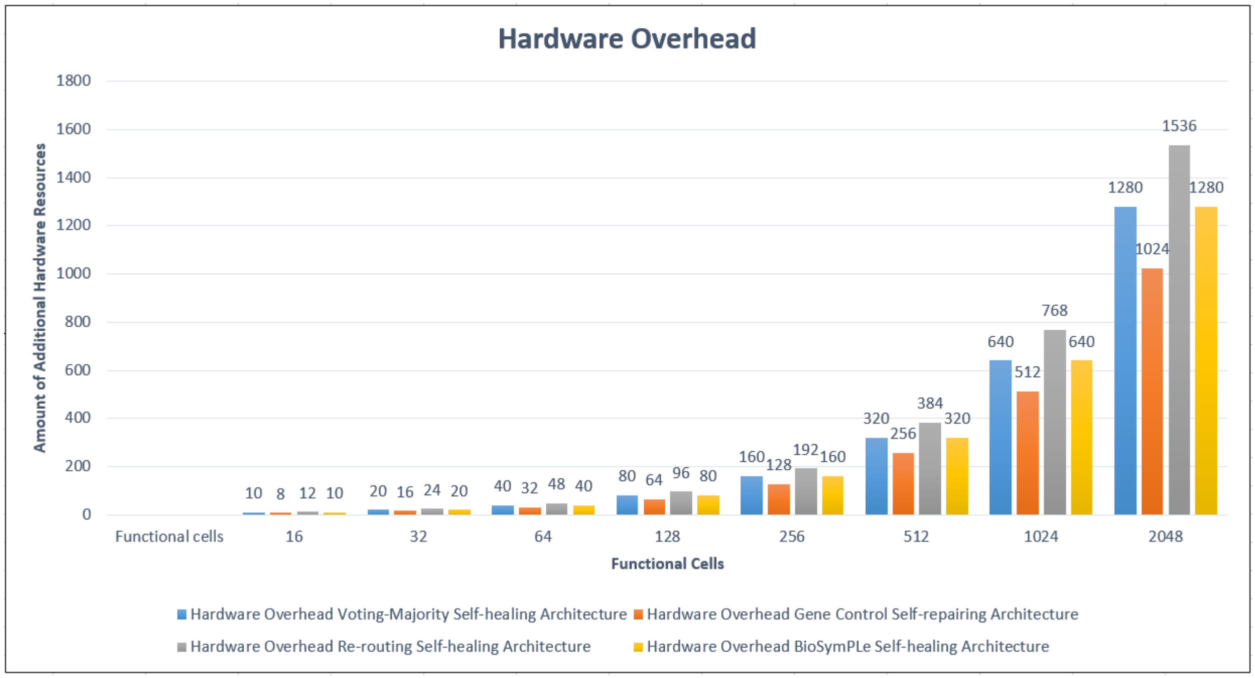


Figure 4-10 (b): Hardware Area overhead between different architectures as system size increases

The reason behind that pre generated T cells are distributed throughout the system structure in such a way that each functional B cell has its own T cell. As a result, no row or column elimination strategy is needed to recover the system against the failure, which is considered an inefficient method in terms of hardware area resources in other systems. When a cell goes faulty, only one of its surrounding redundant T cells can replace it in a self-healing system. However, as a drawback, the unutilized hardware resources embedded in the same cell when the faulty cell is removed from the system due to only a single fault occurrence in one component is considered inefficient.

Regarding the re-routing architecture, self-healing properties can be achieved for the same number of faulty cells defected by permanent faults as a comparison with the three presented architectures. However, the self-healing capacity coverage is less and equal to 0.333 (see Table 4-3) due to the availability of only four spare cells in a network comprised of sixteen functional cells. The results presented in Table 4-3 are dependent on fault classes that they were assumed and BioSymPLe was designed for- some other self-healing hardware architectures work only for tolerating transient faults, soft errors, intermittent faults, etc. Consequently, a comprehensive comparison table between the proposed self-healing architectures found in the literature is challenging. However, we can that the BioSymPLe architecture is superior to other research works in terms of predicting multiple fault classes, self-healing coverage, and area overhead.

$$Area\ overhead\ = \frac{(No.of\ spare\ cells + No.of\ routing\ cells)}{No.of\ functional\ cells} * \ 100 \qquad (4\text{-}6)$$

In addition, BioSymPLe has been compared to the previous work that has presented a fully verified FPGA overlay architecture called SymPLe as it is presented in [63], and our architecture achieves some advantages in terms of performance and reliability as shown in Table 4-4. In [63], the researchers have developed a FPGA overlay architecture called SymPLe whose purpose is (1) provide PLC (Programmable Logic Controller) like functionality with constraints on execution behavior, and (2) maximize verifiability to address systematic faults. SymPLe uses function block organized in independent "task lanes" that are sequenced by a global controller to emulate the IEC 1131 programmer model.

TABLE 4-4
COMPARISON BETWEEN BIOSYMPLE AND SYMPLE ARCHITECTURES

BioSymPLe	SymPLe
GFB is built based on N parallel input registers	FB is built based on 4 sequential input registers
1 clock cycle is required to read four digital input data	4 clock cycles are required to read four digital input data
The execution time for one GFB is 4 clock cycles	The execution time for one FB is 7 clock cycles based on reading in the additional input registers
Resilient against different hardware based transient and permanent faults	Includes traditional fault detection methods for fail safe design
Uses different biological concepts: B cells and T cells in immune system, cell life cycle	No biological concepts
Changing a hardware architecture by activating a list of genetic codes stored inside distributed configuration memories	Sequencing a list of instructions (IRs) stored in a program memory (OP-code, source address, destination address)
VHDL Hardware Description Language-based design implemented on FPGA using Quartus Design Tool	Mathworks Simulink Software-based design targeting FPGA

4.7.1 Reliability and Safety Modeling Analysis of BioSymPLe

Traditionally, the reliability and safety analysis of safety critical system is accomplished with combinatorial mathematics methods. These include Probabilistic Risk Analysis, Fault tree methods, reliability block diagrams, and hazard tree analysis. These standard methods of reliability and safety analysis are based mathematics that are stateless and time invariant [64]. Unfortunately, most combinatoric approaches are incapable expressing dynamic or regenerative behavior in systems where renewal and expiration processes are a factor. Regenerative behaviors include repair, reconfiguration, maintenance, upgrades. Expiration processes typically include timeouts, real-time deadlines, etc… Furthermore, these dynamic processes are often sequence dependent, which implies global state information. When encountering systems with these characteristics, it is necessary to model such systems by using the more powerful Markov modeling technique [65]. A Markov process is a stochastic process whose behavior depends only upon the current state of the system. The particular sequence of steps by which the system entered the current state is irrelevant to its future behavior.

4.7.1.1 Markov Process basics

Consider a parallel structure of two components. Each component is assumed to have two states, a functioning state and a failed state. The structure has therefore $2^2 = 4$ possible states, and the state space is

X = {0, 1, 2, 3}.

S0 = both up

S1 = A failed, B up

S2 = B failed, A up

S3 = Both failed

Let X(t) denote the state of the system at time t**.** The state space is the set of all the possible system states. In most texts, number the states by integers from 0 to r. The state space is therefore

$$X = \{0, 1, 2, \ldots, r\}$$

$Let\ Pi(t) = Pr(X(t) = i)$ be the probability that the system is in state i at time t.

The state probability distribution is denoted

$$P(t) = (P0(t), P1(t), \ldots, Pr(t))$$

Markov property and Processes

Assume that the process is in state i at time s, that is, X(s) = i.

The conditional probability that the process will be in state j at time t + s is

$$Pr(X(t+s) = j \mid X(s) = i, X(u) = x(u), 0 \leq u < s)$$

where {x(u), 0 ≤ u < s} denotes the “history” of the process up to, but not including, time s

The process is said to have the **Markov property** if

$$Pr(X(t+s) = j \mid X(t) = i, X(u) = x(u), 0 \leq u < s)$$

$$= Pr(X(t+s) = j \mid X(s) = i)\ for\ all\ possible\ x(u), 0 \leq u < s$$

In other words, when the present state of the process is known, the future development of the process is independent of anything that has happened in the past.

A ‘continuous time’ stochastic process that fulfills the Markov property is called a **Markov process.**

We will further assume that the Markov process for all i, j in X fulfills

$$Pr(X(t+s) = j \mid X(s) = i) = Pr(X(t) = j \mid X(0) = i)$$

$$for\ all\ s, t \geq 0$$

which says that the probability of a transition from state i to state j **does not depend** on the global time and only depends on the time interval available for the transition.

The transition probabilities of the Markov process

$$Pij(t) = Pr(X(t) = j \mid X(0) = i)$$

may be arranged as a matrix

$$\wp(t) = \begin{matrix} P(t)_{00} & P(t)_{01} \cdots & P(t)_{0r} \\ P(t)_{10} & P(t)_{11} \cdots & P(t)_{1r} \\ P(t)_{r0} & P(t)_{r1} \cdots & P(t)_{rr} \end{matrix}$$

When a process is in state i at time 0, it must either be in state i at time t or have made a transition to a different state. We must therefore have

$$\sum_{j=0}^{r} 1\, P_{i,j} = 1$$

A Markov-based chain model is defined as a group of known states and state transitions of a system and defines the probability of the system to transition between operational and non-operational states. Furthermore, certain states in the model represent the stable behavior of digital system such as fully operational state, while others represent the unstable behavior like fail-operational (fault occurs but the system continues to work) state, fail-safe (failure that does not cause harm) state, or a fail-unsafe (failure that causes harm or death) state.

To define the systems failure, we need to carefully consider all the case scenarios of the failure modes that can lead to the death state. System failure is often a complex function of environmental conditions, external events, software bugs, and hardware failures.

For any given system, a Markov model consists of a list of the possible states of that system, the possible transition paths between those states, and the rate parameters of those transitions. In reliability analysis the transitions usually consist of failures and repairs. When representing a Markov model graphically (as is in Figure 4-11), each state is usually depicted as a "bubble", with arrows denoting the transition paths between states.

In Figure 4-11, the symbol λ denotes the *rate parameter* of the transition from State 3 to State 2. In addition, we denote by P_j (t) the probability of the system being in State j at time t. If the device is known to be healthy at some initial time t = 0, the initial probabilities of the two states are P_0 (3) = 1 and P_1 (F) = 0. Thereafter the probability of State 3 decreases at the constant rate λ (assuming constant failure process), which means that *if* the system is in State 3 at any given time, the probability of making the transition to State 2 during the next increment of time Δt (dt) is λdt. Therefore, the overall probability that the transition from State 3 to State 2 will occur during a specific incremental interval of time dt is given by multiplying (1) the probability of being in State 3 at the beginning of that interval, and (2) the probability of the transition during an interval dt given that it was in State 3 at the beginning of that increment.

To calculate the reliability and safety of a reconfiguration by graceful degradation digital circuit as it is shown in Figure 3-3 and in Figure 3-4, three basic states Markov model has been constructed (see Figure 4-11).

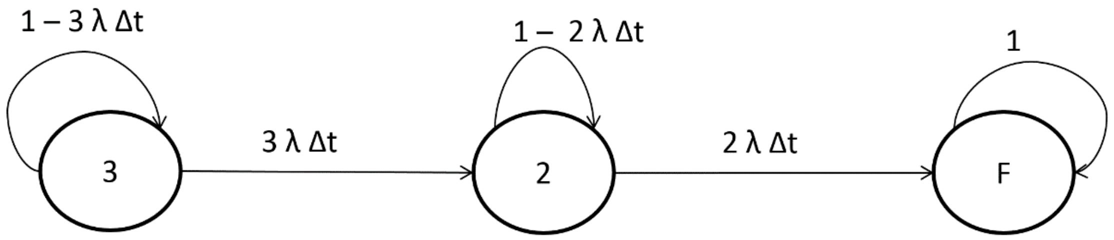


Figure 4-11: Transition state diagram for a reconfiguration by graceful degradation

The Markov model presented in Figure 4-11 includes one fully-operational state, one failed-safe state, and one failed-unsafe state. This model contains the states for all cases that we have assumed for the system without repair rates and perfect fault detection coverage. State 3, which is defined as a fully-operational state; represent the case where all the hardware components of the BioSymPLe architecture are operating correctly. State 2, which is called failed-safe state, represent the system in which one of the three bio-inspired functional cells has failed permanently and that failure was successfully detected by the duplication with comparison unit embedded in the same functional cell. State F, which is defined as failed-unsafe (death) state that can only be reached through the failure of two or three functional cells embedded in one GC unit due to the sequence of two or three permanent faults as it can be seen in Figure 3-4.

To illustrate the method and the assumptions that have been used to develop a reliability model for the reconfiguration by graceful degradation circuit, the equations for the Markov model of the designed system can be written in matrix form as

$$P_{system\ (t+\Delta t)} = T_{system} * P_{system\ (t)}$$

Where:-

T_{system} is the state transition matrix

$P_{system\ (t)}$ is the probability of being in the corresponding state at the time t.

$P_{system\ (t+\Delta t)}$ is the probability of being in the corresponding state at the time t + Δt.

<u>Assumptions:-</u>

All the states are either in a fully operational state, a fail-safe operational state, or the fail-unsafe state. The states and the cases are illustrated in Table 4-5:

TABLE 4-5
STATE NUMBER AND STATUS FOR THE MARKOV MODELED SYSTEM BASED ON FIG 4.11

State Number (3-F)	State Status
3	System Fully Operational (all the H/W components are healthy)
2	Fail-Safe Operational (Only one bio-inspired functional cell fails with detection)
F	Fail-Unsafe (death) (Two or three bio-inspired functional cells permanently fail without detection)

4.7.2 Markov Mathematical Analysis

The equations of the markov model of the designed system can be written from the state diagram shown in Figure 4-11. For example, the probability of the system being in a state 1 at time t + Δ t depends on the probability that the system was in a state from which it could transition to state 1 and the probability of being in that state. The probability equations of the Markov model and the resulted 2D state transition matrix that includes all the failure rates and fault coverages for the system are shown below. At initial state: Y = V1 (FC-A) = V2 (FC-B) = V3 (FC-C), If the functional cell FC-A fails then FC-B = FC-C= Y (see Figure 3-12), which masks the FC-A permanent failure.

For the reliability Markov model, we assume that each functional cell

1- Obeys the exponential failure law.
2- Has a constant failure rate of λ.

The probability that a system will be failed at some time (t + Δt)

P (t + Δt) = 1 - $e^{-\lambda \Delta t}$ = $\lambda \Delta t$

P3 (t + Δt) = (1 - 3λ Δt) P3 (t)

P2 (t + Δt) = (3 λ Δt) P3 (t) + (1 – 2 λ Δt) P2 (t)

PF (t + Δt) = (2 λ Δt) P2 (t) + PF (t)

P (t + Δt) = A P (t) ➔ P (Δt) = A P (0)

The reliability of the TMR system: R (t) = 1- PF (t) = P3 (t) + P2 (t)

The equations of the discrete time Markov model for the Triple Modular Redundant (TMR) system can be written as:-

$$\frac{\text{P3 (t + Δt)} - \text{P3 (t)}}{\Delta t} = -3\,\lambda\,\text{P3 (t)}$$

$$\frac{\text{P2 (t + Δt)} - \text{P2 (t)}}{\Delta t} = 3\,\lambda\,\text{P3 (t)} - 2\,\lambda\,\text{P2 (t)}$$

$$\frac{\text{PF (t + Δt)} - \text{PF (t)}}{\Delta t} = 2\,\lambda\,\text{P2 (t)}$$

By allowing the time interval Δt to approach zero through (algebraic manipulation) results in a set of differential equations:-

$$\frac{d\,\text{P3 (t)}}{dt} = -3\,\lambda\,\text{P3 (t)}$$

$$\frac{d\,\text{P2 (t)}}{dt} = 3\,\lambda\,\text{P3 (t)} - 2\,\lambda\,\text{P2 (t)}$$

$$\frac{d\,\text{PF (t)}}{dt} = 2\,\lambda\,\text{P2 (t)}$$

By using the Laplace Transform, we have

S P3 (S) – P3 (0) = -3 λ P3 (S)

S P2 (S) – P2 (0) = 3 λ P3 (S) - 3 2 λ P2 (S)

S P3 (S) – PF (0) = 2 λ P2 (S)

We assume that the system starts in the perfect state and time t=0 so

P3 (0) = 1, P2 (0) = 0, PF (0) = 0

Consequently, the Laplace Transform equations can be written as:-

$$P3\ (S) = \frac{1}{S + 3\,\lambda}$$

$$P2\ (S) = \frac{3\,\lambda}{(S +\ 2\,\lambda)\ (S +\ 3\,\lambda)}$$

$$PFS\ (S) = \frac{6\,\lambda^2}{S\ (S +\ 2\,\lambda)\ (S +\ 3\,\lambda)}$$

These equations can be written as:-

$$P3\ (S) = \frac{1}{S + 3\,\lambda}$$

$$P2\ (S) = \frac{3}{(S + 3\,\lambda)} + \frac{-3}{(S + 3\,\lambda)}$$

$$PFS\ (S) = \frac{1}{S} + \frac{-3}{(S + 2\,\lambda)} + \frac{2}{(S + 3\,\lambda)}$$

Taking the inverse Laplace Transform results in the solution given by:-

$$P3(t) = e^{-3\,\lambda\,t}$$

$$P2(t) = 3\,e^{-2\,\lambda\,t} -\ 3\,e^{-3\,\lambda\,t}$$

$$PFS(t) = 1 -\ 3\,e^{-2\,\lambda\,t} +\ 2\,e^{-3\,\lambda\,t}$$

So, the Reliability of the TMR system is the probability of being in either State 3 or State 2.

R (t) = P3 (t) + P2 (t) = $e^{-3\,\lambda\,t} + 3\,e^{-2\,\lambda\,t} -\ 3\,e^{-3\,\lambda\,t}\ =\ 3\,e^{-2\,\lambda\,t} -\ 2\,e^{-3\,\lambda\,t}$

This Markov model has been used to model the TMR system that

1- Does not depend on the fault coverage (C).

2- Does not depend on the repair process (μ).

However, developing a Markov model that includes these two cases will be more complicated as it will be presented in the following section.

4.7.3 Reliability Modeling Analysis of One Lane of BioSymPLe

To calculate the reliability and safety of one lane of the BioSymPLe architecture that includes multi-levels of fault-tolerance and monitoring elements and has a property of reconfiguration by graceful degradation as it is shown in Figure 3-12 and in Figure 4-11, eight basic states Markov model has been constructed (see Figure 4-12).

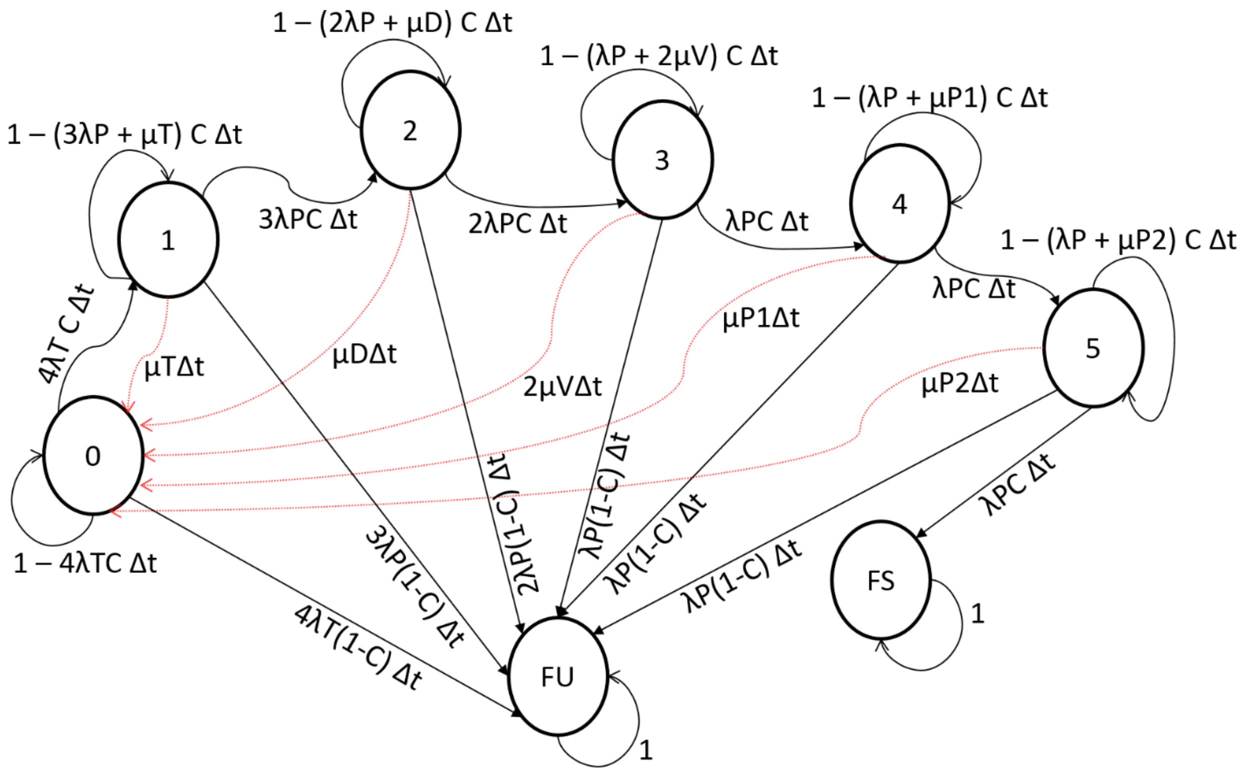


Figure 4-12: Transition state diagram for one lane of BioSymPLe

The Markov model presented in Figure 4-12 includes six operational states, one failed-safe state, and one failed-unsafe state. This model contains the states for all cases that we have assumed for the system with repair rates and perfect fault detection coverage.

State 0, which is called fully-operational state; represent the case where all the hardware components of one lane of the BioSymPLe architecture are operating correctly. State 1, which is called fail-operational state, represent the system in which any one of the four input registers embedded inside the three bio-inspired functional cells has defected by a transient fault and this transient failure can be successfully detected by a monitoring switch. State 2, which is called fail-operational state, represent the system in which the first 61131-based functional block is defected by a permanent fault and this failure is detected by a duplication with comparison circuit and masked by a voting by majority circuit. State 3, which is also called fail-operational state, represent the system in which the second 61131-based functional block is defected by a permanent fault and this permanent failure is masked by a voting by majority circuit. State 4, which is called fail-operational state, represent the system in which the third 61131-based functional block is defected by a permanent fault and this failure is detected by a duplication with comparison circuit and repaired by a Local Function Migration Layer (LFML) and the T-functional cell. State 5, which is called fail-operational state, represent the system in which the 61131-based functional block embedded inside the T functional cell is defected by a permanent fault and this failure is healed by the Global Function Migration Layer (GFML) and the Stem-functional cell. State 5 can only be reached through the failure of four bio-inspired functional cells (FCs): three B functional cells and one T functional cell due to the sequence of four sequential permanent faults.

To illustrate the method and the assumptions that have been used to develop a reliability model for one lane of BioSymPle, the equations for the Markov model of the designed system can be written in matrix form as

$$P_{system\,(t+\Delta t)} = T_{system} * P_{system\,(t)}$$

Where:-

T_{system} is the state transition matrix

$P_{system\ (t)}$ is the probability of being in the corresponding state at the time t.

$P_{system\ (t+\Delta t)}$ is the probability of being in the corresponding state at the time t + Δt.

<u>Assumptions:-</u>

All the states are either in fully operational state, fail-safe operational state, or the fail-unsafe state. The states and the cases are illustrated in Table 4-6:

TABLE 4-6
STATE NUMBER AND STATUS FOR THE MARKOV MODELED SYSTEM BASED ON FIG 4.12

State Number (0-FU)	State Status
0	One Lane of BioSymPLe Fully Operational (all the H/W components are healthy)
1	Fail-Operational at Register Level (Transient faults tolerance for unlimited number of times)
2	Fail-Operational at Functional Block Level (Permanent faults masking and detection for 1st and 2nd sequential faults)
3	Fail-Operational at Cellular Level (Permanent faults detection and repairing for 3rd sequential fault)
4	Fail-Operational at Critical Service Level (Permanent faults detection and healing for 4th sequential fault)
5	Fail-Operational at Critical Service Level (Permanent faults detection and healing for 4th sequential fault)
FS	Fail-Safe Operational (Only one bio-inspired functional cell fails with detection)
FU	Fail-Unsafe (death) (Two or three bio-inspired functional cells permanently fail without detection)

A list of the assigned symbols and coefficients used in the Markov Model is shown below in Table 4-7:

TABLE 4-7
FAILURE RATES FOR DIFFERENT COMPONENTS FOR THE SYSTEM MODELED USING THE MARKOV MODEL OF FIG 4.12

$\lambda_1 = \lambda_-$ Stratix IV FPGA chip = 37.5 FIT = $0.02666667 \times 10^{-9}$
MTTR-Cell = 55 ns.
µ-Cell = Cell Repair Rate = 1/MTTR = 0.018 repair/ns.
MTTR-Register = 5 ns.
µ-Register = Register Repair Rate = 1/MTTR = 0.2 repair/ns.
C = C-Assumption Coverage = 0.80 $\rightarrow$ 1.00 (0.99999)

4.7.4 Markov Mathematical Analysis

The equations of the markov model of the designed system can be written from the state diagram shown in Figure 4-12. For example, the probability of the system being in a state 1 at time t + Δ t depends on the probability that the system was in a state from which it could transition to state 1 and the probability of being in that state. The probability equations of the Markov model and the resulted 2D state transition matrix that includes all the failure rates and fault coverages for the system are explained below. At initial state: Y = V1 (FC-A) = V2 (FC-B) = V3 (FC-C), If the functional cell FC-A fails then FC-B = FC-C= Y (see Figure 3-12), which masks the FC-A permanent failure.

For the reliability Markov model, we assume that each functional cell

3- Obeys the exponential failure law.

4- Has a constant failure rate of λ.

The probability that a system will be failed at some time (t + Δt)

P (t + Δt) = 1 - $e^{-\lambda\,\Delta t}$ = $\lambda\,\Delta t$

P0 (t + Δt) = (1 - 4$\lambda T\ C\ \Delta t$) P0 (t) + μT Δt P1 (t) + μD Δt P2 (t) + 2μV Δt P3 (t) + μP1 Δt P4 (t) + μP1 Δt P5 (t)

P1 (t + Δt) = [1- (3 $\lambda P +$ μT) C Δt] P1 (t)

P2 (t + Δt) = [1- (2 $\lambda P +$ μD) C Δt] P2 (t)

P3 (t + Δt) = [1- ($\lambda P +$ 2μV) C Δt] P3 (t)

P4 (t + Δt) = [1- ($\lambda P +$ μP) C Δt] P4 (t)

P5 (t + Δt) = [1- ($\lambda P +$ μP2) C Δt] P5 (t)

PFS (t + Δt) = λP C Δt P5 (t) + PFS (t)

PFU (t + Δt) = 4$\lambda T\ (1 - C)\ \Delta t$) P0 (t) + 3$\lambda P\ (1 - C)\ \Delta t$ P1 (t) + 2$\lambda P\ (1 - C)\ \Delta t$ P2 (t) + $\lambda P\ (1 - C)\ \Delta t$ P3 (t) + $\lambda P\ (1 - C)\ \Delta t$ P4 (t) + $\lambda P\ (1 - C)\ \Delta t$ P5 (t) + PFU (t)

The equations of the discrete time Markov model for the one lane of BioSymPLe system can be written as:-

P (t + Δt) = T-system P-system (t) ➔ P (Δt) = T-system P-system (0)

The reliability of the one lane of BioSymPLe: R (t) = 1- PFU (t) = PFS (t) + P5 (t) + P4 (t) + P3 (t) + P2 (t) + P1 (t) + P0 (t)

Where:-

$$\frac{\text{P0 (t + Δt)} - \text{P0 (t)}}{\Delta t} = -\,4\lambda TC\ \text{P0 (t)} + \mu\text{T P1 (t)} + \mu\text{D P2 (t)} + 2\mu\text{V P3 (t)} + \mu\text{P1 P4 (t)} + \mu\text{P2 P5 (t)}$$

$$\frac{\text{P1 (t + Δt)} - \text{P1 (t)}}{\Delta t} = -\,(3\,\lambda P + \mu\text{T})\ \text{C P1 (t)}$$

$$\frac{\text{P2 (t + Δt)} - \text{P2 (t)}}{\Delta t} = -\,(2\,\lambda P + \mu\text{D})\ \text{C P2 (t)}$$

$$\frac{\text{P3 (t + Δt)} - \text{P3 (t)}}{\Delta t} = -\,(\lambda P + 2\mu\text{V})\ \text{C P3 (t)}$$

$$\frac{\text{P4 (t + Δt)} - \text{P4 (t)}}{\Delta t} = -\,(\lambda P + \mu\text{P1})\ \text{C P4 (t)}$$

$$\frac{\text{P5 (t + Δt)} - \text{P5 (t)}}{\Delta t} = -\,(\lambda P + \mu\text{P2})\ \text{C P5 (t)}$$

$$\frac{\text{PFS (t + Δt)} - \text{PFS (t)}}{\Delta t} = \lambda P\ \text{C P5 (t)}$$

$$\frac{\mathrm{PFU}\,(\mathrm{t} + \Delta\mathrm{t}) - \mathrm{PFU}\,(\mathrm{t})}{\Delta t} = 4\lambda T\,(1-C)\,\mathrm{P0}\,(\mathrm{t}) + 3\lambda P\,(1-C)\,\mathrm{P1}\,(\mathrm{t}) + 2\lambda P\,(1-C)\,\mathrm{P2}\,(\mathrm{t}) + \lambda P\,(1-C)\,\mathrm{P3}\,(\mathrm{t}) + \lambda P\,(1-C)\,\mathrm{P4}\,(\mathrm{t}) + \lambda P\,(1-C)\,\mathrm{P5}\,(\mathrm{t})$$

$$\text{P-system}\,(\mathrm{t}+\Delta\mathrm{t}) = \begin{vmatrix} \mathrm{P0}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{P1}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{P2}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{P3}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{P4}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{P5}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{PFS}\,(\mathrm{t} + \Delta\mathrm{t}) \\ \mathrm{PFU}\,(\mathrm{t} + \Delta\mathrm{t}) \end{vmatrix}$$

The two dimensional state transition matrix of a Markov model for one Lane of BioSymPLe is:

$$\text{P-system}\,(\mathrm{t}+\Delta\mathrm{t}) = \begin{vmatrix} -4\lambda TC & \mu\mathrm{T} & \mu\mathrm{D} & 2\mu\mathrm{V} & \mu\mathrm{P1} & \mu\mathrm{P2} & 0 & 0 \\ 0 & -(3\,\lambda P + \mu\mathrm{T})\,\mathrm{C} & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -(2\,\lambda P + \mu\mathrm{D})\,\mathrm{C} & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -(\lambda P + 2\mu\mathrm{V})\,\mathrm{C} & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & -(\lambda P + \mu\mathrm{P1})\,\mathrm{C} & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -(\lambda P + \mu\mathrm{P2}) & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & \lambda P\,\mathrm{C} & 0 & 0 \\ 4\lambda T\,(1-C) & 3\lambda P\,(1-C) & 2\lambda P\,(1-C) & \lambda P\,(1-C) & \lambda P\,(1-C) & \lambda P\,(1-C) & 0 & 0 \end{vmatrix}$$

$$\text{P-system}\,(\mathrm{t}) = \begin{vmatrix} \mathrm{P0}\,(\mathrm{t}) \\ \mathrm{P1}\,(\mathrm{t}) \\ \mathrm{P2}\,(\mathrm{t}) \\ \mathrm{P3}\,(\mathrm{t}) \\ \mathrm{P4}\,(\mathrm{t}) \\ \mathrm{P5}\,(\mathrm{t}) \\ \mathrm{PFS}\,(\mathrm{t}) \\ \mathrm{PFU}\,(\mathrm{t}) \end{vmatrix}$$

By allowing the time interval Δt to approach zero through (algebraic manipulation) results in a set of differential equations:-

$$\frac{d\,\mathrm{P0}\,(t)}{dt} = -4\lambda TC\,\mathrm{P0}\,(t) + \mu T\ \mathrm{P1}\,(t) + \mu D\ \mathrm{P2}\,(t) + 2\mu V\ \mathrm{P3}\,(t) + \mu P1\ \mathrm{P4}\,(t) + \mu P2\ \mathrm{P5}\,(t)$$

$$\frac{d\,\mathrm{P1}\,(t)}{dt} = -(3\,\lambda P + \mu T)\,C\,\mathrm{P1}\,(t)$$

$$\frac{d\,\mathrm{P2}\,(t)}{dt} = -(2\,\lambda P + \mu D)\,C\,\mathrm{P2}\,(t)$$

$$\frac{d\,\mathrm{P3}\,(t)}{dt} = -(\lambda P + 2\mu V)\,C\,\mathrm{P3}\,(t)$$

$$\frac{d\,\mathrm{P4}\,(t)}{dt} = -(\lambda P + \mu P1)\,C\,\mathrm{P4}\,(t)$$

$$\frac{d\,\mathrm{P5}\,(t)}{dt} = -(\lambda P + \mu P2)\,C\,\mathrm{P5}\,(t)$$

$$\frac{d\,\mathrm{PFS}\,(t)}{dt} = \lambda P\,C\ \mathrm{P5}\,(t)$$

$$\frac{d\,\mathrm{PFU}\,(t)}{dt} = 4\lambda T\,(1-C)\,\mathrm{P0}\,(t) + 3\lambda P\,(1-C)\,\mathrm{P1}\,(t) + 2\lambda P\,(1-C)\,\mathrm{P2}\,(t) + \lambda P\,(1-C)\,\mathrm{P3}\,(t) + \lambda P\,(1-C)\,\mathrm{P4}\,(t) + \lambda P\,(1-C)\,\mathrm{P5}\,(t)$$

By using the Laplace Transform, we have

$$S\,\mathrm{P0}\,(S) - \mathrm{P0}\,(0) = -4\lambda TC\,\mathrm{P0}\,(S) + \mu T\ \mathrm{P1}\,(S) + \mu D\ \mathrm{P2}\,(S) + 2\mu V\ \mathrm{P3}\,(S) + \mu P1\ \mathrm{P4}\,(S) + \mu P2\ \mathrm{P5}\,(S)$$

$$S\,\mathrm{P1}\,(S) - \mathrm{P1}\,(0) = -(3\,\lambda P + \mu T)\,C\,\mathrm{P1}\,(S)$$

$$S\,\mathrm{P2}\,(S) - \mathrm{P2}\,(0) = -(2\,\lambda P + \mu D)\,C\,\mathrm{P2}\,(S)$$

$$S\,\mathrm{P3}\,(S) - \mathrm{P3}\,(0) = -(\lambda P + 2\mu V)\,C\,\mathrm{P3}\,(S)$$

$$S\,\mathrm{P4}\,(S) - \mathrm{P4}\,(0) = -(\lambda P + \mu P1)\,C\,\mathrm{P4}\,(S)$$

$$S\,\mathrm{P5}\,(S) - \mathrm{P5}\,(0) = -(\lambda P + \mu P2)\,C\,\mathrm{P5}\,(S)$$

$$S\,\mathrm{PFS}\,(S) - \mathrm{PFS}\,(0) = \lambda P\,C\ \mathrm{P5}\,(S)$$

$$S\,\mathrm{PFU}(S) - \mathrm{PFU}\,(0) = 4\lambda T\,(1-C)\,\mathrm{P0}\,(S) + 3\lambda P\,(1-C)\,\mathrm{P1}\,(S) + 2\lambda P\,(1-C)\,\mathrm{P2}\,(S) + \lambda P\,(1-C)\,\mathrm{P3}\,(S) + \lambda P\,(1-C)\,\mathrm{P4}\,(S) + \lambda P\,(1-C)\,\mathrm{P5}\,(S)$$

In the analysis, we assume that the system starts in the perfect state and time t=0 so :-

P0 (0) = 1,   P1 (0) = 0,   P2 (0) = 0 , P3 (0) = 1,   P4 (0) = 0,   P5 (0) = 0, PFS (0) = 0,   PFU (0) = 0.

Consequently, the Laplace Transform equations can be written as:-

So, the Reliability of one lane of BioSymPLe is the probability of being in either State 0 or State 1 or State 2 or State 3 or State 4 or State 5.

R (t) = P0 (t) + P1 (t) + P2(t) + P3 (t) + P4 (t) + P5 (t) + PFS (t)

This Markov model has been used to model a lane of BioSymPLe that

1- Does depend on the fault coverage (C).
2- Does depend on the repair process (μ).

4.7.5 Simulation Results

This section presents the reliability simulation results of modeling one lane of the proposed architecture BioSymPLe using the WinSTEM reliability analysis program. The WinSTEM software is an analysis tool that was developed by NASA to calculate the probabilities of failure within a specific time period. In order to calculate the probability that a one lane of BioSymPLe architecture will be failed after a specific period of time, different failure rates and repair rates were assigned to several hardware components in the proposed digital system as it can be seen in Table 4-8. The failure and repair processes were assumed to be exponentially distributed with rates λ and μ and the same failure rate λ were used in all case studies due to that we have used the same FPGA chip as a hardware platform. In addition, the failure and repair rates values for the transient and permanent failure modes assumptions were changed to be different values as shown in figures below to see the effects of their values on the unreliability of the system. The Stratix IV FPGA chip has a failure rate of 37.5 FITs and the mean time between failures (MTBF) is of 37.5 thousands to billion hours. As a result, the value of the failure rate λ is equal to 1/MTBF = 1/ 37.5 * 1000 hours that lead to λ = 0.02666667 * 1/1000 failure/hour.

TABLE 4-8
FAILURE AND REPAIR RATES OF THE HARDWARE COMPONENTS IN BIOSYMPLE

Hardware Component	**Failure Rate**	**Repair Rate**
Functional Cell	0.02666667 * 10E-3	0.018 * 10E-9
Register Redundancy Scheme	0.02666667 * 10E-3	0.2 * 10E-9
Duplication with Comparison	0.02666667 * 10E-3	0.018 * 10E-9
Voting by Majority Circuit	0.02666667 * 10E-3	0.018 * 10E-9
Local Function Migration Layer	0.02666667 * 10E-3	0.018 * 10E-9
Global Function Migration Layer	0.02666667 * 10E-3	0.018 * 10E-9

In one case study, the fault coverage (C), which can be defined as the ability of the digital system to detect and recover from faults, has been changed to two different values (C1= 0.80 and C1= 0.90) and the effects of these values on the unreliability which represents the probability of failure in the unsafe state is presented in Figure 4-13 below. The unreliability with a higher coverage value is less than the unreliability with a small coverage.

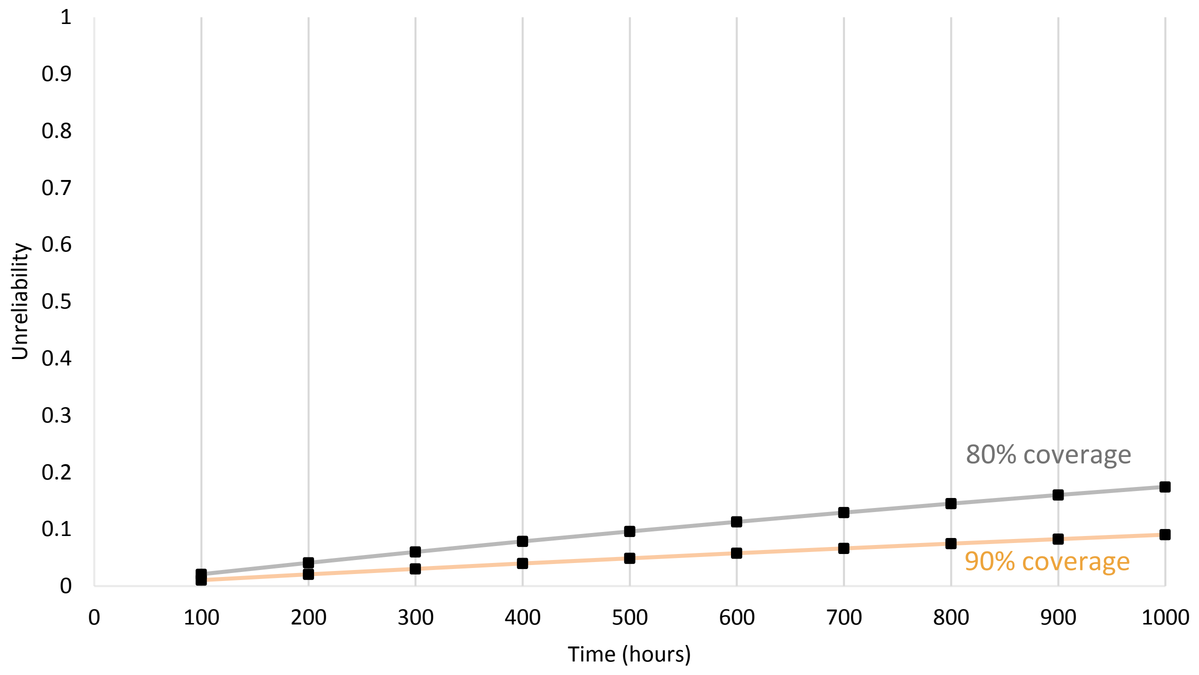


Figure 4-13: A smaller coverage causes higher unreliability

Figure 4-14 presents the reliability R (Time), which was calculated based on the total summation of probabilities for all states of Markov model except the unsafe (death) state with the time per two coverage values (C1= 0.80, C2= 0.90). The results show that the reliability of each one of the two case studies is decreased as time goes with a constant failure rate (λ) Lambda. In addition, the second case with a coverage 0.9 over time is achieving higher levels of reliability than the first case with a coverage 0.8.

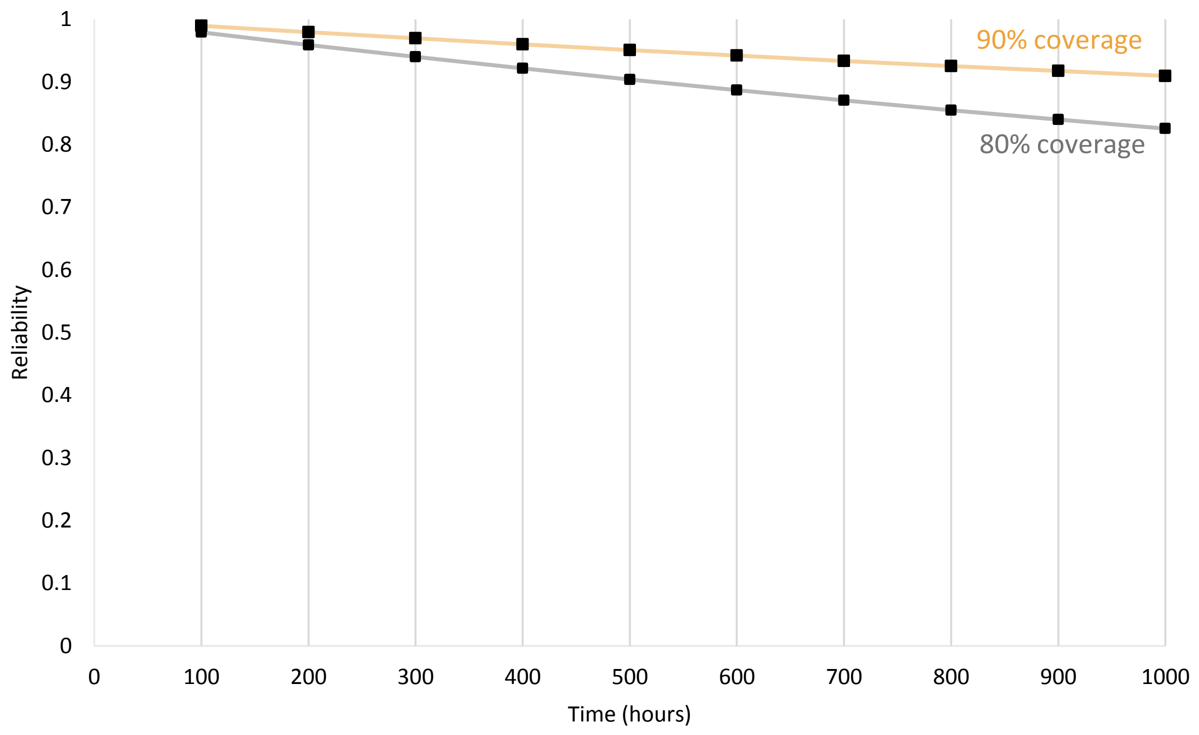


Figure 4-14: As time goes on reliability decreases with a constant failure rate λ

The results presented in Figure 4-15 show that the unreliability which represents the probability of failure in the unsafe state PFU (Time) is decreased as the coverage value (C) decreases for a constant failure rate λ.

The data of Figure 4-16 shows the change in unreliability vs. coverage (C) vs. failure rate (λ) for the system (one lane of BioSymPLe) were modeled using the Markov model and run using the WinSTEM reliability program. Eight case studies have been assumed as in table shown inside the Figure 4-16 to see the effects of both the fault coverage (C) and the failure rate (λ) on the unreliability.

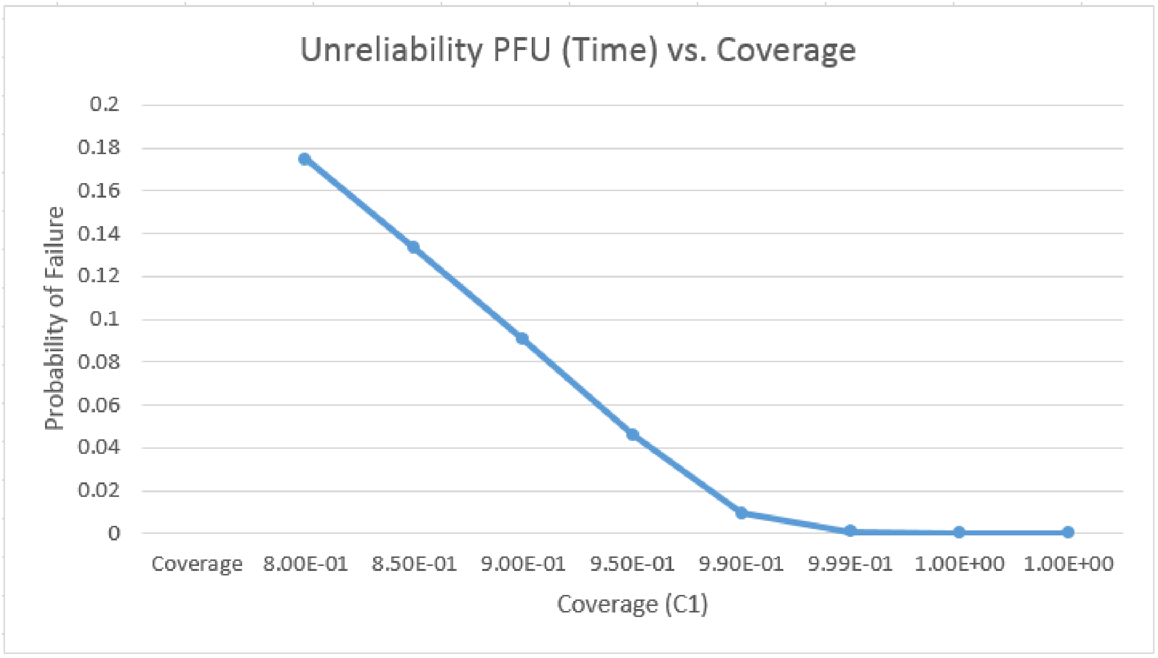
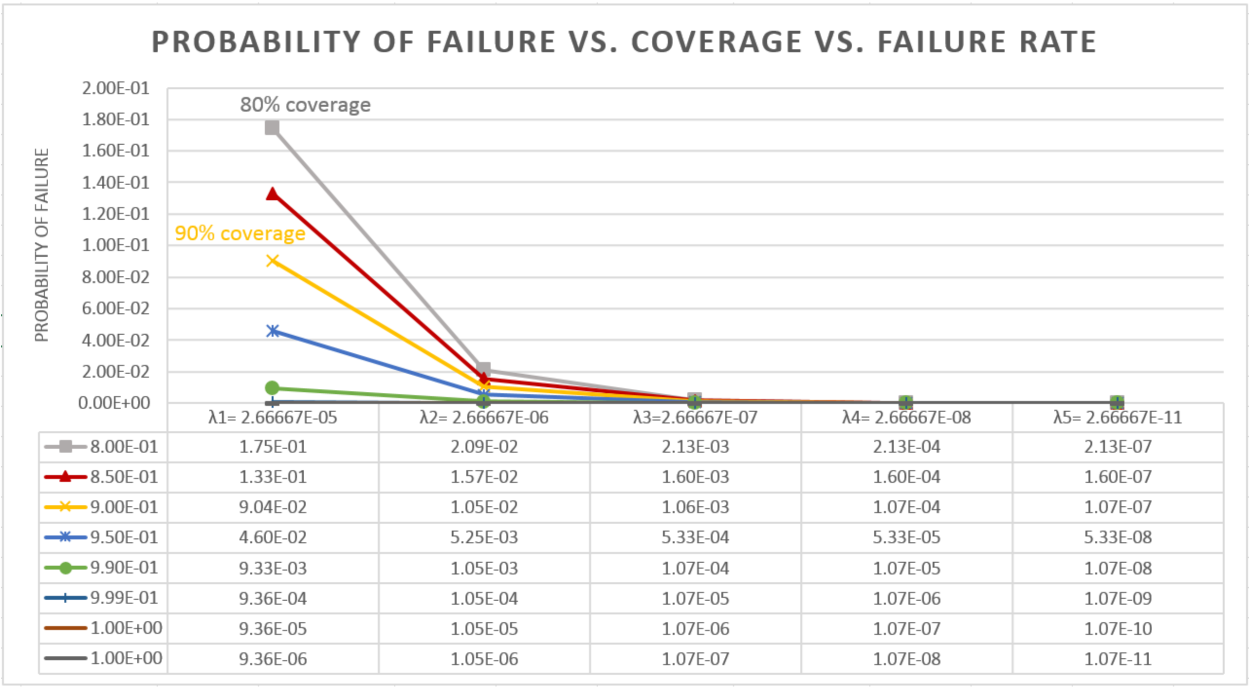


Figure 4-15: A higher coverage causes smaller unreliability

The simulation results show that at time 100 hours, for both graphs Figure 4-15 and Figure 4-16, we have the same achieved values for the unreliability with a failure rate Lambda λ value is 0.00002666667, with eight different coverage values (C= 0.8, 0.85, 0.90, 0.95, 0.99, 0.999, 0.9999, 0.99999). The unreliability values are (UR= 0.174527626753, 0.133231621356, 0.0904037078956, 0.0460059838504, 0.00933172167892, 0.000936131975976, 0.0000936428423091, 0.00000936458072464). The unreliability range for both graphs is 0.00 to 0.20 and the reliability range is 0.80 to 1. The eight coverage values 0.80 to 0.99999 represent the eight case studies highlighted with grey to dark blue colors. Consequently, the reliability R(Time) achieved for these case studies are 0.825472373247, 0.866728378644, 0.9095962921044, 0.9539940161496, 0.99066827832108, 0.999063868024024, 0.9999063571576909 and 0.99999063541927536. Therefore, the maximum reliability can be achieved for one lane of BioSymPLe is 0.99999063541927536 for the fault coverage 0.99999.



	λ1= 2.66667E-05	λ2= 2.66667E-06	λ3=2.66667E-07	λ4= 2.66667E-08	λ5= 2.66667E-11
■ 8.00E-01	1.75E-01	2.09E-02	2.13E-03	2.13E-04	2.13E-07
▲ 8.50E-01	1.33E-01	1.57E-02	1.60E-03	1.60E-04	1.60E-07
× 9.00E-01	9.04E-02	1.05E-02	1.06E-03	1.07E-04	1.07E-07
✱ 9.50E-01	4.60E-02	5.25E-03	5.33E-04	5.33E-05	5.33E-08
● 9.90E-01	9.33E-03	1.05E-03	1.07E-04	1.07E-05	1.07E-08
+ 9.99E-01	9.36E-04	1.05E-04	1.07E-05	1.07E-06	1.07E-09
— 1.00E+00	9.36E-05	1.05E-05	1.07E-06	1.07E-07	1.07E-10
— 1.00E+00	9.36E-06	1.05E-06	1.07E-07	1.07E-08	1.07E-11

Figure 4-16: A higher coverage causes smaller probability of failure for different failure rates

As it can be seen in the figure shown above, the unreliability Q (time) that was calculated based on the value of the probabilities for one lane of BioSymPLe of being in unsafe states. As this figure shows, the unreliability UR(Time) can be effected by two factors which are the value of the coverage (C1), represented by the eight colored lines and the value of the failure rate (λ), represented by the five values (λ1= 2.66667E-05, λ2= 2.66667E-06, λ3= 2.66667E-07, λ4= 2.66667E-08, λ5= 2.66667E-11) for both the transient and permanent faults.

The results show that the unreliability decreases vertically with the Y-axes as the coverage value is increased (0.80, 0.85, 0.90, 0.95, etc.). However, this conclusion cannot be seen clearly in the figure for the last three case studies which are (C1= 0.999, C2= 0.9999, C3= 0.99999) due to the high values of the coverages. Regarding the effect of failure rate on the unreliability, the results show that the unreliability decreases horizontally with the X-axes whenever the failure rate is decreased. However, this concluded result cannot be seen obviously in the case studies (λ3= 2.66667E-07, λ4= 2.66667E-08, λ5= 2.66667E-11) due to the small values of the failure rates.

4.8 Version2 Concept of BioSymPLe Architecture

The design of the second proposed version of the BioSymPLe architecture shown in Figure 4-17, was motivated by how we can achieve a higher level of self-healing capacity coverage (C) with the same amount of hardware resources: functional B cells and T cells. The system size is assumed to be comprised of N*N/2 = 8 as similar as to two connected machines of BioSymPLe verion1 presented in chapter 3. In addition, the same amount of hardware resources required for self-healing and fault management mechanisms is intended to be used in such a way that two forming health syndrome units and two syndrome switching circuits are being used in the design (see Table 4-9).

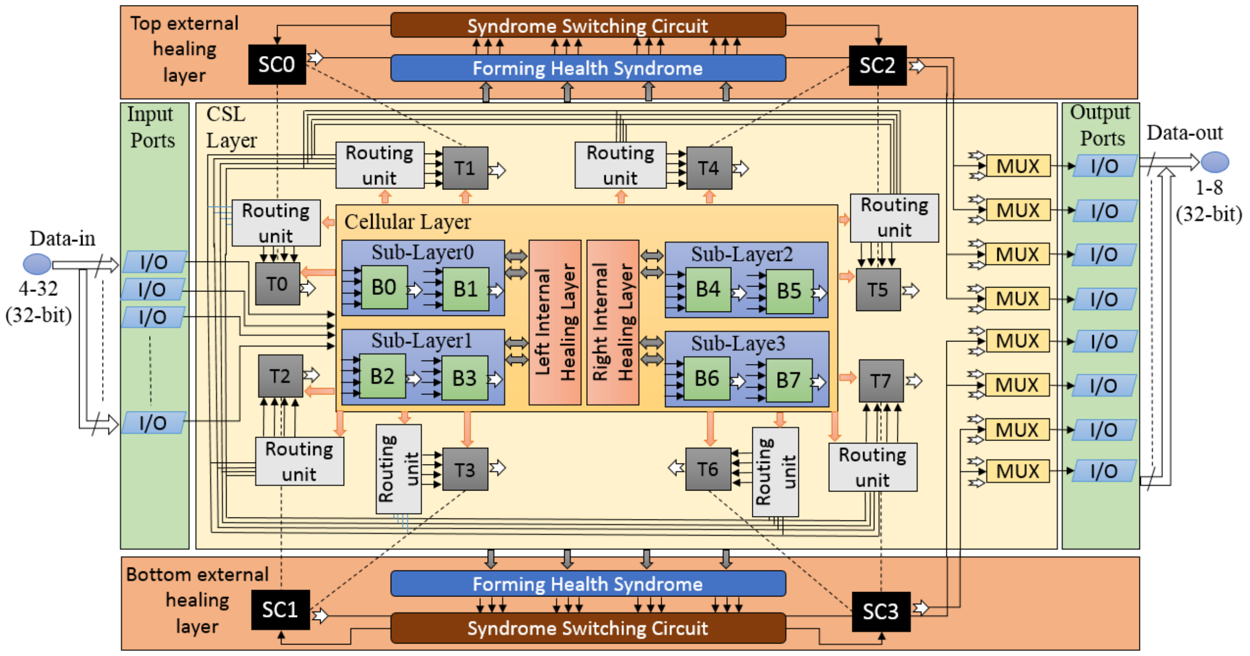


Figure 4-17: BioSymPLe architecture version2 concept

The second version of BioSymPLe is basically comprised of four principle partitions (see Figure 4-17); *the external healing layer, the critical service layer, the cellular layer, and the internal healing layer.* The *cellular layer* is inspired by B-cell functionality in immune systems. The *cellular layer*'s purpose is to host/execute digital I&C application functions and continuously monitor the correct operation of functions being executed. The cellular layer is farther decomposed into four cellular sublayers: sublayer0, sublayer 1, sublayer 2, and sublayer 3 and two internal healing layers: left and right [46]. The purpose of the sublayers is to provide redundant structure to allow continuous monitoring of I&C functions, and reconfigure when faults, failures are detected. The *Critical Service Layer (CSL)* exploits the concept of T cells in the immune system and is used to repair any one of the eight functional B cells defected by a permanent fault. The *external healing layer* including two sublayers: top and bottom and this layer is built based on a concept of embryonic stem cells.

The top external healing sub-layer is responsible for monitoring the correct behavior of the functions being executed inside the two cellular sub-layers from the top: sublayer0 and sublayer2 and triggering the required repair mechanisms to heal the critical service layer. However, the bottom external healing sub-layer is responsible for monitoring the correct behavior of the functions being executed inside the two cellular sub-layers from the bottom: sublayer1 and sublayer3. Table 4-9 shows the main similarities and differences between the two architectural concepts version1 and version2 of BioSymPLe.

TABLE 4-9
COMPARISON RESULTS BETWEEN THE TWO VERSIONS OF BIOSYMPLE

	BioSymPLe version 1	**BioSymPLe version 2**
No. of machines required for N * N = 16 Cells	Two machines	One machine
Self-healing Capacity Coverage (C)	0.833	1
Hardware Area Overhead (HAO)	125 %	150 %
Self-healing and Fault Management Resources	Two forming health syndrome units	Two forming health syndrome units
	Two syndrome switching circuits	Two syndrome switching circuits
Organization Layers of execution	Each BioSymPLe machine has three layers (Local Function Migration, Critical Service, Global Function Migration)	BioSymPLe machine has five layers (External healing: top and bottom, Critical service, , Cellular, Sublayer Internal healing: left and right)

CHAPTER 5

CONCLUSIONS AND FUTURE WORK

5.1 Research Activities

Since BioSymPLe was built based on a network of Functional Cells (FCs), its operation will depend on connecting the functional cells in a correct way. For example, if there is a three critical service layers need to be connected in order to perform the functionality of a simple critical application found in industrial automation, where a high level of dependability is required, the input signals of each functional cell in these critical layers will be connected either to other functional cells or to the output ports of the sensors, switches, or any input devices. In addition, the output signals of each FC will drive either an output device such as actuator, valve, pump, or another FC. So, the connection procedure will be based on the functional requirements of the critical application. Additionally, in BioSymPLe there are 16 different scenarios that can be assumed for the execution of four FCs inside the critical service layer.

Consequently, Markov chain models have been sketched for the proposed architecture: BioSymPLe. This analysis procedure were based on assuming some values for some parameters like the failure rate for each bio-inspired cell, the coverage for the fault detection circuits, and the Finite State Machine (FSM)-based control units. In addition, the Markov state equations for each Markov chain model were written out in order to analyze the dependable behavior of these architectures. Furthermore, after getting the N*N state transition matrix for each architecture, we were able to calculate different metrics such as the reliability R(t), Mean Time to Repair (MTTR), and Mean Time between Failures (MTBF).

5.2 Conclusion and Results Review

This dissertation has presented an architecture, design and application of a new biological-inspired hardware machine called BioSymPLe. This machine is intended for applications where; (1) resilience to failures, and (2) flexibility to reconfigure are important. The adoption of Function Block Programming allows existing PLC application to be translated into BioSymPLe more easily, and promotes understanding by the automation community. BioSymPLe is a unique approach in that resilience principles are derived from a heterogeneous perspective-combining concepts from both biologically-inspired self-healing attributes and system organization properties to achieve efficient and effective fault tolerance to multiple classes of faults. This hardware architecture has been simulated and demonstrated on several systems to date with confirmatory evidence that the approach is feasible and is practical.

The design of three biologically-inspired cells: B cell, T cell, and stem cell was based on combining concepts from biology and PLC architecture semantics. The biological concepts provided the architecture with a high level of resilience against different types of faults with recovery mechanisms at different layers in the system. However, the PLC architecture semantics has led to a concept of separation between the control flow and the data flow which is not available in other self-healing systems and that will enhance the process of verification and validation (V&V) required in safety-related systems.

Preliminary simulation and fault injection studies indicate that the proposed hierarchical multi-layered approach to resilience strikes a good balance between local fault detection's (local function migration layer) and the global fault decisions (global function migration layer) to maintain continuity of system resilience. One of the interesting findings of the BioSymPLe architecture is that B cell, T cell, and Stem cells can provide almost all of the basic resiliency functionality needed for the architecture – which suggests that the design is well-formed. The real benefit of this decision was that the process of formal verification of functional blocks, B-Cells, and T cells was systematic due to the modular nature of the cells – they all shared identical control flow semantics.

5.3 Future Work

Immediate future work will focus on three important research tasks necessary for confirming and characterizing the BioSymPLe concept.

a. The first task is to develop a semi-formal model of the organization of Bio-SymPLe. At present, the architectural and functional specifications of Bio-SymPLe are not unified under a single model. To fully reason about BioSymPLe, we need a model of its complete behavior to test it against empirical evaluations of its implementations. With the formal model we intend to employ commercial formal design verification tools (e.g. Simulink DV) to verify both the data flow and the control flow properties of each functional block inside the cells. In addition, this tool can also be used verify a diverse set of BioSymPLe properties such as the functionality of a group of B cells at the critical service level, and the traditional fault tolerance and monitoring techniques both at the register level and at the local function migration level.

b. After we develop the formal model, we intend to conduct a large-scale fault injection campaign to support verification of the models. That is we will collect real data on BioSymPLe's ability to detect, isolate, and adapt to various injected fault classes (symmetric, asymmetric, transient, permanent, intermittent, CCFs, Byzantine, etc.). These fault injection campaigns will be conducted both on simulated BioSymPle architectures and Physical implementations (e.g. BioSymPLe instances on a FPGA fabric).

c. The fault coverage parameters that have been assumed to be in different values (C= 0.8 to C= 0.99999) in chapter 4 and used in the analysis of BioSymPLe Markov models will be calculated from the large-scale fault injection experiments.

d. To fully automate the design of BioSymPLe architecture and replace the process of adding the repairing logic manually, FPGA/ASIC design and verification automation methodologies and tools will be investigated to come up with standard format. For example, in low power design, the IEEE Unified Power Format (UPF) standards which supported by the Electronic Design Automation (EDA) companies is used to specify the power intent for a given design where the synthesis engine takes both HDL and UPF files as an input and automatically infer the power intent logic around the original design. Similarly, BioSymPLe can be 1) transformed into a set of logical commands or constraints 2) deployed into synthesis tools to make synthesis engines BioSymPLe-aware so the implementation of BioSymPLe layers is handled by synthesis tools instead of being handled manually. In such case, the BioSymPLe designer is responsible to write the BioSymPLe commands or constraints which will describe the exact logical details of where BioSymPLe to be applied and what components are needed, leaving the actual implementation/design instrumentation to be handled by an EDA synthesis tool. Enabling such standards is one step toward automating self-healing techniques and can enable wider self-healing algorithms in the future.

e. Lastly, we will explore diverse design patterns for realizing new self-healing hardware architectures and cells to characterize tradeoff space between reliability, complexity, and performance in BioSymPLe.

As a final observation we note the experience of designing and using a BioSymPLe has yielded more information than just quantifying the self-healing and resiliency aspects of the system. The process itself was an iterative learning experience, allowing circumspection into how Bio-inspired systems can be pragmatic solutions to achieving more dependable embedded systems. Therefore, with the BioSymPLe project we have attempted to bring new insights into the design of bio-inspired autonomic systems for a range of stakeholders from industrial automation to the Internet of Things.

CHAPTER 6

BIBLIOGRAPHY

[1] S. Jackson, “A Multidisciplinary Framework For Resilence To Disasters And Disruptions,” *J Integr Process Sci*, vol. 11, no. 2, pp. 91–108, Apr. 2007.

[2] C. B. B. Ahmed, *Fault-Mitigation Strategies for Reliable FPGA Architecture*. LAP LAMBERT Academic Publishing, 2016.

[3] F. L. Kastensmidt, *Fault-tolerance techniques for SRAM-based FPGAs*. Dordrecht: Springer, 2006.

[4] G. W. Greenwood, *Introduction to Evolvable Hardware A Practical Guide for Designing Self-Adaptive Systems*. Hoboken: Wiley, 2006.

[5] J. J. Rodriguez-Andina, M. J. Moure, and M. D. Valdes, “Features, Design Tools, and Application Domains of FPGAs,” *IEEE Trans. Ind. Electron.*, vol. 54, no. 4, pp. 1810–1823, Aug. 2007.

[6] J. J. Rodríguez-Andina, M. D. Valdés-Peña, and M. J. Moure, “Advanced Features and Industrial Applications of FPGAs-A Review,” *IEEE Trans. Ind. Inform.*, vol. 11, no. 4, pp. 853–864, Aug. 2015.

[7] J. Srinivasan, S. V. Adve, P. Bose, and J. A. Rivers, “The impact of technology scaling on lifetime reliability,” in *International Conference on Dependable Systems and Networks, 2004*, 2004, pp. 177–186.

[8] C. Bobda, *Introduction to reconfigurable computing: architectures, algorithms, and applications*. Dordrecht: Springer-Verlag, 2007.

[9] John von Neumann, *Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components*. New Jersey: Princeton University Press, 1956.

[10] C. E. Shannon and J. McCarthy, *Automata Studies. (AM-34)*. Princeton University Press, 2016.

[11] D. Ghosh, R. Sharman, H. Raghav Rao, and S. Upadhyaya, “Self-healing systems — survey and synthesis,” *Decis. Support Syst.*, vol. 42, no. 4, pp. 2164–2185, Jan. 2007.

[12] H. Psaier and S. Dustdar, “A survey on self-healing systems: approaches and systems,” *Computing*, vol. 91, no. 1, pp. 43–73, 2011.

[13] N. Storey, *Safety Critical Computer Systems*, 1 edition. Harlow, England ; Reading, Mass: Addison-Wesley, 1996.

[14] N. G. Leveson, *Engineering a Safer World: Systems Thinking Applied to Safety*. MIT Press, 2012.

[15] R. Frei, R. McWilliam, B. Derrick, A. Purvis, A. Tiwari, and G. D. M. Serugendo, “Self-healing and self-repairing technologies,” *Int. J. Adv. Manuf. Technol.*, vol. 69, no. 5–8, pp. 1033–1061, Nov. 2013.

[16] C. S. Holling, “Resilience and Stability of Ecological Systems,” *Annu. Rev. Ecol. Syst.*, vol. 4, no. 1, pp. 1–23, 1973.

[17] C. S. Holling, “Understanding the Complexity of Economic, Ecological, and Social Systems,” *Ecosystems*, vol. 4, no. 5, pp. 390–405, Aug. 2001.

[18] C. G. Rieger, D. I. Gertman, and M. A. McQueen, “Resilient control systems: Next generation design research,” in *2009 2nd Conference on Human System Interactions*, 2009, pp. 632–636.

[19] M. Bagatin, S. Gerardin, and K. editor Iniewski, *Ionizing radiation effects in electronics: from memories to imagers*. Boca Raton ; London ; New York: CRC Press, 2016.

[20] T. S. Nidhin, A. Bhattacharyya, R. P. Behera, T. Jayanthi, and K. Velusamy, “Understanding Radiation Effects in SRAM-based Field Programmable Gate Arrays for Implementing Instrumentation and Control Systems of Nuclear Power Plants,” *Nucl. Eng. Technol.*, Oct. 2017.

[21] A. Avizienis, J. C. Laprie, B. Randell, and C. Landwehr, “Basic concepts and taxonomy of dependable and secure computing,” *IEEE Trans. Dependable Secure Comput.*, vol. 1, no. 1, pp. 11–33, Jan. 2004.

[22] B. W. Johnson, *The Design and Analysis of Fault Tolerant Digital Systems*, First Edition, First Printing edition. Reading, Mass: Addison-Wesley, 1989.

[23] W. Nantian, Q. Yanling, L. Yue, Z. Qingqi, and L. Tingpeng, "Survey on evolvable hardware and embryonic hardware," in *2013 IEEE 11th International Conference on Electronic Measurement Instruments*, 2013, vol. 2, pp. 1021–1026.

[24] X. Zhang, G. Dragffy, and A. G. Pipe, "Embryonics: A Path to Artificial Life?," *Artif. Life*, vol. 12, no. 3, pp. 313–332, Jul. 2006.

[25] Z. Zhang, Y. Wang, S. Yang, R. Yao, and J. Cui, "The research of self-repairing digital circuit based on embryonic cellular array," *Neural Comput. Appl.*, vol. 17, no. 2, pp. 145–151, Mar. 2008.

[26] P. K. Lala and B. K. Kumar, "An architecture for self-healing digital systems," *J. Electron. Test.*, vol. 19, no. 5, pp. 523–535, 2003.

[27] C. Bernardeschi, L. Cassano, and A. Domenici, "SRAM-Based FPGA Systems for Safety-Critical Applications: A Survey on Design Standards and Proposed Methodologies," *J. Comput. Sci. Technol.*, vol. 30, no. 2, pp. 373–390, 2015.

[28] U. Farooq, Z. Marrakchi, and H. Mehrez, "FPGA architectures: An overview," in *Tree-based Heterogeneous FPGA Architectures*, Springer, 2012, pp. 7–48.

[29] "Stratix IV FPGA Core Fabric Architecture." [Online]. Available: https://www.intel.com/content/www/us/en/programmable/products/fpga/features/stxiv-alm-architecture.html. [Accessed: 16-Nov-2018].

[30] C. Ortega and A. Tyrrell, "Biologically inspired reconfigurable hardware for dependable applications," in *Hardware Systems for Dependable Applications (Digest No: 1997/335), IEE Half-Day Colloquium on*, 1997, pp. 3–1.

[31] G. Tempesti, D. Mange, P.-A. Mudry, J. Rossier, and A. Stauffer, "Self-replicating hardware for reliability: The embryonics project," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 2, pp. 9-es, Jul. 2007.

[32] S. Kim, H. Chu, I. Yang, S. Hong, S. H. Jung, and K.-H. Cho, "A Hierarchical Self-Repairing Architecture for Fast Fault Recovery of Digital Systems Inspired From Paralogous Gene Regulatory

Circuits," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 20, no. 12, pp. 2315–2328, Dec. 2012.

[33] I. Yang, S. H. Jung, and K. H. Cho, "Self-Repairing Digital System With Unified Recovery Process Inspired by Endocrine Cellular Communication," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 21, no. 6, pp. 1027–1040, Jun. 2013.

[34] I. Yang, S. H. Jung, and K. H. Cho, "Self-Repairing Digital System Based on State Attractor Convergence Inspired by the Recovery Process of a Living Cell," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 25, no. 2, pp. 648–659, Feb. 2017.

[35] S. Yin, Y. Li, Y. l Qian, and X. b Liu, "A novel cell mutual detection method for bio-inspired electronic array," in *2017 Prognostics and System Health Management Conference (PHM-Harbin)*, 2017, pp. 1–5.

[36] F. Meng, J. Cai, Y. Meng, S. Wu, and T. Wang, "Evaluation index system for embryonic self-healing strategy," in *2016 International Conference on Integrated Circuits and Microsystems (ICICM)*, 2016, pp. 86–90.

[37] D. Mange, *Microprogrammed systems: an introduction to firmware theory*. London: Chapman & Hall, 1992.

[38] D. Mange, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, and C. Piguet, "Embryonics: a new methodology for designing field-programmable gate arrays with self-repair and self-replicating properties," *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 6, no. 3, pp. 387–399, Sep. 1998.

[39] G. Tempesti, D. Mange, and A. Stauffer, "Bio-inspired computing architectures: the embryonics approach," in *Seventh International Workshop on Computer Architecture for Machine Perception (CAMP'05)*, 2005, pp. 3–10.

[40] C. Ortega and A. Tyrrell, "Biologically inspired reconfigurable hardware for dependable applications," in *IEE Half-Day Colloquium on Hardware Systems for Dependable Applications*, London, UK, 1997, pp. 3/1-3/4.

[41] Z. Qingqi, Q. Yanling, L. Yue, W. Nantian, and L. Tingpeng, “Embryonic electronics: State of the art and future perspective,” in *Electronic Measurement & Instruments (ICEMI), 2013 IEEE 11th International Conference on*, 2013, vol. 1, pp. 140–146.

[42] N. Wang, Y. Qian, Y. Li, and Q. Zhuo, “Design method for a multi-layer bio-inspired self-healing hardware,” in *Prognostics and System Health Management Conference (PHM-2014 Hunan), 2014*, 2014, pp. 653–657.

[43] M. Samie, G. Dragffy, A. M. Tyrrell, T. Pipe, and P. Bremner, “Novel Bio-Inspired Approach for Fault-Tolerant VLSI Systems,” *IEEE Trans. Very Large Scale Integr. VLSI Syst.*, vol. 21, no. 10, pp. 1878–1891, Oct. 2013.

[44] M. Samie, G. Dragffy, and T. Pipe, “Unitronics: A novel bio-inspired fault tolerant cellular system,” in *Adaptive Hardware and Systems (AHS), 2011 NASA/ESA Conference on*, 2011, pp. 58–65.

[45] Khairullah S.S., Bakker, T., and Elks C.R., “Toward biologically inspired self-healing digital embedded devices: Bio-SymPLe,” presented at the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies, San Francisco, CA., 2017.

[46] S. S. Khairullah and C. R. Elks, “A Bio-Inspired, Self-Healing, Resilient Architecture for Digital Instrumentation and Control Systems and Embedded Devices,” *Nucl. Technol.*, vol. 202, no. 2–3, pp. 141–152, Jun. 2018.

[47] Bruce Alberts, *Molecular biology of the cell*, Sixth edition.. New York, NY: Garland Science, Taylor and Francis Group, 2015.

[48] B. Alberts, *Essential cell biology*, Fourth edition.. New York, NY: Garland Science, 2014.

[49] S. Jackson, “Resilience Architecting,” in *Architecting Resilient Systems*, John Wiley & Sons, Inc., 2009, pp. 159–186.

[50] I. Polian, J. P. Hayes, S. M. Reddy, and B. Becker, “Modeling and Mitigating Transient Errors in Logic Circuits,” *IEEE Trans. Dependable Secure Comput.*, vol. 8, no. 4, pp. 537–547, Jul. 2011.

[51] E. Monmasson and M. Cirstea, "Guest Editorial Special Section on Industrial Control Applications of FPGAs," *IEEE Trans. Ind. Inform.*, vol. 9, no. 3, pp. 1250–1252, Aug. 2013.

[52] E. Monmasson, L. Idkhajine, M. N. Cirstea, I. Bahri, A. Tisan, and M. W. Naouar, "FPGAs in Industrial Control Applications," *IEEE Trans. Ind. Inform.*, vol. 7, no. 2, pp. 224–243, May 2011.

[53] M. Chmiel, J. Kulisz, R. Czerwinski, A. Krzyzyk, M. Rosol, and P. Smolarek, "An IEC 61131-3-based PLC implemented by means of an FPGA," *Microprocess. Microsyst.*, 2015.

[54] Z. Hajduk, J. Sadolewski, and B. Trybus, "FPGA-based execution platform for IEC 61131-3 control software," *Przegląd Elektrotechniczny*, vol. 87, no. 8, pp. 187–191, 2011.

[55] M. Chmiel, R. Czerwinski, and P. Smolarek, "{IEC} 61131-3-based {PLC} Implemented by means of {FPGA}," *IFAC-Pap.*, vol. 48, no. 4, pp. 374–379, 2015.

[56] J. Knight, *Fundamentals of Dependable Computing for Software Engineers*. Chapman and Hall/CRC, 2012.

[57] I. Koren and C. M. Krishna, *Fault-Tolerant Systems*. San Francisco, UNITED STATES: Elsevier Science & Technology, 2007.

[58] "Design and analysis of fault tolerant digital systems: Johnson, B W Addison-Wesley, USA (1989) $41.35 pp 584," *Microprocess. Microsyst.*, vol. 15, no. 1, p. 64, Jan. 1991.

[59] V. A. Pedroni, *Finite state machines in hardware: theory and design (with VHDL and SystemVerilog)*. Cambridge, Massachusetts: The MIT Press, 2013.

[60] S. Romanov, "The Future of Railway Interlocking | Prover - Engineering a Safer World," *Prover*. [Online]. Available: https://www.prover.com/. [Accessed: 08-Dec-2017].

[61] A. Biere, A. Cimatti, E. M. Clarke, O. Strichman, and Y. Zhu, "Bounded Model Checking," in *Advances in Computers*, vol. 58, Supplement C vols., Elsevier, 2003, pp. 117–148.

[62] M. Sheeran, S. Singh, and G. Stålmarck, "Checking Safety Properties Using Induction and a SAT-Solver," in *International Conference on Formal Methods in Computer-Aided Design*, Berlin, Heidelberg., 2000, pp. 127–144.

[63] C. R. Elks, Bakker, T., Hite, R., Gautham, S., Venkatesh, V., and Moore, J., “SymPLe 1131: A novel architecture solution for the realization of verifiable digital I&C systems and embedded digital devices,” presented at the 10th International Topical Meeting on Nuclear Plant Instrumentation, Control, and Human Machine Interface Technologies, San Francisco, CA, USA., 2017.

[64] Marvin Rausand, “Reliability of Safety-Critical Systems | Wiley Online Books.” [Online]. Available: http://onlinelibrary.wiley.com/doi/book/10.1002/9781118776353. [Accessed: 18-Nov-2018].

[65] R. W. J. Butler, “Techniques for modeling the reliability of fault-tolerant systems with the Markov state-space approach,” NASA Langley Research Center;, Hampton, VA, United States, 19960000862, Sep. 1995.