COMPUTER FAULT TOLERANCE STUDY INSPIRED BY THE IMMUNE SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY

ATIF DEĞER CANIBEK

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
ELECTRICAL AND ELECTRONICS ENGINEERING

DECEMBER 2005

Approval of the Graduate School of Natural and Applied Sciences.

Prof. Dr. Canan Özgen
Director

I certify that this thesis satisfies all the requirements as a thesis for the degree of Master of Science.

Prof. Dr. İsmet Erkmen
Head of Department

This is to certify that we have read this thesis and that in our opinion it is fully adequate, in scope and quality, as a thesis for the degree of Master of Science.

Prof. Dr. Hasan Güran
Supervisor

Examining Committee Members

Asst. Prof. Dr. Cüneyt Bazlamaççı	(METU, EEE)	
Prof. Dr. Hasan Güran	(METU, EEE)	
Dr. İlkay Ulusoy	(METU, EEE)	
Dr. Ece Schmidt	(METU, EEE)	
Gökhan Göksügür (MSc.)	(ASELSAN)	

ABSTRACT

COMPUTER FAULT TOLERANCE STUDY INSPIRED BY THE IMMUNE SYSTEM

CANIBEK, Atıf Değer

MSc., Department of Electrical and Electronics Engineering

Supervisor: Prof. Dr. Hasan Güran

December 2005, 81 pages

Since the advent of computers numerous approaches have been taken to create hardware systems that provide a high degree of reliability even in the presence of errors. This study seeks to address the problem from a biological perspective using the human immune system as a source of inspiration. The immune system uses many ingenious methods to provide reliable operation in the body and so may suggest how similar methods can be used in the design of reliable systems.

This study provides a brief introduction into a relatively new discipline: artificial immune systems (AIS) and demonstrates a new application of AIS with an immunologically inspired approach to fault tolerance. It is shown a finite state machine can be provided with a hardware immune system to provide a novel form of fault detection giving the ability to detect faulty states during a normal operating cycle. It is called immunotronics.

Keywords: Artificial immune system, binary immune system, immunotronics, fault tolerance, positive tolerance conditions.

ÖZ

BAĞIŞIKLIK SİSTEMİNDEN ESİNLENİLMİŞ BİLGİSAYAR HATA TOLERANSI ÇALIŞMASI

CANIBEK, Atıf Değer
Yüksek Lisans, Elektrik ve Elektronik Mühendisliği Bölümü
Tez Yöneticisi: Prof. Dr. Hasan Güran

Aralık 2005, 81 sayfa

Bilgisayarların geniş bir biçimde kullanılmaya başlamasından bu yana hata meydana geldiğinde bile yüksek derecede güvenilirlik sağlayan donanım sistemlerinin yaratılmasında pek çok yaklaşım olmuştur. Bu çalışma insan bağışıklık sisteminden esinlenerek biyolojik bir perspektiften problemi ele almaktadır. Bağışıklık sistemi vücudun güvenilir bir biçimde çalışmasını sağlayan çeşitli akıllı yöntemler kullanır ve buna benzer yöntemlerin güvenilir sistemlerin tasarımında nasıl kullanılabileceğini öne sürer.

Bu çalışma yeni bir disiplin olan yapay bağışıklık sistemine kısa bir giriş yapmakta ve bağışıklık sisteminden esinlenilmiş yaklaşımla hata toleransına getirilen yeni bir uygulama gösterilmektedir. Donanımsal bağışıklık sistemi ile birlikte sonlu durum makinasının nasıl normal çalışma döngüsünde hatalı durumların sezilme yeteneğini kazandıran hata tanımanın yeni bir biçimini sağladığı gösterilmektedir.

Anahtar Kelimeler: Yapay bağışıklık sistemi, “binary” bağışıklık sistemi, “immunotronics”, hata toleransı, pozitif tolerans durumları.

To My Parents

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my supervisor, Prof. Dr. Hasan Güran, for his guidance, advice, criticism and encouragement and insight throughout the research.

I would also like to thank my parents, my uncle Asım Canıbek and my friends Zeynep Ceren Nurata, Cem İpekgil, Ceren İpekgil and Tarık Parlak for their valuable helps, unlimited patience, encouragements and suggestions during the preparation of this thesis.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

AIS	Artificial Immune System
ASM	Algorithmic State Machine
BCD	Binary Coded Decimal
BIS	Binary Immune System
BIST	Built-In Self-Test
CAM	Content Addressable Memory
EHW	Evolvable Hardware
FSM	Finite State Machine
NMR	N-Modular Redundancy
PLA	Programmable Logic Array

CHAPTER 1

INTRODUCTION

1.1 Fault-Tolerant Systems

Electronic systems have been used widely in everyday life. Electronic systems like computer can be found not only in banks, at schools, and in our homes, but also in nuclear power plants, expensive satellites and life-supporting medical equipments. While the presence of a fault on one system is often just an annoyance, for safety-critical applications, if the computer or any other electronic control system fails, the costs in terms of money and even human life could be incalculable. Therefore, many electronic systems must be fully functional under the most critical operating conditions, and should work satisfactorily for at least a predetermined period of time. There are two possible approaches to achieve this objective: to build either **fault free** or **fault-tolerant** systems [1].

Since, with time, hardware becomes worse and software becomes ever more complex to test reliably, it is impossible to design and build fault free system that will not develop faults during its operating lifetime. Therefore the more feasible alternative is to implement systems, which are capable of tolerating faults.

1.2 The Development of Fault-Tolerant Computing

The first theoretical work in fault-tolerant computing is generally credited to John von Neumann. In 1956 von Neumann published an article entitled: "Probabilistic Logics and the Synthesis of Reliable Organisms from Unreliable Components". In that study, he presented the concept of majority voting and analyzed the impact that such arrangements could have on the probability of a system producing erroneous results [2].

Since around 1970, the field of fault tolerant computing has been rapidly developing. Several Journals such as the Computer, IEEE Micro, the Proceedings of the IEEE etc., regularly present special issues that deal with fault-tolerant computing.

Recent years has shown radically different approaches to the design of reliable systems with biology being a major source of inspiration. Our understanding of the natural or, more specifically, human immune system has increased over the last few decades and has provided a miraculous insight into how the body defends itself from intrusion and maintains reliable operation. Through the increased understanding, new techniques inspired by the human immune system have given rise to improved approaches to computer security [3], virus protection [4], [5], anomaly detection [6], process monitoring [7], robot control [8], and software fault tolerance [9]. The human immune system provides a distributed fault-tolerant architecture within the body and so suggests a radically different approach to current reliable system design.

1.3 Purpose and Scope

Artificial immune systems (AIS) are computer systems exploiting the natural immune system metaphor: protect an organism against intruders. The subject of this thesis is about binary AIS in which all the information is represented by the bit strings of fixed length. It is shown that a finite state machine can be provided with a hardware immune system to provide a novel form of fault detection giving the ability to detect every faulty state during a normal operating cycle [10].

The purpose of this study is to find an answer to the problem that can be stated as follows: given a set of self patterns representing normal behaviour of a system under considerations find a set of detectors (i.e, antibodies, or more precisely, receptors) identifying all non-self patterns corresponding to abnormal states of the system.

1.4 Layout of Thesis

In chapter 2, the requirements for fault-tolerant hardware design are discussed.

In chapter 3, human immune system is introduced and the feature that have inspired the development of a hardware immune system is discussed.

In chapter 4, the similarities and differences between immunology and fault tolerance are presented and steps needed to immunize a system for providing fault detection are demonstrated.

In chapter 5, binary immune system is explained and the brief overview of the receptors generation algorithm is presented.

In chapter 6, the simulation that are used to construct the tolerance conditions for immunization of the system is explained.

In chapter 7, implementation of the fault detection hardware, results of the immunization cycle and the analysis of these results are presented.

In chapter 8, a conclusion of this study is given.

CHAPTER 2

FAULT TOLERANCE

Over thirty years ago, developments based on the challenge of designing reliable systems from unreliable components resulted in the notion of fault tolerance.

2.1 The Phases of Fault Tolerance

Fault tolerance techniques, either software or hardware, always imply the use of redundancy. In general, fault-tolerant systems should be able to implement the following phases [17]:

1. **Error detection** or output deviating from the norm;
2. **Minimization** or **eradication** of the resulting effects of the fault;
3. Activation of a suitable **recovery** procedure. There are two methods of recovery: a) **Backward error recovery** can return the system to a previously stored valid state and b) **Forward error recovery** can make selective corrections to the current state until an acceptable state is reached.

Biologically inspired fault tolerance must address these processes.

2.2 Hardware Fault Tolerance

There are several approaches to implement fault tolerance of hardware.

1. **Retry Strategy:** The operation, which was detected to be erroneous is to be repeated. Here redundancy is only needed to detect the fault.

2. **Backup Computers:** In case of failure, until the repair personnel could fix the main system, all work is transferred to the backup system.
3. **Pair-and-Spare:** At the component level designer uses a pair of identical components to built a unit that detects its own error. At the system level of organization, the designer builds a computer using a pair of error-detecting units. One pair operates as the main unit and the other as a spare unit. Control unit automatically switches the operation to the spare unit if the main unit fails.
4. **N-Modular Redundancy (NMR):** It is similar with pair-and-spare logic, but with N identical components (N≥3). Special voting logic compares the outputs and accepts the majority output as being correct.
5. **Built-In Self-Test (BIST):** The task of testing a chip with several millions of transistors is extremely complex, expensive and often very time consuming. A widely accepted approach to deal with the testing problem at the chip level is to incorporate BIST capability within the chip. One way of achieving self-checking design is the use of error detecting codes. In this case a code checker detects the presence of faults when its input is not a member of the set of valid codes. Figure 2.1 shows the block diagram of totally self-checking circuit.

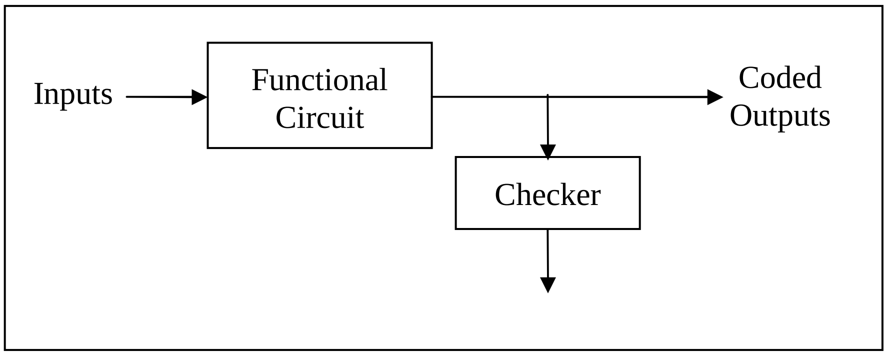


Figure 2.1 Totally self-checking circuit

2.3 Alternative Techniques in Hardware Fault Tolerance

Nature demonstrates radically different approaches to complex problem solving that are now being used within computing and electronic fields to improve the behaviour of the system instead of the classical techniques. Many different species possess defence mechanism that can be referred to as an immune system. Vertebrates have evolved a highly complex, multi-layered defence mechanism in the form of the immune system to provide protection from potentially hazardous external influences such as bacterial and viral infections (antigens). The similarities between the requirements of fault tolerance and the operation of the body's defence mechanisms have highlighted the relevance of the immune system as a conceptual model for the design of future reliable systems [11].

CHAPTER 3

HUMAN IMMUNE SYSTEM

The human immune system (and that of all vertebrates) is unique amongst all living species in that it has evolved a complex genetic level defence mechanism for protection from invaders. The detailed information of the human immune system is presented in [12] and [13].

3.1 Immune Architecture and Organisation

Defence against intruders, or **antigens** is accomplished through four distinguishable layers of protection [4] [14] from physical barriers through physiological barriers in the forms of temperature and acidity to chemical and cellular interactions in the forms of innate and acquired immunity. Acquired immunity involves antibody and cell mediated immunity that defend against extra-cellular and intra-cellular infection respectively.

The immune system cells are produced from stem cells in the bone marrow and are divided into three types of cell [15]:

1. **Macrophages** are roaming scavenger cells that take part in both innate and acquired immunity. They perform a signalling role presenting fragments of antigens to other cells of the immune system.
2. **B cells** can recognise antigens and produce a single type of antibody to counteract a specific antigen.

3. **T cells** are developed to form **helper**, **suppressor** and **killer** T cells. Helper and suppressor cells act as the master switched for the immune system by initiating and slowing down immune responses in the presence of an antigen. Killer T cells detect and destroy virus infected cells.

Response to an antigen occurs by the use of complementary receptors on the antigen and antibody known as **epitopes** and **paratopes** respectively. Both B and T cells have the ability to detect and counteract only one type of antigen and so huge diversity is a necessity. Such specificity means that at any one time there are over 10^{12} B cells within the body creating over 10^{8} different types of antibody - a number impossible to encode for in the human genome of 10^{5} genes. The rearrangement of antibody protein segments creates the huge variation needed. The presence of such a wide number of different antibodies means that an exact epitope-paratope match rarely occurs. An immune response can be initiated by approximate matching in a process called **affinity maturation** (Figure 3.1) [15].

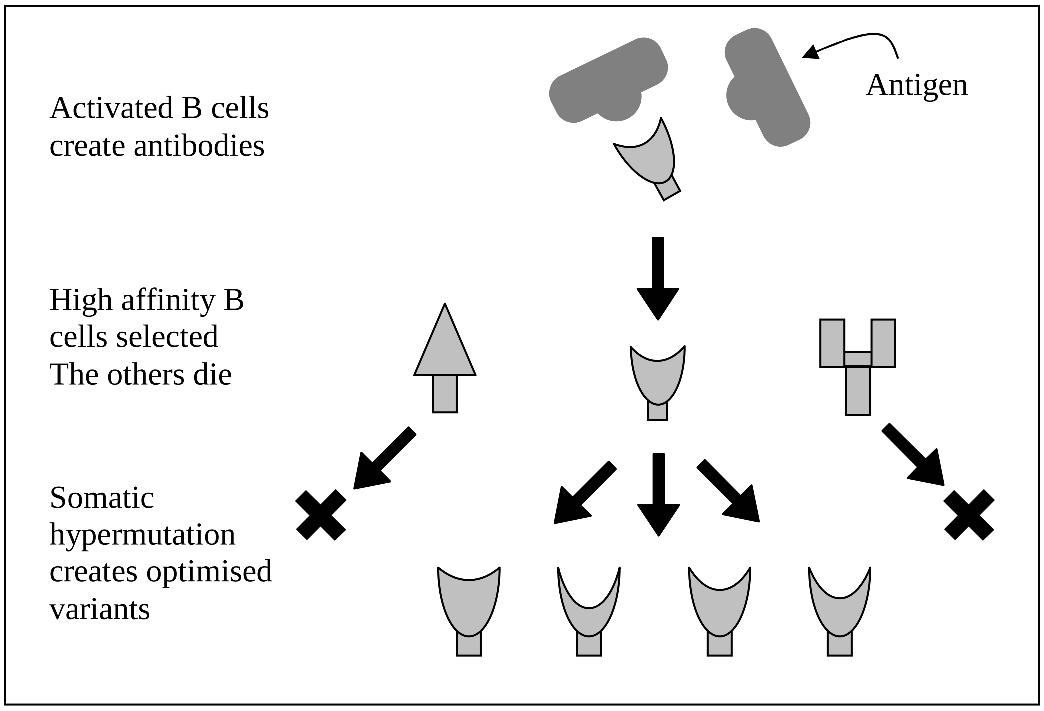

Activated B cells create antibodies
Antigen
High affinity B cells selected The others die
Somatic hypermutation creates optimised variants

Figure 3.1 Affinity maturation of antibodies

Under a continuous cycle of repeated optimisation, the B cells with the highest affinity to the invading antigen generate minor variants by **somatic hypermutation** resulting in an over-whelming quantity of antibodies to destroy the invading antigen [15].

3.2 Antibody Mediated Immunity

Antibody mediated immunity protects the body from extra-cellular infection. B cells are constantly on patrol for antigens that they can bind to. If an approximate match occurs between a patrolling B cell and an antigen a response is initiated as in Figure 3.2. Proliferation of antibodies only occurs if the corresponding T cells exist to stimulate the manufacture of optimised antibodies. When an antigen is encountered for the first time it takes several days for antibody proliferation to occur. Through the use of memory B and T cells secondary responses can provide a much more rapid response to the same infection at a later date [15].

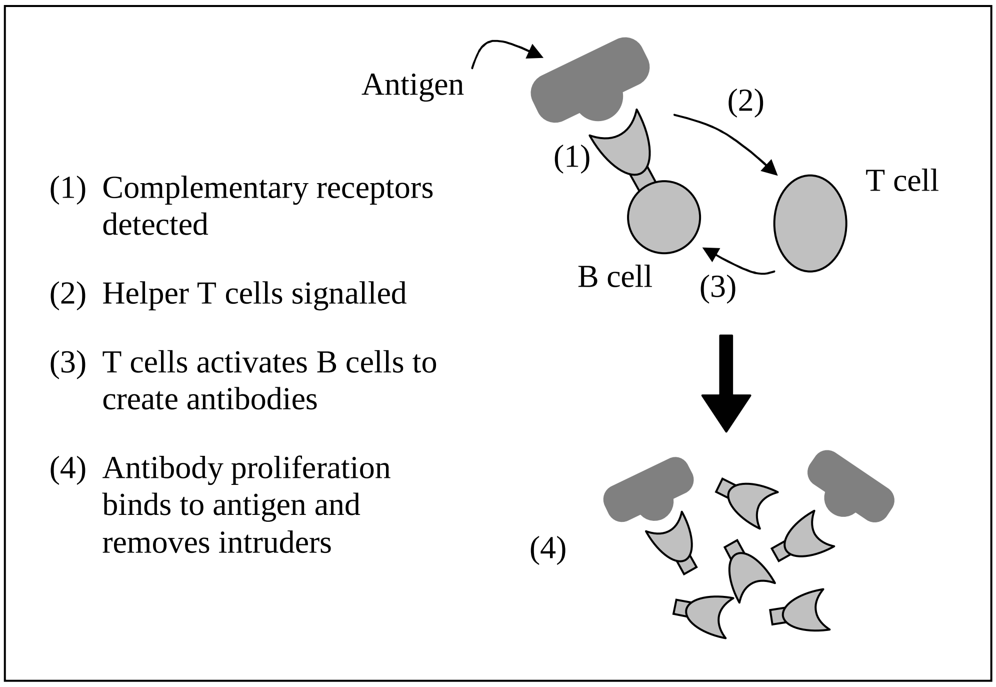

Antigen
(1)
(2)
T cell
B cell
(3)
(1) Complementary receptors detected
(2) Helper T cells signalled
(3) T cells activates B cells to create antibodies
(4) Antibody proliferation binds to antigen and removes intruders
(4)

Figure 3.2 Antibody mediated immunity

3.3 Self/Non-Self Discrimination

The process of antigen detection by random generation and complementary receptor matching is a very effective method of protection, but what prevents the immune system from binding to cell proteins that occur naturally within the body? Several theories have been proposed to explain how the immune system differentiates between self and non-self cells of the body [16]. The most widely accepted answer is **clonal deletion** which is demonstrated in Figure 3.3. In contrast to the matured functional immune system with distributed censoring, a centralised development stage occurs first and carries out a process called **negative selection**. Immature helper T cells move to the thymus where self-proteins circulate through and are exposed to the helper T cells. If a maturing T cell binds to one of the self proteins, it is destroyed. Only those T cells that are self tolerant survive to become fully functional activators of B cells [15].

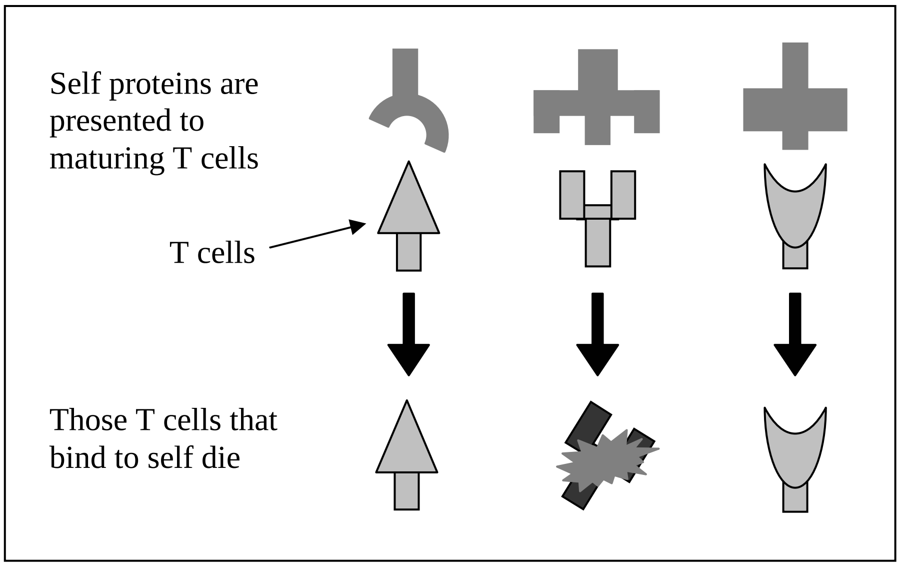

Self proteins are presented to maturing T cells
T cells
Those T cells that bind to self die

Figure 3.3 Clonal deletion

CHAPTER 4

THE IMMUNOLOGICAL TO HARDWARE TRANSITION

4.1 Key Immunological Features

Based upon the fundamental attributes of the immune system, the following five key analogies can be summarised [15]:

1. The immune system functions continuously and autonomously, only intervening the normal operation of the body when an intruder or erroneous condition is detected, much like in the presence of a faulty state. In a mapping to hardware, the analogy is that of fault detection and removal without the need for software support.
2. The cells that provide the defence mechanisms are distributed throughout the body through its own communications network in the form of lymphatic vessels to serve all the organs. The hardware equivalent promotes distributed detection of faults with no centralised fault recognition and recovery.
3. The immune cells that provide the detection mechanisms are present in large quantities and exist with a huge range of diversity. Limited diversity is already a common solution to fault tolerant system design.
4. The immune system can learn and remember from past experiences what it should attack. The hardware analogy suggests the training (and possibly even continued improvement during operation) of fault detection mechanisms to differentiate between fault free and faulty states.

5. Detection of intruding antigens by the immune system is imperfect. The onset of faults in hardware systems is often due to the impossibility to exhaustively test a system. The testing phase can never test for every eventuality and so the analogy of imperfect detection suggests one remedy.

4.2 Hardware Representation

Methods of self/non-self differentiation are developed through a finite state machine (FSM) representation of the system to be immunised. In principle, any hardware system can be represented by an individual or set of inter-connected finite state machines and is therefore a logical start point. Finite state machine define the acceptable states and transitions between states (Figure 4.1) [10].

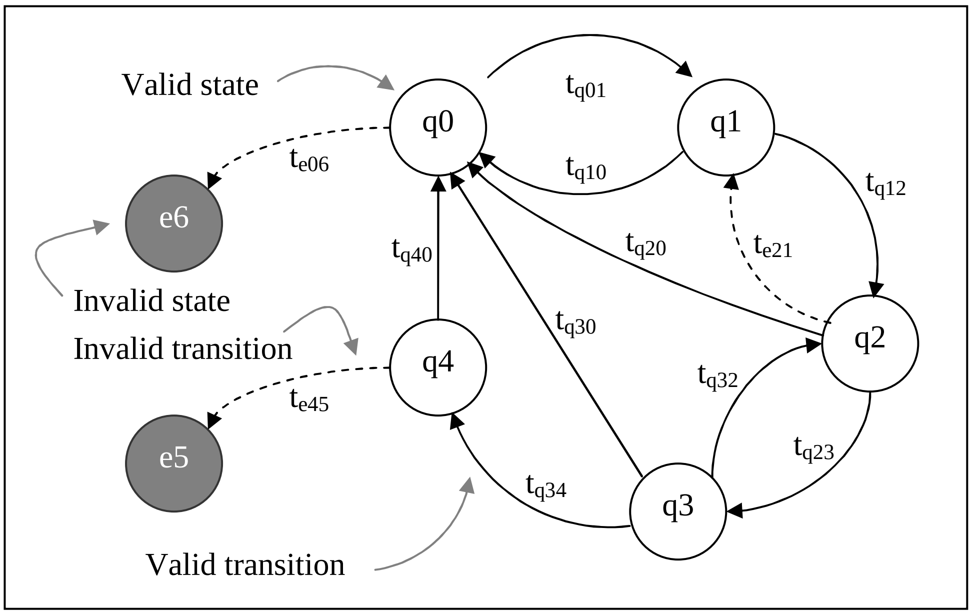


Figure 4.1 State machine under test

Under normal and reliable operation (self) only transitions t_{qx} can occur. Invalid transitions, t_{ex}, signify a potential problem (non-self). Concentrating on the transitions between states rather than the state themselves is a much better solution, as two states may valid, but the transition not [17]. Table 4.1 demonstrates the advantages of monitoring state transitions.

Table 4.1 The advantage of monitoring state transitions
CS – current state, NS – next state

CS	NS	Valid NS	Transition	Valid Transition
2	3	Yes	t_{q23}	Yes
4	**5**	No	t_{e45}	No
2	**1**	Yes	t_{e21}	**No**

4.3 Feature Mapping

The features and operations of the immune system can be translated into the hardware domain. Tables 4.2 and 4.3 summarise the mappings in terms of **entities**, or physical elements and **processes**, or operations that the system may undergo. Tables 4.2 and 4.3 show that an immunologically inspired approach based upon the use of FSMs is feasible [17].

Table 4.2 Entity feature mapping

Immune System	Hardware Fault Tolerance
Self	Valid state/state transition
Non-self (antigen, intruder)	Invalid state/state transition
Antibody	Error tolerance conditions
Gene used to create antibody	Variables forming tolerance conditions
Antigen presenting cell	Data collection and tolerance condition creation component
Paratope	Invalid state/state transition tolerance conditions
Epitope	Valid state/state transition tolerance conditions
Helper T cell	Recovery procedure activator
Memory T cells	Set of tolerance conditions

Table 4.3 Process feature mapping

Immune System	**Hardware Fault Tolerance**
Recognition of self	Recognition of valid transition
Recognition of non-self	Recognition of invalid transition
Learning during gestation	Learning of correct transition
Antibody mediated immunity	Error detection and recovery
Clonal deletion	Isolating of self-recognising tolerance conditions
Inactivation of antigen	Return the normal operation
Life of organism	Operation lifetime of the hardware

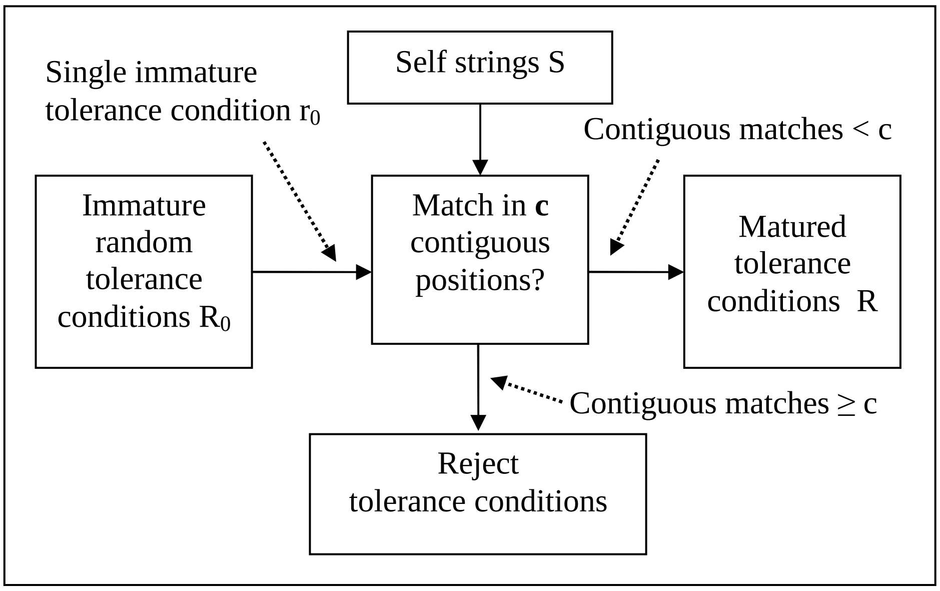
4.4 Data Gathering

The immunotronic hardware initially undergoes a learning stage similar to the centralised maturation of T cells in the thymus. The goal of the data gathering stage is to create a data set S that represents a complete or substantial percentage of all possible valid state transitions within normal operation of the FSM. Unlike the human immune system which does not store a map of self conditions, storing this within the immunotronic hardware has its uses. The generation of invalid conditions can be initiated through the injection of faults and simulated errors. This only provides a starting point for the learning of non-self however. Approximate matching techniques, through the use of non-deterministic methods (random/ evolutionary generation) is to be used to provide a set of partial matching data [18]. The current approach is to immunise the state of the system by monitoring the inputs, current state and previous state of the system and storing each instance of self as a binary string [15].

4.5 Tolerance Condition Generation

The negative selection algorithm was developed by Forrest and Perelson [4] from theoretical analyses of the matching and binding properties of the immune system for the detection of viruses within computer systems and network intrusion. In contrast to the existing fault tolerant architecture, such as NMR and embryonics which work by checking constantly for the presence of valid operation, the negative selection algorithm works by checking constantly for the presence of invalid operation.

The algorithm is based upon the method of selecting a set of strings R of length ℓ from a randomly generated original set of data R_0. Each string $r \in R$ fails to match any of the self strings $s \in S$, also of length ℓ, in at least c contiguous positions. Any string that match in at least c contiguous positions are deleted. The mature set of tolerance conditions R are generated from an initial randomly generated set R_0 corresponding to immature tolerance conditions, which undergo a negative selection process [17]. Figure 4.2 adopted from [19] demonstrates the algorithm [10].



Self strings S
Single immature tolerance condition r_0
Contiguous matches < c
Immature random tolerance conditions R_0
Match in **c** contiguous positions?
Matured tolerance conditions R
Contiguous matches ≥ c
Reject tolerance conditions

Figure 4.2 The negative selection algorithm

4.6 Architecture of the Hardware Immune System

With the tolerance conditions generated they must then be downloaded to the host hardware immune system (Figure 4.3 [10]).

Hardware immune system acts as a wrapper to the state machine under protection. Under normal operation, only self strings are present. The presence of a fault creates a non-self state, analogous to the presence of antigen.

The string generation component gathers the user inputs and system state (and/or output) from the state machine, combining with the previous system state (and/or output) to create a search string for presentation to the immune system memory.

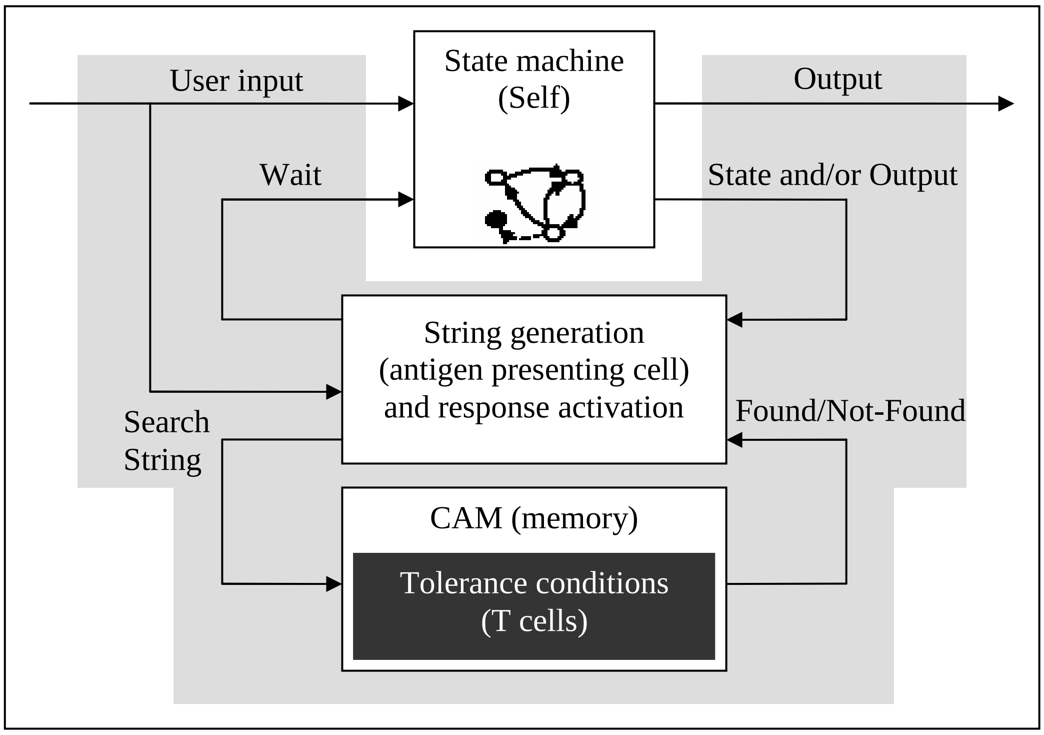


Figure 4.3 The hardware immune system attached to the FSM

The partial-matching content-addressable memory (CAM) stores the tolerance conditions and returns a positive result if c contiguous bits out of ℓ match the search string. The data are presented to the CAM and a found or not-found signal returned in addition to the address where the data were found [17].

4.7 Fault Detection

The immunized state machine is monitored at every change of state and the gathered data sent to the tolerance condition memory and searched [17]. If valid state is confirmed, then normal operation is allowed to continue. If faulty state is detected, number of defined responses can be activated. If the memory read clock driving the CAM is fast enough then the result of search to be returned to the "string generation and response activation" component of Figure 4.3 can be gotten before the current internal state of the state machine propagates to the output on the next clock cycle. Therefore, the internal states of the hardware should always be monitored, rather than just the outputs, because in this way it is possible to detect fault before the effects have propagated to the output.

4.8 Fault Recovery

The ideal way of removing any potential problems in the body is the destruction of a cell, which was detected as infected, due to the enormous levels of redundancy. Such process is not ideal, but often necessary, for hardware because of the finite level of redundancy. In the presence of an intermittent error, a more ideal approach would be recovery or repair.

The potential methods of achieving fault removal can be summarised:

1. **Classical Architecture:** Creating a form of NMR that is replicate the protected state machine and switch to a spare if the first is detected as faulty. It is not ideal to disable a large hardware component unless a disastrous failure occurs. Because, if a self string is detected accidentally as non-self through incomplete coverage of self strings or the presence of a fault within the tolerance condition storage, then a fault-free state machine may be deactivated completely. In this method, transient errors result in the deactivation of the state machine that operates normally to specification.

2. **Immunologically Inspired Architecture:** The detection of a non-self string may signal the user to request the next action. The possibility of providing two sets of tolerance conditions has been discussed in [18] so that a set of potential recovery states corresponding to self string can also be stored. It may then be possible to automatically correct the faulty output through an output multiplexer selecting either the normal or the immune-activated response. This means that transient errors do not result in the deactivation of the complete system [17].

   The phenomenal cellular redundancy in the body enables normal operation to persist when cells are neutralized and later destroyed due to infection. In an FSM architecture, a similar technique could be used through the use of spare states or latch bits within the hardware. The detection of a fault would then cause the state machine to be reconfigured to ensure the faulty state was circumvented and a spare state used [17].

3. **Total Biologically Inspired Architecture:** Embryonics and EHW are two other biologically inspired approaches to fault detection and tolerance. Embryonics is based upon the development of multi-cellular organism. Through the development of biological multi-cellular organism, cells differentiate according to “instructions” stored in their DNA. Depending on the position of the cell within the embryo, different parts of the DNA are interpreted. Before differentiation, because each cell possesses a copy of the DNA, cells are (theoretically) able to take over any function within the body. Correspondingly, every electronic cell in an embryonic array stores not only its own configuration register, but also those of its neighbours [17]. To differentiate, every cell selects a configuration register according to its position, which is determined by a set of coordinates that is calculated from the coordinates of the nearest neighbours, within the array.

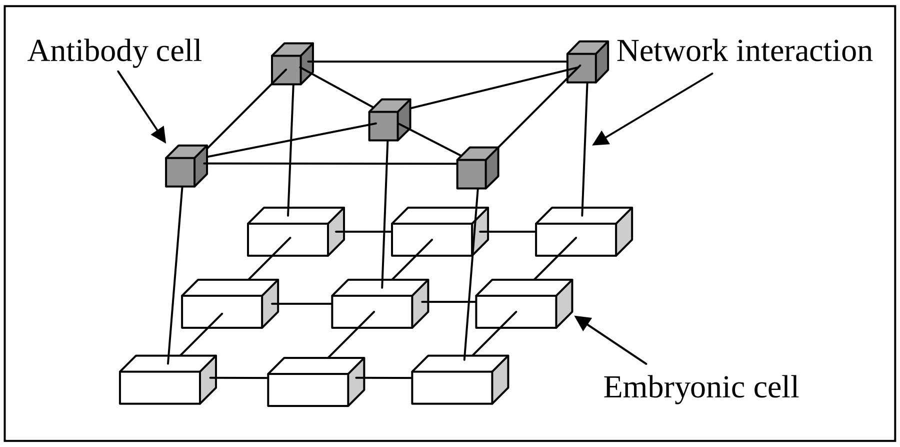


Figure 4.4 Mapping of lymphatic interactions to an integrated immunotronic-embryonic multilayered fault-tolerant architecture

Every embryonic cell continuously performs self-checking. In the presence of fault, the faulty cell issues a status signal that eliminates the cell. By recalculating their coordinates and selecting a new configuration register, every surviving cell performs a new function. The integration of a cellular hardware immune system within the architecture, as shown in Figure 4.4, removes the need of self-checking from each embryonic cell. By doing so, there is no need for duplication of functional units within each cell. This architecture allows each immune cell or antibody cell to continuously monitor its neighbouring embryonic cells for faults. Due to the repeated checking of every embryonic cell by more than one antibody cell, interaction between neighbouring antibody cells also allows for error detection within each antibody cell. If the results from the antibody cells that have checked the operation of the same embryonic cell differ, then the faulty antibody cell is deactivated and the array reconfigured [17].

CHAPTER 5

BINARY IMMUNE SYSTEM

Binary immune system introduced in 1987 by Farmer, Packard and Perelson. Instead of a genetic alphabet with four symbols (Adenine, Cytosine, Thymine, and Guanine) the model uses a binary alphabet. Both receptors and intruders (foreign/non-self cell or molecule) are represented as binary strings of fixed length.

5.1 Intrusion Selection Algorithm

The principles of self/non-self discrimination in the immune system are the inspiration of the intrusion selection algorithm, where any intruder should be distinguished from the body cells [20]. Below an abstract formulation of the algorithm is presented.

Let U be the set of all binary strings of length ℓ; obviously $|U|$, the cardinality of U, equals 2^{ℓ}. Let $S \subseteq U$ be a proper subset of U, called self strings, which represent e.g. valid states of a system. The strings from the set U-S are referred to as non-self strings. The problem relies upon constructing a set of detectors, denoted R, such that each $r \in R$ does not recognize any self string, and it should recognize at least one non-self string representing invalid state of the system. This way of detecting invalid states was proposed by [4] under the name of negative selection method. It has a number of interesting features. The most important, among them, are [20]:

1. **No prior knowledge of anomaly is requested**.

2. **Detection is probabilistic and tuneable:** instead of constructing a set of detectors recognizing all non-self strings (complete repertoire) a smaller set of detectors is generated. It recognizes all but a small fraction P_f (failure probability) of non-self strings in exchange for a smaller set of detectors.

3. **Detection is local:** only small sections of data are checked and when a detector does find an anomaly it can be localized to the string that the detector is checking.

4. **Detection is distributable:** small sections of the protected system can be checked separately and no communication among detectors is needed until an anomaly is detected.

The strings from R can be loosely treated as a concise characterization of a notion U-S. Denoting by R* the set of strings detected by the receptors in R, the problem can be stated as follows: knowing the description of S, find a subset $R \subseteq U\text{-}S$ of minimal cardinality such that $R^* = U\text{-}S$. Here, typically, the cardinality of S is relatively small in comparison with the cardinality of U [20].

To implement the algorithm of identifying the set R the followings should be defined in general: receptors representation (binary in our case), the method of their activation (matching rule), and the method of receptors generation [20].

5.2 Matching Rules

There is no unique receptors activation method. Probably a simpler one is Hamming matching: two strings x and y match ($\text{match}_H(x,y)$) under the rule if they have different bits in at least c positions, $1 \leq c \leq \ell$, i.e.

$$\text{match}_H(x,y) \text{ iff } d_H(x,y) \geq c \qquad (1)$$

where $d_H(x,y)$ stands for the Hamming distance between x and y. The total number of strings recognized by a single receptor $r \in R$ under the Hamming match with threshold c, $D_H(\ell,c)$, equals

$$D_H(\ell,c) = \sum_{i=c}^{\ell} \left[\begin{matrix}\ell \\ i\end{matrix}\right] \tag{2}$$

Knowing this number, $p_H(\ell,c)$ - the probability that two random strings match at least c bits can be easily found: $p_H(\ell,c) = 2^{-\ell} \cdot D_H(\ell,c)$ [20].

In this study c-contiguous bits rule [21] is used as a plausible abstraction of receptor binding in the immune system. Two strings, x and y match under the rule if x and y have the same bits in at least c contiguous positions. Suppose for instance that $\ell = 6$, $c = 3$ and assume that the strings r (receptor), x1 and x2 are of the form r = 110110, x1 = 001100 and x2 = 010100. Then $match_C(r,x1)$ = FALSE while $match_C(r,x2)$ = TRUE, so x1 is a self pattern and x2 is an antigen (anomaly) [20].

$$\boxed{\begin{array}{ll} r & 1\ \boxed{\mathbf{1\ 0\ 1}}\ 1\ 0 \\ x2 & 0\ \boxed{\mathbf{1\ 0\ 1}}\ 0\ 0 \end{array}}$$

Figure 5.1 Matching under the c-contiguous rule

Matching under c-contiguous bits rule can be imagined as moving a window of width c over the receptor (r) and tested (x) strings (Figure 5.1). If the two substrings within the window are identical, receptor activates.

5.3 Templates

Moving the window of width c over the self strings (Figure 5.1), we can split each of them into (ℓ-c+1) substrings of length c. These substrings induce templates to build receptors. Since each receptor does not recognize any self string, $s \in S$, it is obvious that it can not contain any template recognized in a self string.

To be more precise, let w be a binary string of length c (c is the threshold value). Consider strings of length ℓ over the alphabet {0,1,*} where * stands for irrelevant. By a template $t_{i,w}$ of order c, a string (of length ℓ), whose substring of length c taken from position i equals w, and all the remaining positions of the template are filled by the star symbol, is expressed. For instance, when ℓ = 6, c = 3, and w = 011 then $t_{1,w}$ = 011***, $t_{2,w}$ = *011**, $t_{3,w}$ = **011*, and $t_{4,w}$ = ***011. A self string s = 001101 splits into four templates: $t_{1,001}$ = 001***, $t_{2,011}$ = *011**, $t_{3,110}$ = **110*, and $t_{4,101}$ = ***101. A template of order c is a schema [22] of order[1] c in which all the significant bits are contiguous.

The set of all possible templates, denoted T, contains $(\ell\text{-}c+1)\cdot 2^c$ different elements. T can be split into two disjoint subsets: T_S consisting of all the templates contained in at least one self string and the set of remaining templates, T_N, used to construct receptor strings. Typically T_S is a low fraction of T [20].

5.4 Discriminative Power of a Receptor

Consider a single receptor $r = b_1b_2,...,b_\ell$ where $b_i \in \{0,1\}$ denotes bit value at i-th position, i = 1,...,ℓ. The problem is to find the number D(ℓ,c) of unique strings from U detected (by means of the c-contiguous bits rule) by the receptor r. Obviously this number depends on the receptor length and the threshold value only. To find D(ℓ,c) all the templates $t_{i,w}$ constituting a given receptor by the set of schemas forming a partition of the set of all detected strings will be represented. In other words, if $X = \{x_1,...,x_m\}$ is the set of schemas generated by the receptor and u is an antibody

[1] The order of a schema is defined as the number of relevant positions in this schema. For instance if x1 = 0000**** and x2 = 00000*** the order(x1) = 4 and order(x2) = 5.

detected by r then u is an instance[2] of exactly one schema $x_i \in X$. A schema derived from a template $t_{i,w}$ has first (c+i-1) positions meaningful and remaining (ℓ-c-i+1) positions are filled in by the star symbol. To find the number D(ℓ,c) two cases are considered: a simpler one when $c \geq (\ell/2)$ and more complicated case when $c < (\ell/2)$ [20].

5.4.1 The Threshold Value c ≥ (ℓ/2)

In this case the number D(ℓ,c) can be found by counting the number of schemas generated by the templates $t_{i,w}$, i =1,...,ℓ-c+1. The template $t_{1,u}$, where $u = b_1b_2,...,b_c$, detects strings that agree with the schema $b_1b_2...b_c$*...* containing (ℓ-c) star symbols. The template $t_{2,v}$, where $v = b_2b_3...b_{c+1}$, detects strings agreeing with the schema $*b_2b_3...b_{c+1}$*...* containing (ℓ-c-1) stars. According to our convention this schema divides into two schemas: $b_1b_2b_3...b_{c+1}$*...* and $(1\text{-}b_1)b_2b_3...b_{c+1}$*...*, where $(1\text{-}b_1)$ stands for the complement of b_1. The first schema is an instance of the schema induced by the template $t_{1,u}$; hence only second schema is fresh, i.e. it recognizes new strings. Similarly, the template $t_{3,w}$, where $w = b_3b_4,...b_cb_{c+1}b_{c+2}$ splits into four schemas: $b_1b_2...b_cb_{c+1}b_{c+2}$*...*, $(1\text{-}b_1)b_2...b_cbc+1b_{c+2}$*...*, $b_1(1\text{-}b_2)...b_cb_{c+1}b_{c+2}$*...*, and $(1\text{-}b_1)(1\text{-}b_2)...b_cb_{c+1}b_{c+2}$*...*. The first schema is an instance of the schema generated by the first template and the second schema is an instance of a schema generated by the second template. Thus only third and fourth schemas are fresh. To list all fresh schemas generated by all the templates contained in a receptor r, proceed as follows [23]:

a. Put on a first position of a list the schema induced by the first template.
b. Let current length of the list, k_1, equals 1.
c. For any template $t_{i,w}$ (i=2,..., ℓ-c+1) do the following.
d. For j =1 to k_{i-1} copy j-th schema from the list to (k+j)-th position and replace (i-1)-th bit in the schema by its complement, $b_{i-1} < 1\text{-}b_{i-1}$.
e. Modify current length of the list: $k_i < k_{i-1} + k_{i-1} = 2^{i-1}$.

[2] That is, if x = 00000*** then e.g. u = 00000101 is an instance of x.

Table 5.1 shows the fresh (unique) schemas generated by four initial templates.

Table 5.1 Fresh (unique) schemas generated by four initial templates

Template	Substring generating template	Fresh schemas generated by the template
$T_{1,u}$	$u = b_1b_2\dots b_c$	$b_1\ \ b_2\ \ b_3\ \ b_4\ \dots\ b_c\ \ *\ \ *\ \ *\ \ *\ \dots\ *$
$T_{2,v}$	$v = b_2b_3\dots b_{c+1}$	$1\text{-}b_1\ \ b_2\ \ b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ *\ \ *\ \ *\ \dots\ *$
$T_{3,w}$	$w = b_3b_4\dots b_{c+2}$	$b_1\ \ 1\text{-}b_2\ \ b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ *\ \ *\ \dots\ *$
		$1\text{-}b_1\ \ 1\text{-}b_2\ \ b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ *\ \ *\ \dots\ *$
$T_{4,x}$	$x = b_4b_5\dots b_{c+3}$	$b_1\ \ b_2\ \ 1\text{-}b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ b_{c+3}\ \ *\ \dots\ *$
		$1\text{-}b_1\ \ b_2\ \ 1\text{-}b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ b_{c+3}\ \ *\ \dots\ *$
		$b_1\ \ 1\text{-}b_2\ \ 1\text{-}b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ b_{c+3}\ \ *\ \dots\ *$
		$1\text{-}b_1\ \ 1\text{-}b_2\ \ 1\text{-}b_3\ \ b_4\ \dots\ b_c\ \ b_{c+1}\ \ b_{c+2}\ \ b_{c+3}\ \ *\ \dots\ *$

Observe that each template $t_{i,w}$, ($i=2,\dots, \ell-c+1$), divides into 2^{i-1} schemas (because it contains i-1 leading star symbols) and only half of them is fresh. Thus i-th template ($i\geq2$) generates 2^{i-2} new schemas and each of them covers $2^{\ell-c-i+1}$ different strings (since each schema contains $\ell-c-i+1$ star symbols). In summary, first template covers $2^{\ell-c}$ strings and any other template $t_{i,w}$, $i=2,\dots,\ell-c+1$, covers $2^{i-2}\cdot2^{\ell-c-i+1}= 2^{\ell-c-1}$ different strings. The total number of strings recognized by a receptor equals [23]:

$$D(\ell,c) = 2^{\ell-c} + (\ell-c)\cdot2^{\ell-c-1} = 2^{\ell-c-1}\cdot(2+\ell-c) \qquad (3)$$

The reasoning presented here easily extends to the case when strings over an alphabet consisting of m symbols are considered. For instance when $c \geq (\ell/2)$, each template $t_{i,w}$ introduces (m-1) fresh schemata. Hence the total number of strings recognized by a single receptor equals [23]:

$$D_m(\ell,c) = m^{\ell-c} +(\ell-c)\cdot(m-1)\cdot m^{\ell-c-1} = m^{\ell-c-1}\cdot[(\ell-c)\cdot(m-1) + m] \qquad (4)$$

Dividing $D_m(\ell,c)$ by m^ℓ (total number of strings), a formula describing the probability that a randomly chosen string is detected by a receptor is obtained. It is important however that both (3) and (4) are valid only if $c \geq (\ell/2)$ [23].

5.4.2 The Threshold Value $c < (\ell/2)$

This case is more complicated and considered in two cases: (a) c is close to $(\ell/2)$, and (b) c is close to 1.

5.4.2.1 The Threshold Value is Close to $(\ell/2)$

The procedure described to list all fresh schemas generated by all the templates contained in a receptor r for threshold value $c \geq (\ell/2)$ pretty works for $i = 1,...,c+1$. Suppose now $i = c+2$ and $c < \ell/2$. Then by step (d) of the procedure (c+1)-th bit must be changed in all schemas belonging to the current list. But the first schema from the list has star symbol on this position. It means that an empty string must be inserted on (k+1)-th position since this schema has been exhausted by the first template (see Table 5.2, first row in the block corresponding to the template $t_{5,000}$). Now if $i = c+3$, both the first and second schema from the list has star symbol on (c+2)-th position. Further, current list contains empty string already introduced when the template t_{c+2},w was converted into fresh schemas. Thus, when developing the template $t_{c+3,w}$ we must insert 2+1 empty strings to the list. In general, the number of empty strings introduced by i-th template, α_i, equals [23]:

$$\alpha_i = 2^{i-c-2} + \beta_i \quad , \quad i = c + 2, \ldots, \ell\text{-}c+1 \tag{5a}$$

where

$$\beta_j = \alpha_{c+2} + \ldots + \alpha_{j-1} \tag{5b}$$

is the number of empty strings already introduced when previous templates have been developed. Table 5.3 shows the values of α_i and β_i for $i = c+2,\ldots,c+10$ [23].

Table 5.2 Fresh schemas induced by the receptor 00000000 with threshold c = 3

Template	**Schema**	**Template**	**Schema**
$t_{1,000}$	000*****	$t_{6,000}$	empty
$t_{2,000}$	1000****		empty
$t_{3,000}$	01000***		01001000
	11000***		11001000
$t_{4,000}$	001000**		00101000
	101000**		10101000
	011000**		01101000
	111000**		11101000
$t_{5,000}$	empty		empty
	1001000*		10011000
	0101000*		01011000
	1101000*		11011000
	0011000*		00111000
	1011000*		10111000
	0111000*		01111000
	1111000*		11111000

Table 5.3 Values of α_i and β_i defined in equations (5a), (5b)

	$\boldsymbol{\alpha_i}$	$\boldsymbol{\beta_i}$
c+2	1	0
c+3	3	1
c+4	8	4
c+5	20	12
c+6	48	32
c+7	112	80
c+8	256	192
c+9	576	448
c+10	1280	1024

5.4.2.2 The Threshold Value is Close to 1

When c is relatively small, the number of reduced strings must be slightly modified. When $\ell \geq 2(c+1)+c+1$ then for the $i \geq 2(c+1)+1$, equation (5b) starts to count some empty strings which are already introduced when previous templates have been converted into schemata. The number of already counted empty strings is α_{j-k+1} and the modified form of equation (5b) is:

$$\beta_j = \begin{cases} 0 & j \leq c+2 \\ \alpha_{c+2} & j = c+3 \\ \alpha_{c+2} + \ldots + \alpha_{j-1} & c+3 < j < 2(c+1)+1 \\ \alpha_{c+2} + \ldots + \alpha_{j-1} - \beta_{j-c} & j \geq 2(c+1)+1 \end{cases} \qquad (5c)$$

5.5 Reduction of the Discriminative Power

Although a single receptor can distinguish $D(\ell,c)$ unique strings from the universe U, its discriminative power radically changes when it cooperates with another receptors. To be more illustrative consider two receptors 000000 and 001100, and assume that the threshold $c = 3$. Using the method described in previous section it is easily stated that both the receptors recognize 38 unique strings and not $2 \cdot D(6,3) = 40$ strings. On the other hand, the ensemble consisting of two receptors 000000 and 100001 recognizes only 28 receptors [20].

Knowing $D(\ell,c)$, $p(\ell,c)$, the probability that a randomly chosen string $u \in U$ matches with a receptor (i.e. that u is an antigen) can be computed: $p(\ell,c) = D(\ell,c)/2^{\ell}$. When $c \geq (\ell/2)$ then [23],

$$p(\ell,c) = 2^{\ell-c-1} \cdot (\ell-c+2)/2^{\ell} = 2^{-c} \cdot [((\ell-c)/2) + 1] \qquad (6)$$

This formula was derived by Perelson [4] using binomial distribution and with additional requirement that $2^{-c} << 1$.

In general, to estimate the average number of strings recognized by a set of n receptors statistical approach should be used. Assuming that the receptors are chosen independently $p_f(\ell,c,n)$, the failure probability can be defined as [20]:

$$p_f(\ell,c,n) = (1- p(\ell,c))^n \approx e^{-n.p(\ell,c)} \qquad (7)$$

This last approximation is valid for large values of n and small values of $p(\ell,c)$. The average number of strings detected by n receptors is determined by the formula [20]:

$$d(\ell,c,n) = (1 - p_f(\ell,c,n))\cdot 2^{\ell} \qquad (8)$$

and the average number strings detected by a single receptor among the ensemble of cardinality n equals $d_{avg}(\ell,c,n) = d(\ell,c,n)/n$. Figure 5.2 shows how this number varies for different values of ℓ, c and n [23]. The parameter ℓ and c were chosen such $D(\ell,c)$ is fixed and equals 48.

To achieve maximal discrimination power of the receptors, they should be chosen in such a way that each receptor enters maximal number of different schemas. To achieve this, receptors must be build from diverse templates.

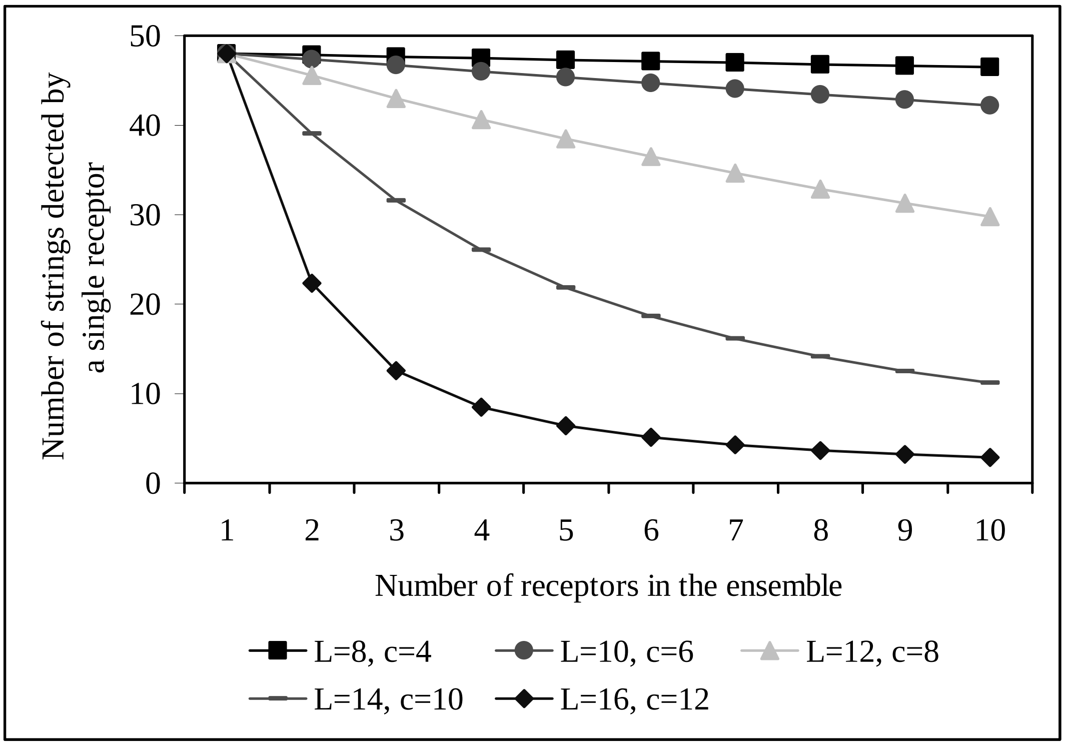


Figure 5.2 Reduction of the discriminative power of a single receptor

5.6 Lower Bound for The Fault Probability

The fault probability $p_f(\ell,c,n)$ defined in formula (7) is applied to the case when $S = \varnothing$. When $S \neq \varnothing$ it is not possible, in general, to construct detectors recognizing all the strings from the set U-S. It is hard to define the lower bound analytically, but it is relatively easy to treat the problem numerically. There are two sources of non-detectability, which will be discussed below [23].

The first source are holes defined by D'haeseleer [25]. Intuitively by hole, any string $u \in$ U-S build up from the templates belonging to the set T_S only, should be understood.

Before introducing the second source of non-detectability, some useful notions are introduced.

Let w be a substring of length c. Denote by $(\rightarrow w)$ the substring obtained by deleting the first bit from w, and by $(w\leftarrow)$ the substring resulted from deletion of last bit from w. The symbol $(\rightarrow w)+b$ denotes the string $(\rightarrow w)$ appended with b, where $b \in \{0,1\}$, and similarly $b+(w\leftarrow)$ denotes b appended with $(w\leftarrow)$; obviously in both cases the length of new strings is again c [23].

Wierzchoń [24] observed that the self strings can be represented as binary trees whose nodes correspond to the templates: a template $t_{1,w}$ is said to be the root of a tree. In general, given a template $t_{i,w} \in T_S$, $1 \leq i \leq (\ell\text{-}c)$, template $t_{i+1,(\rightarrow w)+0} \in T_S$ is called as the left child of $t_{i,w}$, and the template $t_{i+1,(\rightarrow w)+1} \in T_S$ is called as the right child of $t_{i,w}$. Surely, if $i+1 = \ell\text{-}c+1$ then the corresponding child is just a leaf of the binary tree. By analogy, given a template $t_{i,w} \in T_S$, $2 \leq i \leq (\ell\text{-}c+1)$, we call the template $t_{i-1,0+(w\leftarrow)} \in T_S$ the left parent of $t_{i,w}$, and the template $t_{i-1,1+(w\leftarrow)} \in T_S$ the right parent of $t_{i,w}$ [20].

The second source of non-detectability comes from the templates that can not be used to construct receptors. Roughly speaking for each template $t_{i,w} \in T_S$ parents has to be checked: if a parent, say $t_{i-1,v}$ is not a member of T_S, but both its children are members of T_S, $t_{i-1,v}$ is moved to the set T_S, because it can not possible to construct valid receptor which contains the template $t_{i-1,v}$. Similarly, the children of $t_{i,w} \in T_S$ are checked: if a child, say $t_{i+1,r}$ is not a member of T_S, but both its parents are member of T_S, $t_{i+1,r}$ is moved to the set T_S [20].

Counting the number of strings induced by the modified set T_S, whole number of non-detectable strings which consists of the number of self strings, the number of holes, and the number of additional non-detectable strings is determined.

5.7 Positive Tolerance Conditions

A new set of tolerance condition which is called positive tolerance conditions and new method that these tolerance conditions use for self/non-self discrimination are presented in this study.

Receptors recognize non-self strings if there is at least one match between search string and any receptor in the repertoire for c contiguous bits because all receptors are constructed from only non-self templates and the string that has at least one non-self template can not be self.

Positive tolerance conditions are constructed only from the self templates in other words positive tolerance conditions are subset of self strings set, that is, they include all the self templates but has less strings than self strings set. It is obvious that maximum number of strings in the positive tolerance conditions set is equal to the number of strings in the self strings set.

Discrimination method of the self/non-self strings or recognition of the non-self strings are different from receptors for positive tolerance conditions. Unlike the receptors a match between any positive tolerance condition and search string for c contiguous position does not mean the search string is self or non-self. Positive tolerance conditions recognize self strings if there are matches between search string and positive tolerance conditions for every c contiguous bits.

CHAPTER 6

SIMULATION

The software is written and compiled on Visual C++ 6.0. The simulator program is given in the Appendix.

The purpose of the simulation is to find the receptor repertoires of the finite state machines with the intrusion selection algorithm and to compute the failure probabilities for the different combinations of receptors from repertoire.

6.1 Application

The flow chart of the simulation is presented in Figure 6.1. Each box represents one step of the simulation. The boxes with grey background emphasizes the places where the input is asked.

There are 7 different self strings sets and for every set, self strings are written to the text documents by hand except last set.

Last self strings set is constructed by a function that is executed at the beginning of the simulation. This function asks user the length of the strings and the number of strings that will be generated. Then generates random numbers and computes the binary equivalents of these numbers. By executing the simulator, user is confronted with the small menu that asks user if he/she wants to generate random self strings set. Generated self strings set is not erased when the simulation ends. This set is kept in the text document until the random self string generator function is used and new self strings set is written on the previous one.

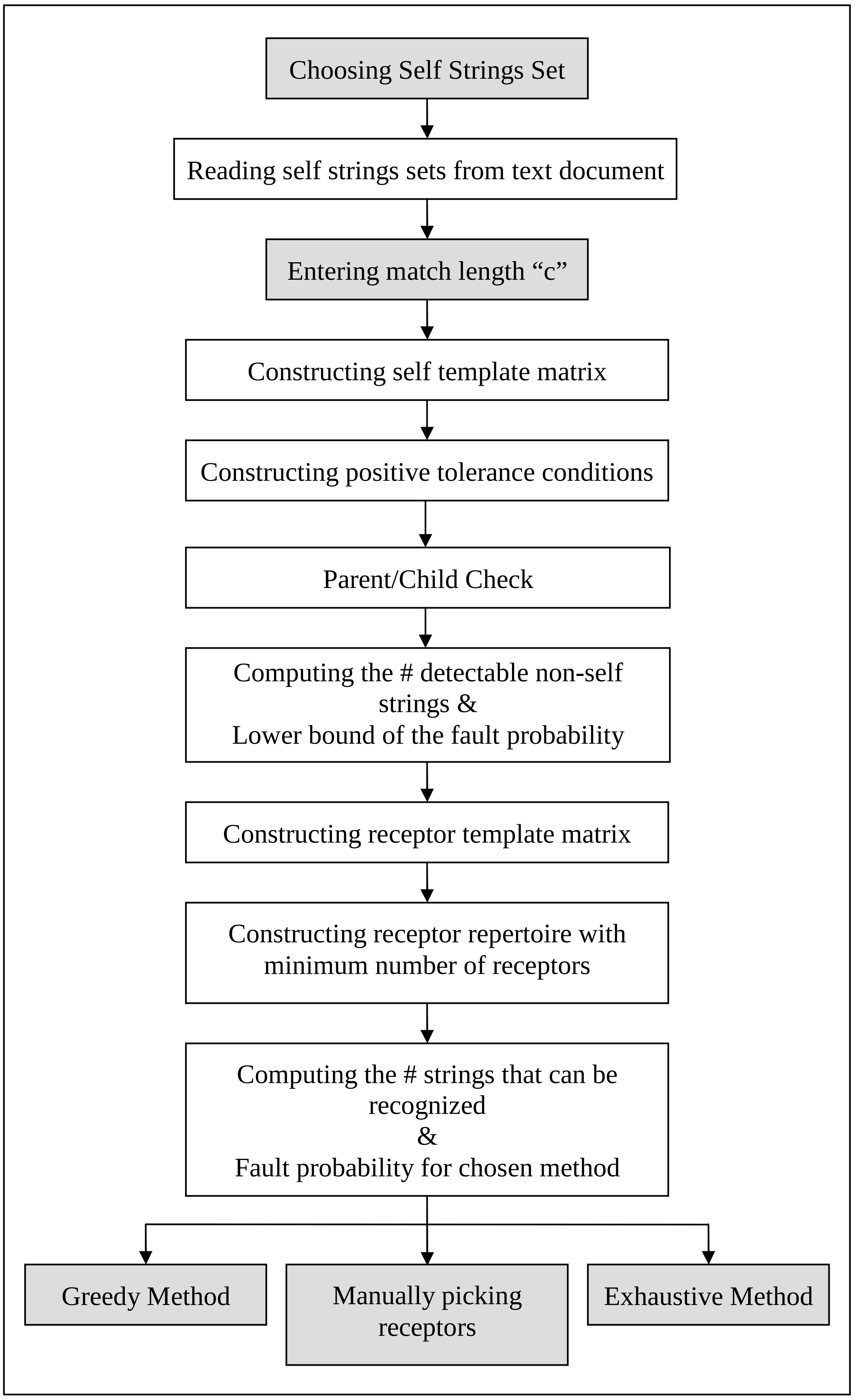

Choosing Self Strings Set
Reading self strings sets from text document
Entering match length "c"
Constructing self template matrix
Constructing positive tolerance conditions
Parent/Child Check
Computing the # detectable non-self strings & Lower bound of the fault probability
Constructing receptor template matrix
Constructing receptor repertoire with minimum number of receptors
Computing the # strings that can be recognized & Fault probability for chosen method
Greedy Method
Manually picking receptors
Exhaustive Method

Figure 6.1 Flow chart of the simulation.

Then, the below menu that is used to choose self strings sets is displayed.

(1) 4-bit 0 to 9 BCD counter.

(2) 4-bit 0 to 9 BCD counter (with different bit-string representation).

(3) 4-bit 0 to 9 BCD counter (with different bit-string representation).

(4) 4-bit 0 to 9 Gray code counter (Excess-3 gray).

(5) PLA for unsigned binary multiplier.

(6) Example ASM.

(7) Randomly generated self strings set.

After choosing the one of the self strings set, simulation reads the data from the text document. Simulation reads first character from the text document and writes it to the first column of the first row of the matrix which is called self strings matrix. Then it reads next symbol and writes it to the next column of the first row. When simulation comes to the end of line, next character is written to the first column of the next row of the self strings matrix. Simulation reads character until it reaches to the end of document. Each string in the text document is a row in the self strings matrix. Then self strings set of the FSM is displayed and the match length c to compute the self templates is asked. Here the match length should be between 1 to the length of the strings of the FSM. With entering the match length, self templates are constructed and simulation displays the self template matrix, the number of self templates and the number of holes generated by these templates. Self template matrix has ℓ-c+1 columns and 2^c rows.

By knowing self strings and match length, positive tolerance conditions can be constructed.

A simple procedure, which is proposed in this study, gathers and organizes self templates to construct the positive tolerance conditions set. First, the first column (left most column) and the last column (right most column) of the self template matrix are compared to choose the start point. The column that contains more template is chosen as a start point. If both columns contain an equal number of templates than any column can be chosen (at these circumstances simulation chooses last column). It is obvious that the column that contains the maximum number of templates identifies the minimum number of strings in the positive tolerance conditions set, because templates in the same column can not be used in the same string. Consider that the first column contains more templates, then positive tolerance conditions are constructed from first template to last, in other words, parent to child or left to right (in self template matrix). First template of the first column of the self template matrix, the template which has the minimal row number, say $t_{1,w,}$ is chosen as a first template of the first positive tolerance condition and the content of this node is increased by one. After that, children of this node are checked. If child $t_{2,(\rightarrow w)+0}$ is a member of self template set, this template is used as a second template and the value of the corresponding node is increased by one. Otherwise other child $t_{2,(\rightarrow w)+1}$ is used and the value of this template is increased by one. Continuing this reasoning first positive tolerance condition is created. After construction of each positive tolerance condition the number of fresh templates (not used in the previous positive tolerance conditions) are counted. Any template that corresponding node in the self template matrix contains 1 is called fresh. If no fresh template used in the positive tolerance condition then this is deleted from the positive tolerance conditions set. Of course each template is considered as fresh for only one receptor. It is obvious that the number of fresh templates added by the first positive tolerance condition is ℓ-c+1. To construct the second positive tolerance condition next template in the first column of the self templates matrix is used. Then the children of this template are checked but in this case the least used child is preferred. It means, if both children of this template are member of the self template set then the contents of these children are compared and the one that contains lower value (less used) is chosen. By using this procedure all the templates of the first column of the self template matrix are used. After constructing each valid positive tolerance condition (adds at least one fresh template) the number of fresh templates added by the positive tolerance condition set and the total number of templates in the self template matrix is compared. If there are templates which are not used in the positive tolerance conditions, the same procedure is repeated. When the number of fresh templates added by the positive tolerance conditions set reaches the total number of templates in the self template matrix, the procedure ends and positive tolerance conditions and the number of non-self strings that they recognize are displayed.

Although the number of non-self strings that positive tolerance conditions recognize is simply the total number of strings minus number of self strings and number of holes, simulation computes this number by computing the number of non-self strings that each positive tolerance condition recognizes.

Then parents and children of all self templates are checked. After these computations are performed, self template matrix, number of additional templates (comes from checking parents and children of self templates) and the total number of templates (self templates and additional templates) are displayed.

At this point, simple procedure counts the total number of non-detectable strings (the strings that can not be detected even complete receptor repertoire used). Suppose moving from the leaves toward the root of the tree (moving from right to left in the modified self template matrix). A node $t_{\ell-c,w}$ in the matrix has at most two children: $t_{\ell-c+1,(\rightarrow w)+0}$ and $t_{\ell-c+1,(\rightarrow w)+1}$. If both children are the member of the self template set (contains 1), it means that two self strings can be constructed: one ends with 0 and the second ends with 1. Hence the values of the self template matrix should be updated according to the rule:

$$\text{If } T[w,i]=1 \text{ then } T[w,i] = T[(\rightarrow w)+0,i+1] + T[(\rightarrow w)+1,i+1]\ ,\ i = (\ell-c,\ldots,1)$$

Modification of the self template matrix may result with updating some nodes with 0 which were member of self template set (previously contains 1). This can only happen when both children of the node are not member of self template set (both contains 0) and this is only possible for additional templates. Because self template matrix will also be used to create receptor template matrix, update procedure should be carried out on the replica of the self template set. After updating procedure, summing up all the entries (nodes) in the first column of the replica matrix will give the total number of non-detectable strings.

The updated copy of the self template matrix, total number of non-detectable strings, the number of detectable non-self strings (when complete receptor repertoire is used) and the percentage of the number of detectable non-self strings to the number of all non-self strings are displayed. Because the number of detectable non-self strings is known, lower bound of the fault probability can be calculated and displayed. Lower bound of the fault probability means that the failure probability of of the system when complete receptor repertoire is used.

After the computation of the number of detectable non-self strings simulation starts to construct the receptor repertoire. The 0's in the self template matrix are identifies the templates that can be used to construct receptor repertoire. New matrix which is called receptor template matrix is created. This new matrix is replica of the self template matrix. Because 0's in the self template matrix are identifies the receptor templates, each node in the receptor templates matrix has to be changed such that T[w,i] = 1 - T[w,i].

After constructing the receptor template matrix total number of templates to construct receptors and maximum number of receptor that can be constructed by these templates are displayed. The procedure of computing the maximum number of receptors is the same procedure that was used to compute the total number of non-detectable strings. At this point, the copy of the receptor template matrix has to be used to avoid loss of data as explained in computation of the number of non-detectable strings.

Now the problem is to find the minimal number of receptors that recognizes all detectable non-self strings which includes all the templates in the receptor template matrix. Receptors are constructed with the same procedure which is used in the positive tolerance conditions construction.

After the procedure, complete receptor repertoire and the number of non-self strings recognized by the repertoire is displayed.

This simple menu is displayed under the complete receptor repertoire.

(0) – EXIT.

(1) – Select Receptors one by one.

(2) – Exhaustive Method to select receptors.

(3) – Greedy Algorithm to select receptors.

With using this menu receptor ensembles from the receptor repertoire can be chosen. To compute the number of strings that one receptor recognizes or receptor ensemble recognize, two new matrix is created. First matrix which contains all possible strings that can be created for the strings length is called AllString. It has 2^{ℓ} rows and ℓ columns. The other is called Flag which is used as a flag matrix for AllString matrix. It has 2^{ℓ} rows and 1 column. When receptor recognizes a string from the AllString matrix, the corresponding row in the Flag matrix is set to 1.

By choosing “Select Receptors one by one”, the number of strings that can be recognized by each receptor in the receptor repertoire can be computed. Simulation asks to enter the row number of receptor in the receptor matrix. Counting the number of strings that the chosen receptor recognizes is done in this way: first templates of the receptor and first string in the AllString matrix are compared, if there is a match, the corresponding row (first row) in the Flag matrix is set and next string in the AllString matrix and receptor are compared. If not the other templates are compared. If there are no matches between receptor and the first string for ℓ-c+1 templates then the chosen receptor does not recognize the first string. It is important to note that if there is a match between templates of the receptor and the corresponding template of the compared string there is no need to check other templates because one match is sufficient enough to identify string as a non-self. Therefore match between first templates lets next string be compared without comparing other templates. When all strings in the AllString matrix and the receptor are compared, the number of strings that chosen receptor recognizes is computed by counting the number of 1’s in Flag matrix. The number of strings recognized and the failure probability are displayed. Entering the row number of other receptor is resulted with displaying the total number of strings recognized by these two receptors. Entering “-1” clears the Flag matrix and entering “0” turns back to the menu.

By choosing “Exhaustive Method to select receptors”, simulation asks user to enter the number of receptor. For the entered value k, all k combinations of the receptor repertoire are computed and the best result is displayed. Here the best result means the highest number of strings recognized (lowest failure probability) by k strings. For the receptor repertoire which has a lot of receptors, this computation takes very long time. For large receptor repertoires, other method that makes computation more simple is needed.

By choosing “Greedy Algorithm to select receptors”, simulation asks user to enter the number of receptor. For the entered value k ($2 \leq k \leq$ receptor repertoire), greedy algorithm chooses receptors as follows: first the number of strings recognized by every couple in the receptor repertoire are computed and the couple that recognizes largest number of strings is chosen. After that if the k is bigger than 2, at least one new receptor has to be added to this couple. Next receptor is chosen in this way: first receptor of the receptor repertoire (chosen receptors are not included) is chosen and the number of strings which these three receptor recognize is computed. Then every other receptors in the repertoire are added to the couple one by one and the receptor that recognize maximum number of strings with the couple is chosen. This procedure is repeated until the number of receptors in the ensemble reaches to k.

CHAPTER 7

IMPLEMENTATION, RESULTS AND ANALYSIS

In this chapter, different self strings sets of the FSMs will be used to analyze the behaviour of the binary immune system. A number of parameters, computed by simulation, will be presented and the reasons of the variation in these parameters will be investigated.

7.1 Implementation

In this study the immunization of the small systems are demonstrated. Fault-detection hardware is relatively simple to extract the data from the system under test, and a memory device with simple control logic. Generic model of hardware immune system was demonstrated in Figure 4.3. The detailed form of that model is presented in Figure 7.1.

In this detailed implementation, 0 to 9 BCD counter is protected or immunized. Under normal operation, only self strings are present. The presence of a fault creates a non-self (invalid) state.

The string generation component of this implementation gathers the user inputs (Count and Reset) and system state (or output) from the state machine, combining with the previous system state to create a search string for presentation to the immune system memory. The generated string is sent to the argument register of the memory unit.

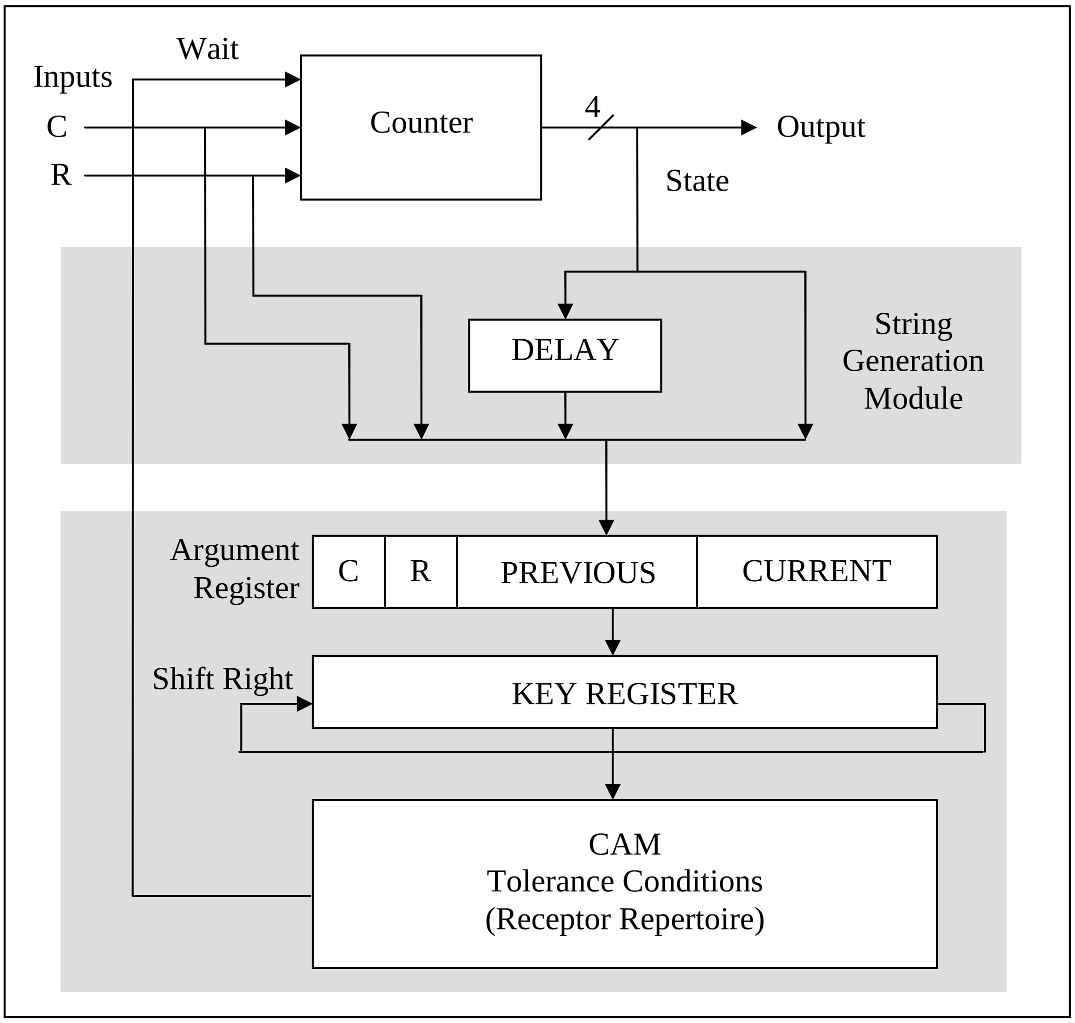


Figure 7.1 Fault-detection hardware

The partial-matching content-addressable memory (CAM) stores the tolerance conditions (receptor repertoire) and returns a positive result if c contiguous bits out of ℓ match the search string.

The key register provides a mask for choosing c contiguous bits in the argument register. Only the bits in the argument that have 1's in their corresponding position of the key register are compared. The block diagram of CAM is shown in Figure 7.2.

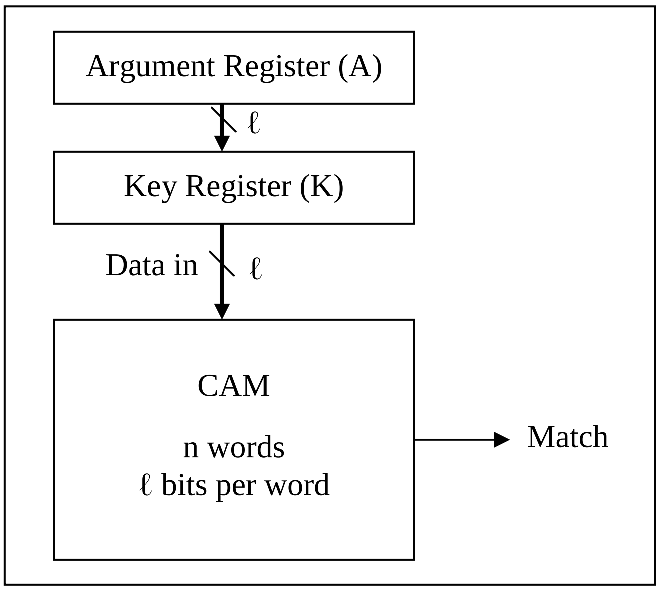


Figure 7.2 Block diagram of partial matching CAM

The immunized state machine is monitored at every change of state and the gathered data sent to the tolerance condition memory and searched. A string is deemed non-self if any tolerance condition matches the generated string in c contiguous position.

7.2 Self Strings Sets

5 different sources are used to generate self strings sets and these are:

- 4-bit 0 to 9 binary coded decimal (BCD) counter.
- 4-bit 0 to 9 Gray code counter (Excess-3 gray).
- PLA for binary multiplier.
- Control system of the example ASM.
- Randomly generated self strings set.

The structure and function of the 4-bit 0 to 9 BCD counter is presented in Table 7.1.

Table 7.1 Structure and function of the BCD counter

Function	0 to 9 counter
States	10
Size (bits)	4
Inputs	Count (C) Reset (R)
Operation	Incremental count (C=1, R=0) Hold (C=0, R=0) Reset (C=X, R=1)

Three different self strings sets are generated from BCD counter according to the bit-string representation of the strings. Self strings sets of both BCD and Gray code counters are generated by combining the two inputs, count and reset, previous and current states. Each string in these sets is 10 bit in length. Figure 7.3 shows the 3 different bit-string representation for BCD counter.

Count (C) / Reset (R) / Previous State / Current State

Count (C) / Reset (R) / Current State / Previous State

Reset (R) / Current State / Count (C) / Previous State

Figure 7.3 Bit-string representations of BCD counter

Because 10 valid states and 2^2 valid inputs to determine the forthcoming state, there are 40 strings that define self and hence valid operation in the self strings sets of the both BCD and Gray code counters. For the BCD counter defined in Table 7.1, all self strings with bit-string representation of Count/Reset/Previous State/Current State is given in Table 7.2. Number of self strings and total number of non-self strings for 4-bit 0 to 9 BCD counter and Gray code counter are 40 and 984 respectively.

Table 7.2 Self strings set of the counter (C/R/Previous/Current)

Inputs		Previous State				Current State			
C	R								
0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1
0	0	0	0	1	0	0	0	1	0
0	0	0	0	1	1	0	0	1	1
0	0	0	1	0	0	0	1	0	0
0	0	0	1	0	1	0	1	0	1
0	0	0	1	1	0	0	1	1	0
0	0	0	1	1	1	0	1	1	1
0	0	1	0	0	0	1	0	0	0
0	0	1	0	0	1	1	0	0	1
0	1	0	0	0	0	0	0	0	0
0	1	0	0	0	1	0	0	0	0
0	1	0	0	1	0	0	0	0	0
0	1	0	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0	0	0
0	1	0	1	0	1	0	0	0	0
0	1	0	1	1	0	0	0	0	0
0	1	0	1	1	1	0	0	0	0
0	1	1	0	0	0	0	0	0	0
0	1	1	0	0	1	0	0	0	0
1	0	0	0	0	0	0	0	0	1
1	0	0	0	0	1	0	0	1	0
1	0	0	0	1	0	0	0	1	1
1	0	0	0	1	1	0	1	0	0
1	0	0	1	0	0	0	1	0	1
1	0	0	1	0	1	0	1	1	0
1	0	0	1	1	0	0	1	1	1
1	0	0	1	1	1	1	0	0	0
1	0	1	0	0	0	1	0	0	1
1	0	1	0	0	1	0	0	0	0
1	1	0	0	0	0	0	0	0	0
1	1	0	0	0	1	0	0	0	0
1	1	0	0	1	0	0	0	0	0
1	1	0	0	1	1	0	0	0	0
1	1	0	1	0	0	0	0	0	0
1	1	0	1	0	1	0	0	0	0
1	1	0	1	1	0	0	0	0	0
1	1	0	1	1	1	0	0	0	0
1	1	1	0	0	0	0	0	0	0
1	1	1	0	0	1	0	0	0	0

4-bit 0 to 9 Gray code counter is a counter whose flip-flops go through a sequence of states as specified in Table 7.3.

Table 7.3 4-bit 0 to 9 Gray code counter (Excess-3 gray)

Binary code	Decimal equivalent
0010	0
0110	1
0111	2
0101	3
0100	4
1100	5
1101	6
1111	7
1110	8
1010	9

The third system that will be protected is the programmable logic array (PLA) for binary multiplier. In this example PLA controls the arithmetic circuit that multiplies two unsigned binary numbers and produces their binary products. Data processor for binary multiplier is presented in Figure 7.4 [23].

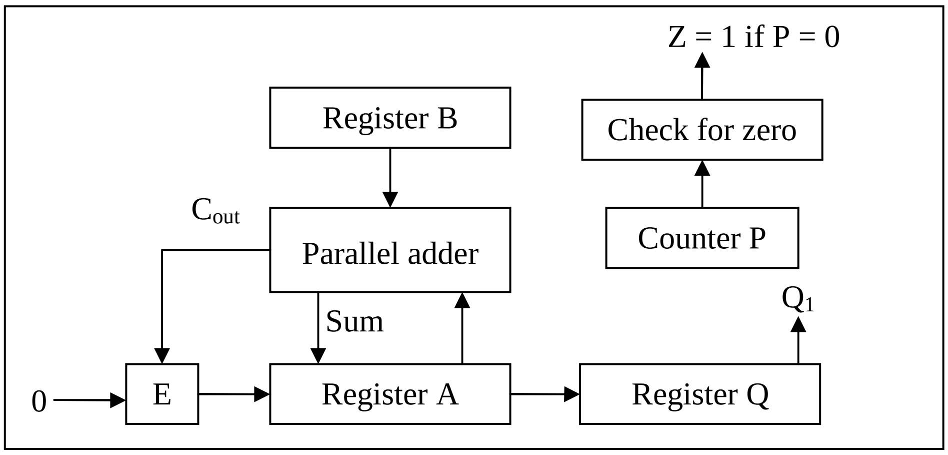


Figure 7.4 Data processor for binary multiplier

ASM chart of binary multiplier and PLA control block diagram are given in Figure 7.5 and Figure 7.6 respectively [23].

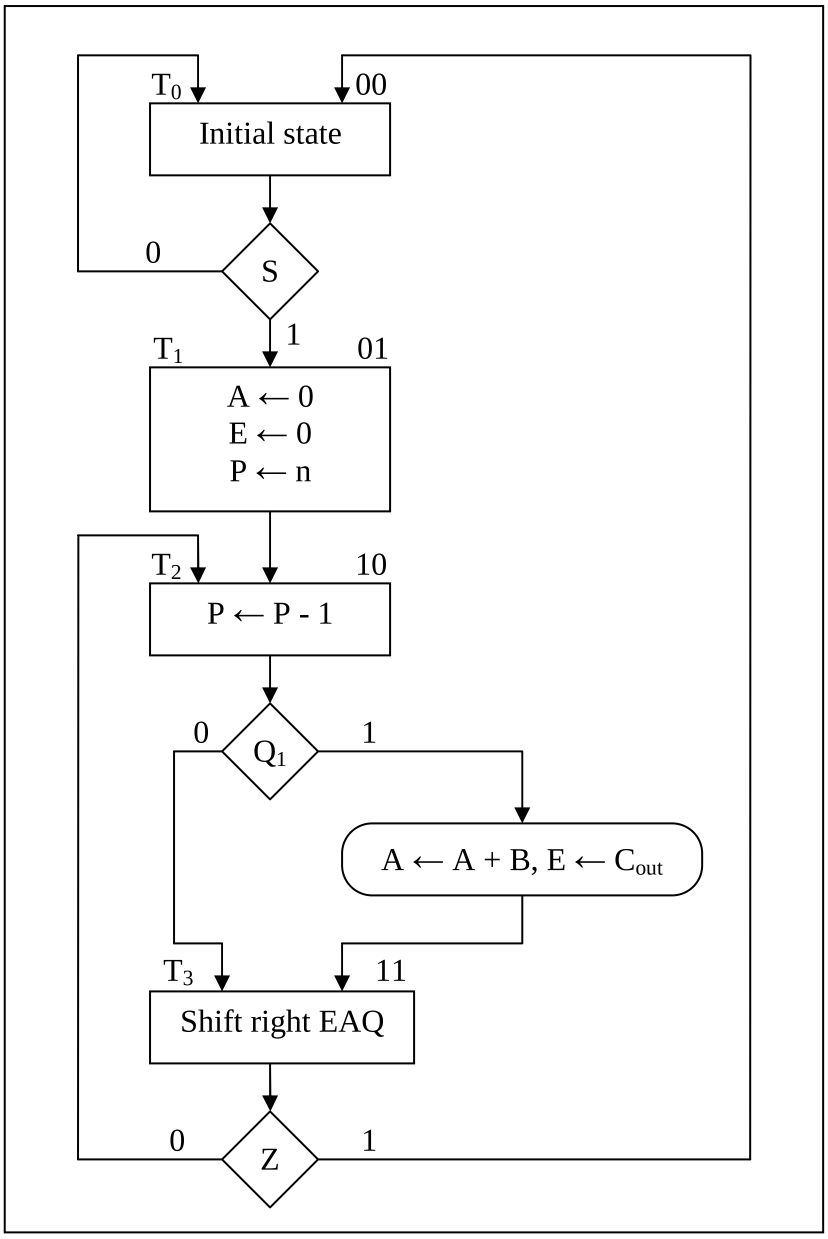


Figure 7.5 ASM chart for binary multiplier

PLA control has 3 inputs, S, Q_1 and Z, and 5 outputs, T_0, T_1, T_2, D and T_3. Output D represents the conditional operation Q_1T_2.

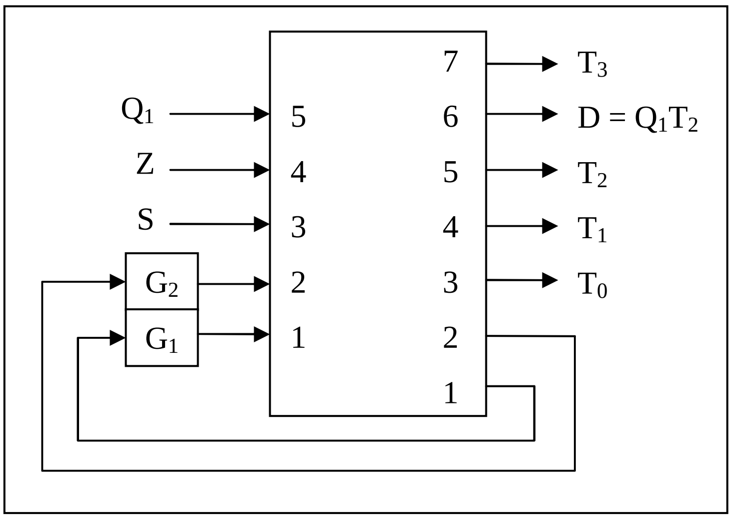


Figure 7.6 PLA control block diagram

Two self strings sets are constructed from the PLA control according to their bit-string representations. Bit-string representation of the first set is Present State/Inputs/Next State/Outputs and the second set is State/Outputs/Next State/Inputs. In contrast to previous systems, combining 2-bit present state, 3 input, 2-bit next state and 5 output data together creates 12-bit-strings that need to be protected. The number of self strings for PLA control is 32 and the state table for the control subsystem of the binary multiplier is shown in Table 7.4 [23].

Table 7.4 State table for PLA control

Present State		**Inputs**			**Next State**		**Outputs**				
G_1	**G_2**	**S**	**Z**	**Q_1**	**G_1**	**G_2**	**T_0**	**T_1**	**T_2**	**D**	**T_3**
0	0	0	X	X	0	0	1	0	0	0	0
0	0	1	X	X	0	1	1	0	0	0	0
0	1	X	X	X	1	0	0	1	0	0	0
1	0	X	X	0	1	1	0	0	1	0	0
1	0	X	X	1	1	1	0	0	1	1	0
1	1	X	0	X	1	0	0	0	0	0	1
1	1	X	1	X	0	0	0	0	0	0	1

The next self strings set is constructed from the control system of the ASM chart in Figure 7.7 [23].

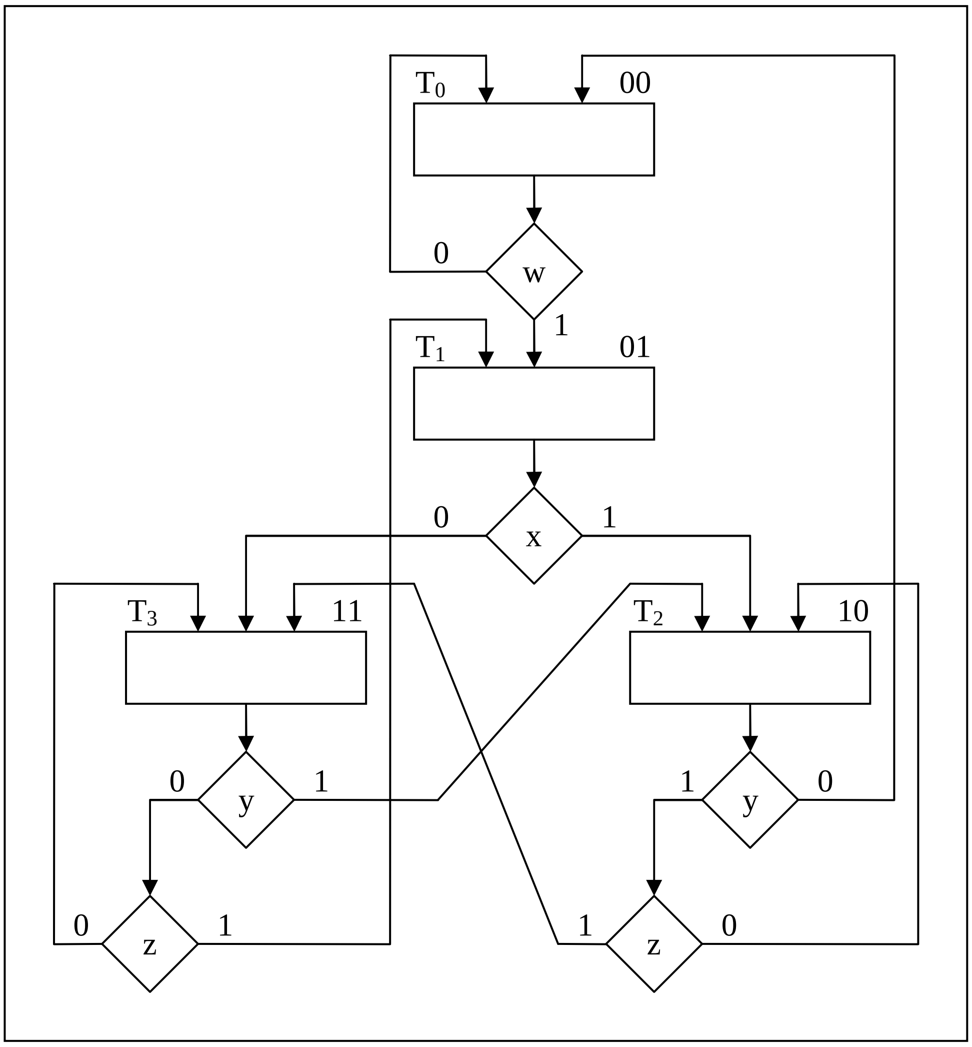


Figure 7.7 ASM chart of an example

Control system has 4 inputs, w, x, y and z, and 4 outputs, T_0, T_1, T_2 and T_3. The decision boxes specify the state transitions as a function of the four control inputs. The state table of the control circuit is presented in Table 7.5 [23].

Table 7.5 State table of control circuit

Present State		Inputs				Next State		Outputs			
G_1	**G_2**	**w**	**x**	**y**	**z**	**G_1**	**G_2**	**T_0**	**T_1**	**T_2**	**T_3**
0	0	0	X	X	X	0	0	1	0	0	0
0	0	1	X	X	X	0	1	1	0	0	0
0	1	X	1	X	X	1	0	0	1	0	0
0	1	X	0	X	X	1	1	0	1	0	0
1	0	X	X	0	X	0	0	0	0	1	0
1	0	X	X	1	0	1	0	0	0	1	0
1	0	X	X	1	1	1	1	0	0	1	0
1	1	X	X	0	1	0	1	0	0	0	1
1	1	X	X	1	X	1	0	0	0	0	1
1	1	X	X	0	0	1	1	0	0	0	1

From control circuit two self strings sets are constructed. Bit-string representations of these sets are Present State/Inputs/Next State/Outputs and Present State/Outputs/ Next State/Inputs. Combining 2-bit present state, 4 input, 2-bit next state and 4 output data together creates 12-bit strings that need to be protected. The number of self strings for the control circuit is 64.

Last self strings set is not constructed from the FSM. Small program fragment constructs these strings. Strings are constructed by the random number generator of the C++. Length of the strings and the number of strings in the self strings set are given to the program and program randomly generates the numbers and computes the binary equivalents (strings) and writes these into the text document.

7.3 Parameters

Simulation evaluates a number of parameters and these parameters are used to analyze the behaviour of the binary immune system for different examples. The explanation of these parameters are presented in Table 7.6. For each self strings set these parameters are evaluated and presented.

Table 7.6 Description of the parameters

Parameter	**Description**
Self templates	Templates that are constructed by self strings set for chosen match length
Holes	Strings that are constructed from only self templates (self strings are not included)
Positive tolerance conditions	The strings set that includes all and only self templates with minimum number of strings
Additional templates	Templates that can not be used to construct receptor
Detectable non-self strings	Strings that can be recognized by the complete receptor repertoire
Failure probability (P_f)	The probability that the system fails when it is confronted with the invalid operation (non-self string)
Lower bound for the failure probability	The failure probability of the system when complete receptor repertoire is used
Receptor templates	Templates that can be used to construct receptors
Complete receptor repertoire	The set of receptors that covers all the receptor templates
Minimum repertoire size	The minimum number of receptors in the complete receptor repertoire

7.4 Results

Self template matrix of the first self strings set which is generated from 4-bit 0 to 9 BCD counter is presented in Table 7.7. For this set, the length of the strings (ℓ) is 10 and for this particular example match length (c) is 5. Table 7.7 consists of 6 (ℓ-c+1) column and 32 (2^c) rows. Nodes that contain “1” represent self templates.

Table 7.7 Self template matrix of BCD counter (ℓ = 10, c = 5)

w	T[w,1]	T[w,2]	T[w,3]	T[w,4]	T[w,5]	T[w,6]
00000	1	1	1	1	1	1
00001	1	1	0	0	0	1
00010	1	1	1	1	1	1
00011	1	1	0	0	0	1
00100	1	1	1	1	1	1
00101	0	1	0	0	0	1
00110	0	1	1	1	0	1
00111	0	1	0	0	0	1
01000	1	1	1	1	1	1
01001	1	1	0	0	1	1
01010	1	0	1	0	1	0
01011	1	0	0	0	1	0
01100	1	0	1	1	1	0
01101	0	0	0	1	0	0
01110	0	0	1	0	0	0
01111	0	0	1	0	0	0
10000	1	1	1	1	1	1
10001	1	1	1	1	1	1
10010	1	1	1	0	0	1
10011	1	1	1	0	1	1
10100	1	1	0	1	0	1
10101	0	1	0	1	0	1
10110	0	1	0	0	0	1
10111	0	1	0	0	0	1
11000	1	1	0	1	1	1
11001	1	1	0	1	1	1
11010	1	0	0	0	1	0
11011	1	0	0	0	1	0
11100	1	0	0	1	0	0
11101	0	0	0	1	0	0
11110	0	0	0	1	0	0
11111	0	0	0	0	0	0

There are 105 self templates and 87 non-self templates. This self template set causes 184 holes, non-self strings that build up from the self template set.

Positive tolerance conditions which are constructed from the self templates presented in Table 7.7 are shown in Table 7.8.

Table 7.8 Positive tolerance conditions for BCD counter (ℓ = 10, c = 5)

	9	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	0	0	0	0	0	0	0	0	0	0
2	0	0	0	0	1	0	0	0	0	1
3	0	0	0	1	1	0	0	0	1	0
4	0	1	0	0	1	0	0	0	1	1
5	0	0	0	1	0	0	0	1	0	0
6	0	1	0	1	0	0	0	1	0	1
7	0	1	0	1	1	0	0	1	1	0
8	1	0	0	1	1	0	0	1	1	1
9	0	0	1	0	0	0	1	0	0	0
10	0	1	1	0	0	0	1	0	0	1
11	1	0	0	0	0	1	0	0	0	0
12	1	0	0	0	1	1	0	0	0	1
13	1	0	0	1	0	1	0	0	1	0
14	0	0	0	1	1	1	0	0	1	1
15	1	1	0	1	0	1	0	1	0	0
16	1	1	0	0	1	1	0	1	0	1
17	0	0	0	1	0	1	0	1	1	0
18	1	1	0	1	1	1	0	1	1	1
19	1	0	1	0	0	1	1	0	0	0
20	1	0	0	1	1	1	1	0	0	1
21	1	0	0	1	0	0	0	0	0	0
22	1	1	1	0	0	0	1	0	0	1
23	0	1	1	0	0	1	0	0	0	0
24	0	0	1	0	0	0	0	0	0	0
25	0	1	0	0	0	1	0	0	0	0
26	1	1	0	0	0	0	0	0	0	0

As it is seen from Table 7.8 there are 26 positive tolerance conditions and these positive tolerance conditions recognize 800 (81%) non-self strings. It means the failure probability when all 26 positive tolerance conditions are used, is 0,813.

Table 7.9 shows the modified self template matrix (added templates from parents and children check). Added templates, which are bold in Table 7.9, are the non-self templates but can not be used to construct receptors.

Table 7.9 Modified self template matrix for BCD counter (ℓ = 10, c = 5)

w	**T[w,1]**	**T[w,2]**	**T[w,3]**	**T[w,4]**	**T[w,5]**	**T[w,6]**
00000	1	1	1	1	1	1
00001	1	1	**1**	**1**	**1**	1
00010	1	1	1	1	1	1
00011	1	1	**1**	**1**	**1**	1
00100	1	1	1	1	1	1
00101	0	1	**1**	**1**	0	1
00110	0	1	1	1	0	1
00111	0	1	**1**	**1**	0	1
01000	1	1	1	1	1	1
01001	1	1	**1**	**1**	1	1
01010	1	0	1	0	1	0
01011	1	0	**1**	0	1	0
01100	1	0	1	1	1	0
01101	0	0	**1**	1	0	0
01110	0	0	1	0	0	0
01111	0	0	1	0	0	0
10000	1	1	1	1	1	1
10001	1	1	1	1	1	1
10010	1	1	1	0	**1**	1
10011	1	1	1	0	1	1
10100	1	1	**1**	1	**1**	1
10101	0	1	0	1	0	1
10110	0	1	**1**	0	0	1
10111	0	1	0	0	0	1
11000	1	1	**1**	1	1	1
11001	1	1	0	1	1	1
11010	1	0	**1**	0	1	0
11011	1	0	0	0	1	0
11100	1	0	**1**	1	0	0
11101	0	0	0	1	0	0
11110	0	0	**1**	1	0	0
11111	0	0	0	0	0	0

The “0”s in the modified template matrix represent the templates that can be used to construct receptors. There are 65 templates in the receptor template matrix which is illustrated in Table 7.10.

Table 7.10 Receptor template matrix for BCD counter (ℓ = 10, c = 5)

w	T[w,1]	T[w,2]	T[w,3]	T[w,4]	T[w,5]	T[w,6]
00000	0	0	0	0	0	0
00001	0	0	0	0	0	0
00010	0	0	0	0	0	0
00011	0	0	0	0	0	0
00100	0	0	0	0	0	0
00101	1	0	0	0	1	0
00110	1	0	0	0	1	0
00111	1	0	0	0	1	0
01000	0	0	0	0	0	0
01001	0	0	0	0	0	0
01010	0	1	0	1	0	1
01011	0	1	0	1	0	1
01100	0	1	0	0	0	1
01101	1	1	0	0	1	1
01110	1	1	0	1	1	1
01111	1	1	0	1	1	1
10000	0	0	0	0	0	0
10001	0	0	0	0	0	0
10010	0	0	0	1	0	0
10011	0	0	0	1	0	0
10100	0	0	0	0	0	0
10101	1	0	1	0	1	0
10110	1	0	0	1	1	0
10111	1	0	1	1	1	0
11000	0	0	0	0	0	0
11001	0	0	1	0	0	0
11010	0	1	0	1	0	1
11011	0	1	1	1	0	1
11100	0	1	0	0	1	1
11101	1	1	1	0	1	1
11110	1	1	0	0	1	1
11111	1	1	1	1	1	1

One possible receptor repertoire that contains minimum number of receptors is presented in Table 7.11.

Table 7.11 Receptor repertoire for BCD counter (ℓ = 10, c = 5)

	9	**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
1	0	0	1	1	0	0	1	0	1	0
2	0	0	1	0	1	0	1	0	1	1
3	0	1	1	1	0	0	1	1	0	0
4	0	1	1	0	1	0	1	1	0	1
5	1	0	1	1	0	0	1	1	1	0
6	0	0	1	1	1	0	1	1	1	1
7	0	0	1	1	0	1	1	0	1	0
8	1	0	1	0	1	1	1	0	1	1
9	1	1	1	1	0	1	1	1	0	0
10	1	1	1	0	1	1	1	1	0	1
11	1	0	1	1	0	1	1	1	1	0
12	1	0	1	1	1	1	1	1	1	1
13	0	1	1	1	1	0	1	0	1	1
14	1	1	1	1	1	1	1	1	1	1

Detectable non-self strings that these 14 receptors recognize is 664 (67%) and lower bound for the failure probability is 0.3252.

In Table 7.12, parameters of the 4-bit 0 to 9 BCD counter with bit-string representation of C/R/Previous/Current for match length $2 \leq c \leq 10$ are presented. For match length $c < 4$ no unique receptors (tolerance conditions) are possible as a match occurs between at least one self string and any receptor. Also for $c = 10$ every receptor matches a unique single non-self string. This self strings set is chosen as a verification of the simulation. Same results at Table 7.12 were presented in [17] and [10].

Table 7.12 Parameters of the 4-bit 0 to 9 BCD counter (C/R/Previous/Current)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Self Templates	34	53	74	105	122	111	93	70	40
Holes	536	344	320	184	80	39	20	10	0
Positive Tol. Conditions	7 45%	13 65%	13 67%	26 81%	40 91%	40 96%	40 97%	40 98%	40 100%
Additional Templates	2	11	9	22	62	37	24	10	0
Detectable Non-Self Strings	0	0	584 59%	664 67%	784 80%	913 93%	932 95%	954 97%	984 100%
P_f (Lower Bound)	-	-	0.406	0.325	0.203	0.072	0.052	0.030	0
Receptor Temp.	-	-	29	65	136	364	651	944	984
Minimum Repertoire Size	-	-	6	14	42	103	222	472	984

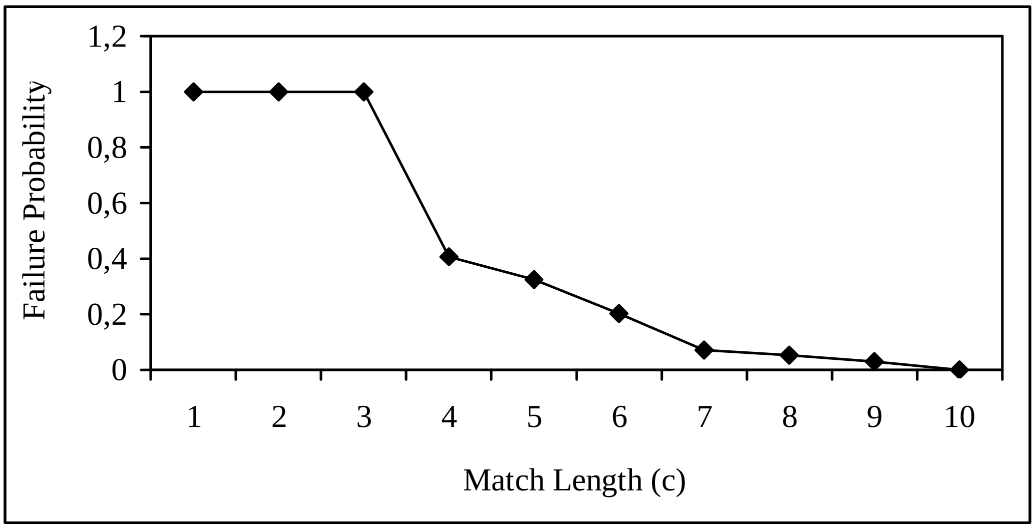


Figure 7.8 Failure probability vs. match length
for 4-bit 0 to 9 BCD counter (C/R/Previous/Current)

The failure probability versus match length, when complete receptor repertoire is used, is given in Figure 7.8. Lower bound for the failure probability is computed as the ratio of unrecognized non-self strings to all non-self strings.

In Table 7.13, again the parameters of the 4-bit 0 to 9 BCD counter is presented. In this example, bit-string representation of the strings is C/R/Current/ Previous. In Figure 7.9, the lower bound for failure probability versus match length is shown.

Table 7.13 Parameters of the 4-bit 0 to 9 BCD counter (C/R/Current/Previous)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Self Templates	33	48	64	81	90	84	74	60	40
Holes	440	216	176	72	48	22	16	10	0
Positive Tol. Conditions	6 55%	10 78%	11 82%	21 92%	25 95%	28 97%	29 98%	30 98%	40 100%
Additional Templates	3	16	7	27	27	27	17	0	0
Detectable Non-Self Strings	0	0	712 72%	768 78%	920 93%	920 93%	954 97%	974 99%	984 100%
P_f (Lower Bound)	-	-	0.276	0.219	0.065	0.065	0.030	0.010	0
Receptor Temp.	-	-	41	84	203	401	677	964	984
Minimum Repertoire Size	-	-	6	20	49	109	228	482	984

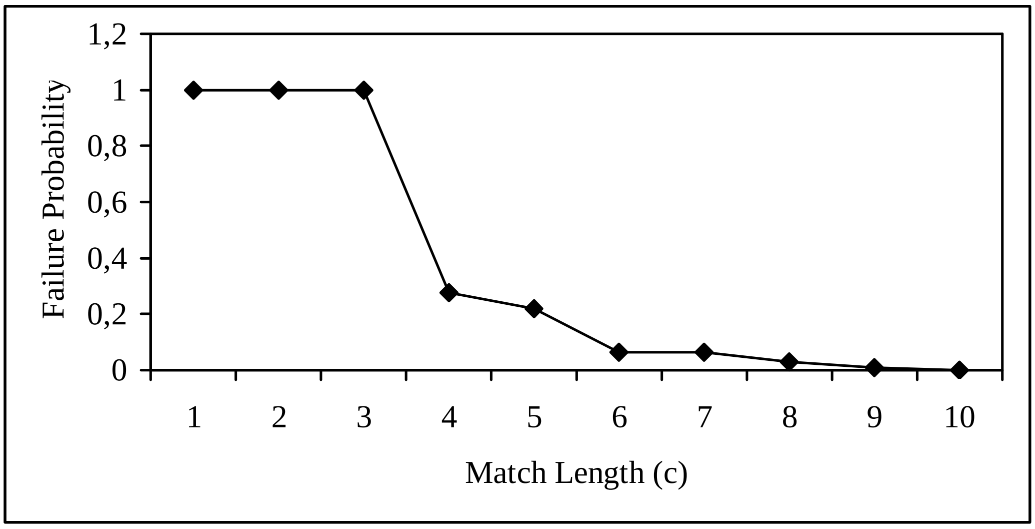


Figure 7.9 Failure probability vs. match length for 4-bit 0 to 9 BCD counter (C/R/Current/Previous)

The parameters in Table 7.14 are again computed from 4-bit 0 to 9 BCD counter but bit-string representation is R/Previous/C/Current. For c ≤ 4, it is not possible to construct any receptor.

Table 7.14 Parameters of the 4-bit 0 to 9 BCD counter (R/Previous/C/Current)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Self Templates	34	54	80	120	138	130	110	78	40
Holes	536	360	324	308	62	36	14	4	0
Positive Tol. Conditions	6 45%	13 63%	18 67%	22 68%	40 93%	40 96%	40 98%	40 99%	40 100%
Additional Templates	2	10	32	2	51	23	7	2	0
Detectable Non-Self Strings	0	0	0	676 68%	762 77%	912 93%	957 97%	976 99%	984 100%
P_f (Lower Bound)	-	-	-	0.313	0.225	0.073	0.027	0.008	0
Receptor Temp.	-	-	-	70	131	359	651	944	984
Minimum Repertoire Size	-	-	-	13	32	95	219	472	984

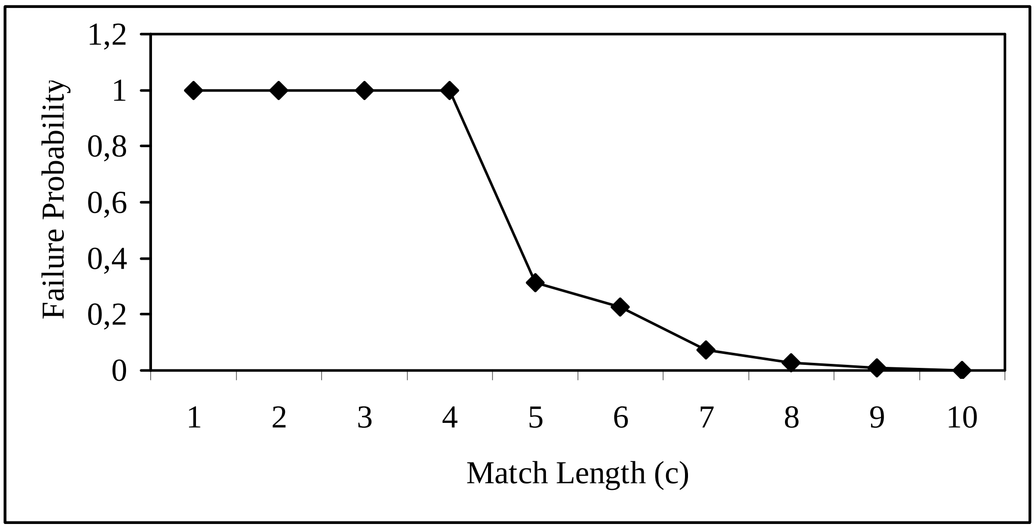


Figure 7.10 Failure probability vs. match length
for 4-bit 0 to 9 BCD counter (R/Current/C/Previous)

In Table 7.15, the parameters of the 4-bit 0 to 9 Gray code counter (Excess-3 gray) are indicated. In Figure 7.11, lower bounds for failure probabilities of the Gray code counter against match length illustrated.

Table 7.15 Parameters of the 4-bit 0 to 9 Gray code counter (C/R/Previous/Current)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**
Self Templates	34	53	82	106	121	110	94	70	40
Holes	536	344	336	128	50	17	12	8	0
Positive Tol. Conditions	6 45%	12 65%	20 65%	27 86%	40 94%	40 98%	40 98%	40 99%	40 100%
Additional Templates	2	11	30	35	72	36	20	10	0
Detectable Non-Self Strings	0	0	0	592 60%	656 67%	894 91%	966 98%	956 97%	984 100%
P_f (Lower Bound)	-	-	-	0.398	0.333	0.091	0.018	0.028	0
Receptor Temp.	-	-	-	51	127	366	654	944	984
Minimum Repertoire Size	-	-	-	13	34	101	226	472	984

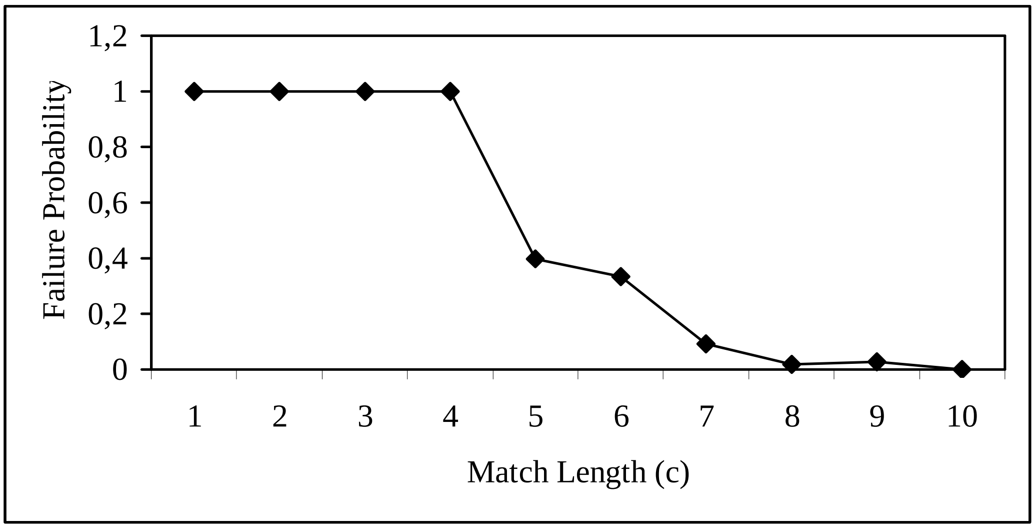


Figure 7.11 Failure probability vs. match length for 4-bit 0 to 9 Gray code counter (C/R/Current/Previous)

Next system is PLA control for the binary multiplier. Bit-string representation of the strings is Preset State/Inputs/Next State/Outputs. The parameters of the PLA control and the lower bounds of the failure probability of the PLA control for various match lengths are listed in Table 7.16 and Figure 7.12 respectively.

Table 7.16 Parameters of PLA control (Present State/Inputs/Next State/Outputs)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Self Temp.	40	62	95	133	133	131	128	116	96	64	32
Holes	1504	608	320	144	20	8	8	0	0	0	0
Positive Tol. Conditions	5 62%	8 85%	16 92%	32 96%	32 99%	32 99%	32 99%	32 100%	32 100%	32 100%	32 100%
Additional Templates	4	18	49	123	56	57	28	12	0	0	0
Detectable Non-Self Strings	0	0	0	0	3970 98%	3880 95%	3988 98%	4040 99%	4064 100%	4064 100%	4064 100%
P_f	-	-	-	-	0.023	0.045	0.018	0.005	0	0	0
Receptor Templates	-	-	-	-	259	580	1124	1920	2976	4032	4064
Minimum Repert Size	-	-	-	-	55	100	228	480	992	2016	4064

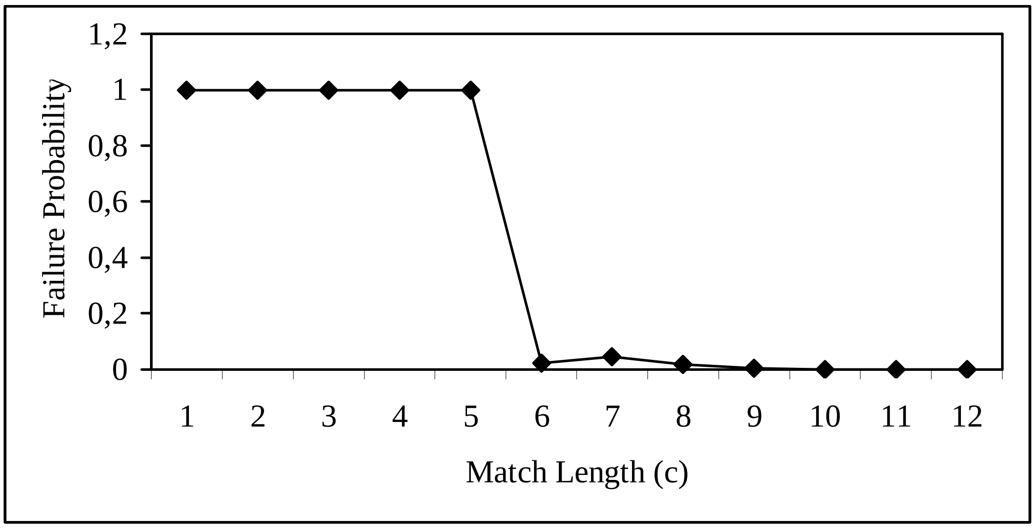


Figure 7.12 Failure probability vs. match length
for PLA control (Present State/S/Z/Q_1/Next State/T_0/T_1/T_2/D/T_3)

Parameters and lower bound for failure probability versus match length graphic are presented in Table 7.17 and Figure 7.13 for bit-string representation of Present State/Outputs/Next State/Inputs.

Table 7.17 Parameters of PLA control (Present State/Outputs/Next State/Inputs)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Self Temp.	42	54	91	105	111	107	103	94	79	58	32
Holes	2272	908	412	122	48	18	13	3	0	0	0
Positive Tol. Conditions	7 44%	9 77%	19 89%	25 96%	27 98%	29 99%	31 99%	31 %99	32 %100	32 100%	32 100%
Additional Templates	2	15	53	36	34	41	29	22	13	6	0
Detectable Non-Self Strings	-	-	-	3677 90%	3938 97%	3988 98%	4037 99%	4013 99%	4044 99%	4052 99%	4064 100%
P_f (Lower Bound)	-	-	-	0.095	0.031	0.018	0.006	0.012	0.004	0.002	0
Receptor Templates	-	-	-	115	303	620	1148	1932	2980	4032	4064
Minimum Rep. Size	-	-	-	19	52	113	241	486	996	2016	4064

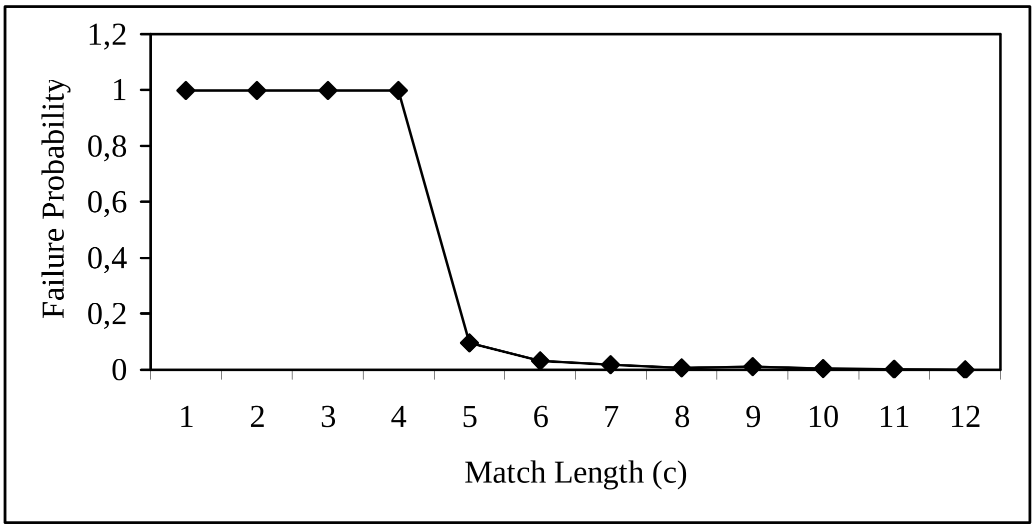


Figure 7.13 Failure probability vs. match length
for PLA control (Present State/T_0/T_1/T_2/D/T_3/Next State/S/Z/Q_1)

The next example is the control circuit of the ASM which is presented in Figure 7.7. In Table 7.18, the parameters of this control circuit are given. In Figure 7.14, the lower bound for failure probability vs. match length is illustrated for control circuit.

Table 7.18 Parameters of control circuit (Present State/Inputs/Next State/Outputs)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Self Temp.	41	67	109	174	230	243	244	232	192	128	64
Holes	1984	960	576	408	152	68	40	26	0	0	0
Positive Tol. Conditions	4 50%	8 76%	16 85%	32 89%	64 96%	64 98%	64 99%	64 %99	64 100%	64 100%	64 100%
Additional Templates	3	13	35	82	218	96	69	24	0	0	0
Detectable Non-Self Strings	0	0	0	0	0	3724 92%	3810 94%	3932 98%	4032 100%	4032 100%	4032 100%
P_f	-	-	-	-	-	0.076	0.055	0.024	0	0	0
Receptor Templates	-	-	-	-	-	429	967	1792	2880	3968	4032
Minimum Rep. Size	-	-	-	-	-	92	201	448	960	1984	4032

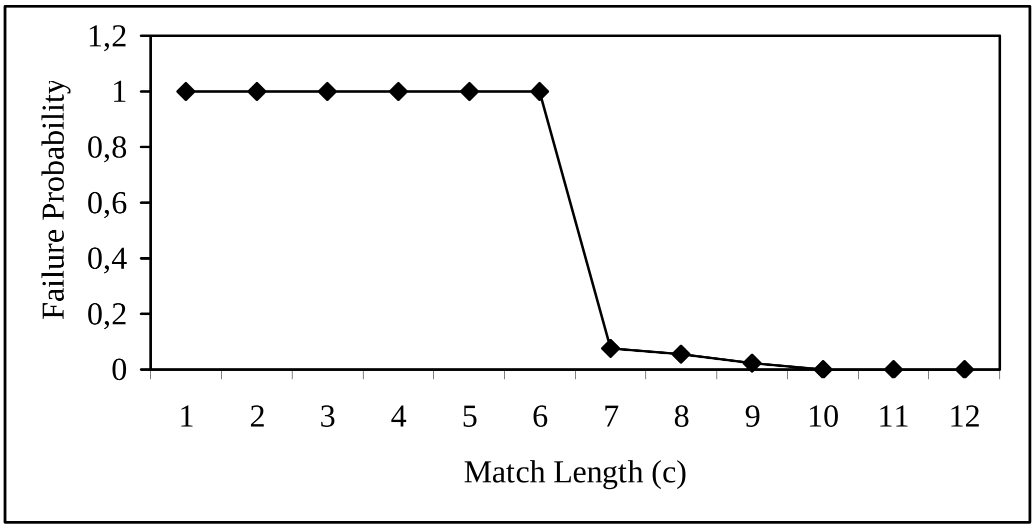


Figure 7.14 Failure probability vs. match length
for control circuit (Present State/x/y/z/w/Next State/T_0/T_1/T_2/T_3)

The parameters and the same figure are presented in Table 7.19 and Figure 7.15 when bit-string representation is Present State/Outputs/Next State/Inputs.

Table 7.19 Parameters of control circuit (Present State/Outputs/Next State/Inputs)

Match Length	**2**	**3**	**4**	**5**	**6**	**7**	**8**	**9**	**10**	**11**	**12**
Self Temp.	39	58	83	116	142	154	157	148	132	104	64
Holes	1280	568	72	54	22	4	0	0	0	0	0
Positive Tol. Conditions	5 68%	8 85%	16 98%	30 98%	46 99%	56 99%	64 100%	64 100%	64 100%	64 100%	64 100%
Additional Templates	5	22	61	59	94	103	101	80	52	24	0
Detectable Non-Self Strings	-	-	-	3798 94%	3972 99%	3700 92%	3872 96%	3864 96%	3924 97%	3984 99%	4032 100%
P_f	-	-	-	0.058	0.014	0.082	0.039	0.041	0.026	0.011	0
Receptor Templates	-	-	-	81	212	511	1022	1820	2888	3968	4032
Minimum Rep. Size	-	-	-	27	59	105	226	468	968	1984	4032

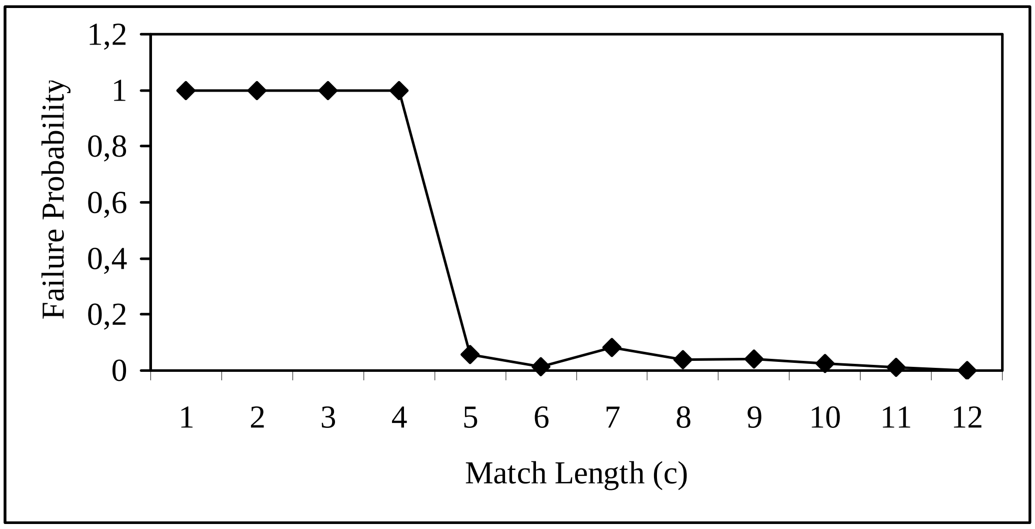


Figure 7.15 Failure probability vs. match length
for control circuit (Present State/T_0/T_1/T_2/T_3/Next State/x/y/z/w)

7.5 Analysis

First three self strings sets in the simulation are constructed from 0 to 9 BCD counter and the length of the strings is 10. The only difference between these sets is the bit-string representations of the strings. The next example is 0 to 9 Gray code (Excess-3) counter and length of the strings is 10 as well. The bit-string representation of the self strings set of the Gray code counter is the same as the first self strings set of the BCD counter. Both 0 to 9 BCD counter and Gray code counter has 40 self strings therefore 984 non-self strings to be recognized. In Figure 7.16, the number of detectable non-self strings versus match length is presented for these four self strings sets.

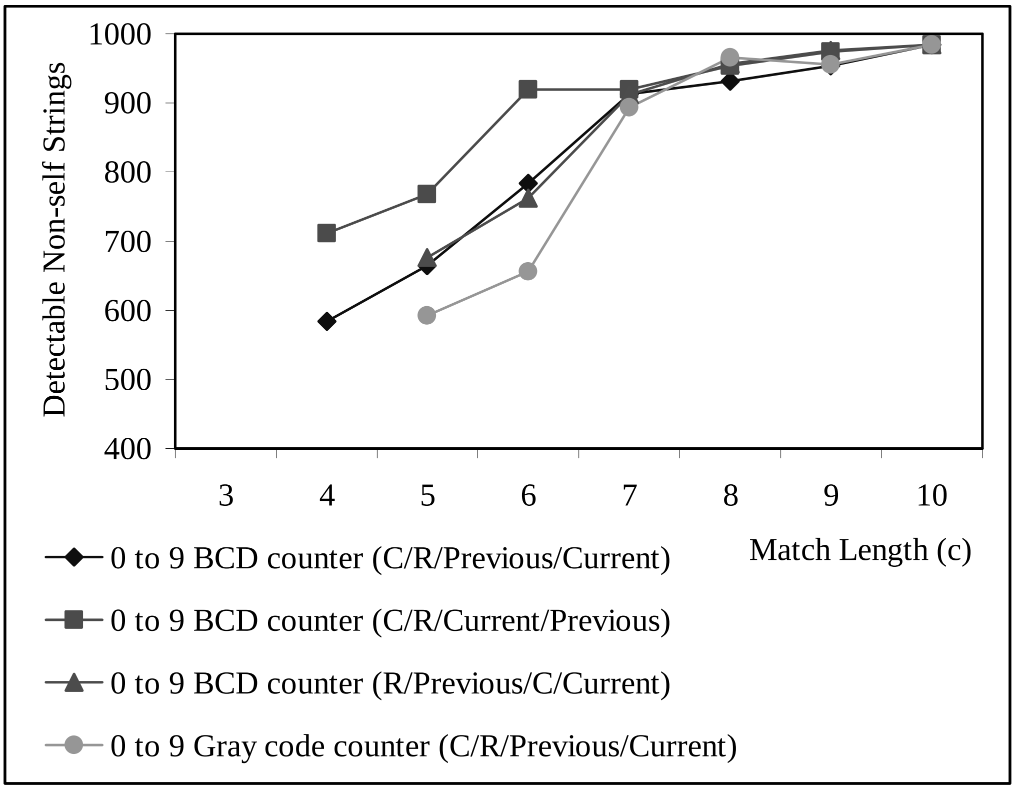


Figure 7.16 The number of detectable non-self strings vs. match length for self strings sets of counters

It can be seen from the figure that for 0 to 9 BCD counter for the bit-string representation of R/Previous/C/Current, unlike other bit-string representations of the same counter, for c = 4, no unique receptors are possible as a match occurs between at least one self string and any receptor. Because only difference between these sets are the bit-string representations, it can be said that the number of detectable non-self strings and also lower bound for failure probability strongly depends on the structure of the self strings set.

The length of the self strings for the PLA control and control circuit of example ASM is 12. Therefore 4096 different strings can be constructed for these systems. The number of self strings are 32 and 64 for PLA control and control circuit of the last example respectively. Number of detectable non-self strings against match length figure, Figure 7.17, shows that the behaviour of these systems for different match lengths are very close to each other, but the important difference between these two systems is, for c = 6, no unique receptors are possible as a match occurs between at least one self string and any receptor for the last example.

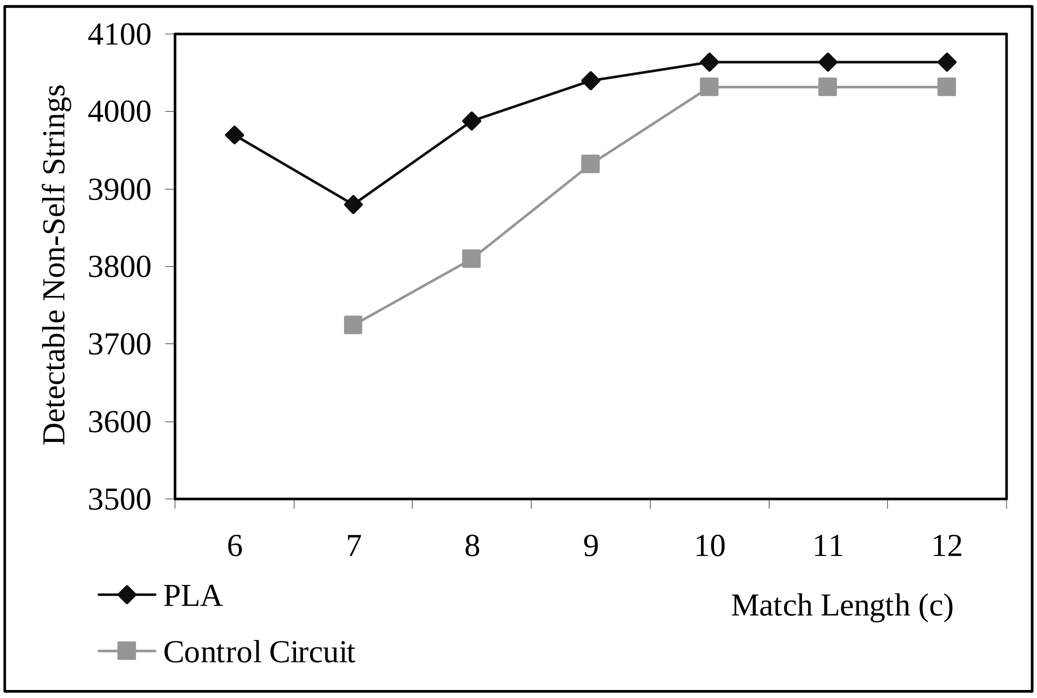


Figure 7.17 The number of detectable non-self strings vs. match length for PLA and control circuit

It is also interesting to state that, commonly when match length increases number of detectable non-self strings increases, in other words, failure probability decreases but there are some exceptions. For example, number of detectable non-self strings for match length 6 is bigger than that of 7 for PLA. Also the same exception is occurred in between match lengths 8 and 9 for Gray code counter.

When match length increases the number of non-self strings detected by the whole repertoire increases. The number of strings recognized by a single receptor decreases approximately twice if the match length increases by one, that is when k increases, the number of receptors increases approximately as the power of two. Figure 7.18 shows how the number of strings detected by a single receptor varies against match lengths. Because this number is only related with the match length and the length of the strings, results shown in Figure 7.18 are the same for all the string sets in these lengths. The values presented in Figure 7.18 can be computed by the equation (3).

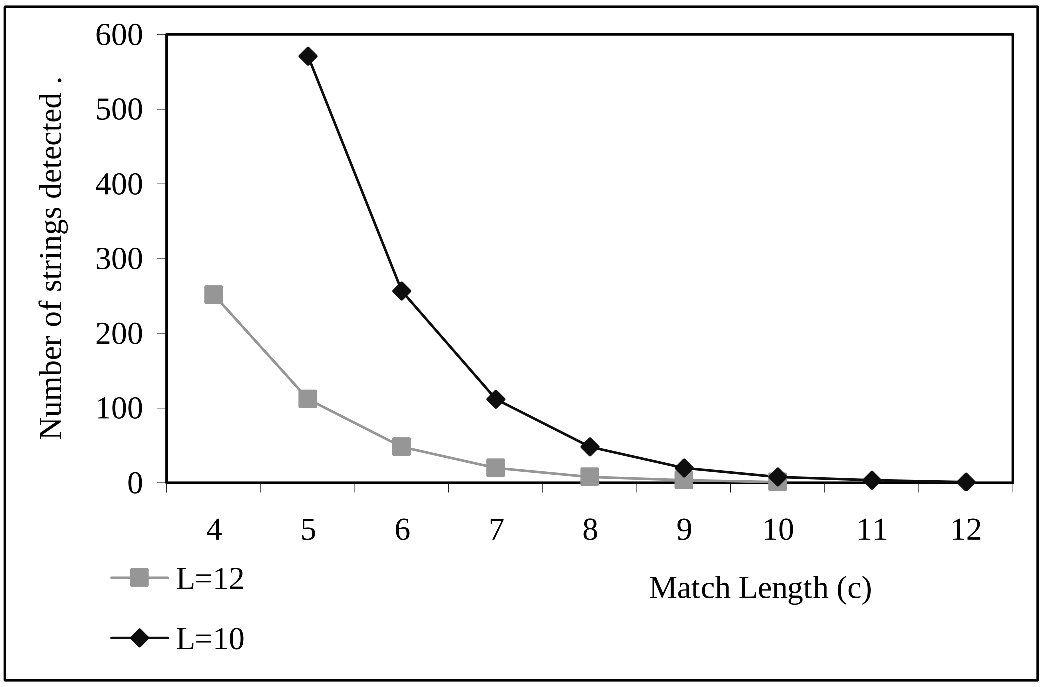


Figure 7.18 Number of strings detected by a single receptor vs. match length

Although for the same match length and the same string length, the number of strings recognized by the single receptor is the same and not related with the self strings set, the number of strings recognized by more than one receptor (receptor ensemble) is related to the self strings set and chosen receptors. Figure 7.19 presents the number of strings detected by five receptors for different match lengths. Greedy algorithm is used to choose the receptors from repertoire.

It can be seen from the figure that when match length is exceed ℓ/2 (ℓ = 10), the number of strings detected by receptors are equal for each set. The reason of this is explained in section 5.4.1. For c ≤ ℓ/2, the number of strings detected by the receptors is directly related with the structure of the self strings sets. The decrease of the number of detected strings by the receptors is related with the number of strings detected by a single receptor when match length increases.

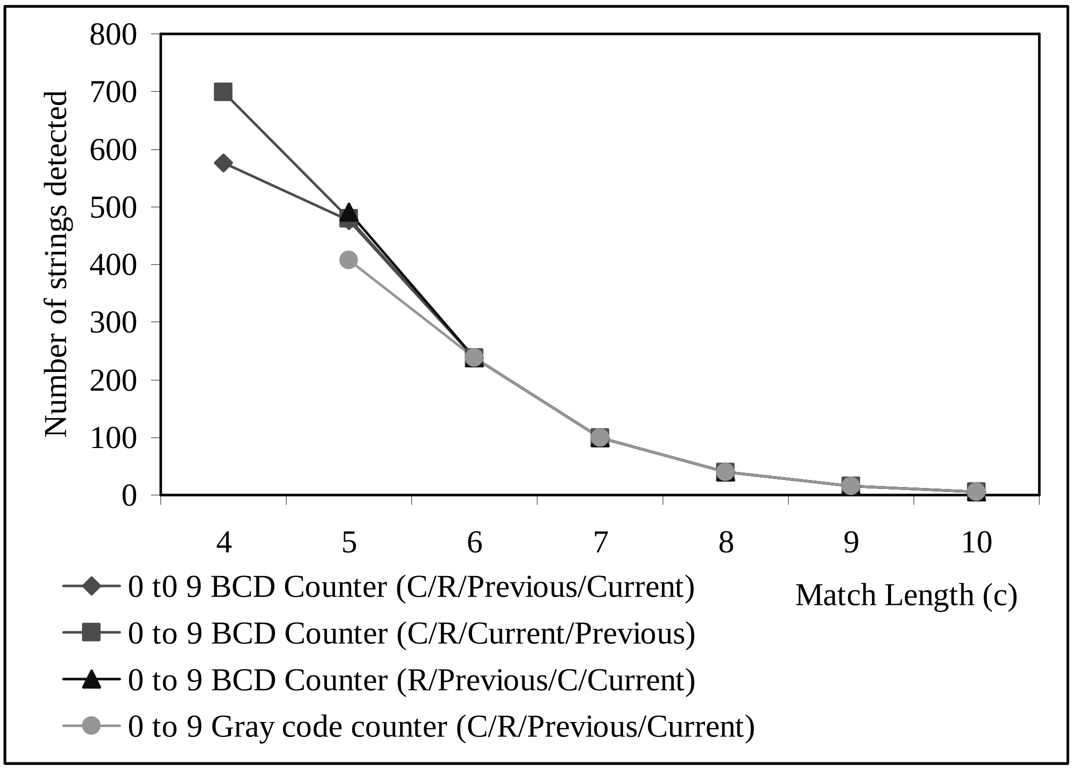


Figure 7.19 Number of strings detected by five receptors for counters

In first four self strings sets, 4% of the strings are self. This number for the PLA control is 0,7 % and 1,5 % for the control circuit. The results show that when the number of strings in the self strings set is high, the failure probability of the system decreases slowly when match length increases. On the other hand, if the number of strings in the self strings set is low, the failure probability of the system decreases rapidly when match length increases. Figure 7.20 shows the behaviour of the failure probabilities of first self strings set of the BCD counter and PLA control. It is important to remember that the length of the strings for BCD counter and the PLA control is different (10 and 12 respectively).

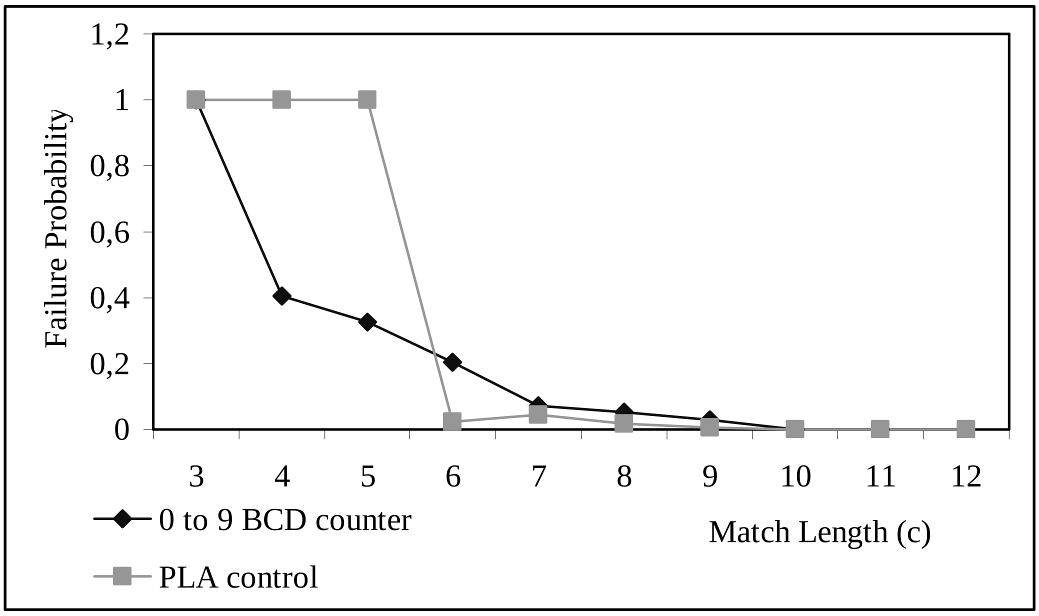


Figure 7.20 Failure probability vs. match length
for BCD counter and PLA control

To understand the behaviour of the BIS for different number of strings in the self strings sets, randomly generated strings are used. Four different strings lengths which are 10, 12, 15 and 18 are employed. For each string length, the number of strings in the self strings sets are listed in Table 7.20.

Table 7.20 Number of self strings for different strings lengths

Length	1%	2%	5%	10%	20%
2^{10}	10	20	51	102	205
2^{12}	41	82	205	410	819
2^{15}	328	655	1638	3277	6554
2^{18}	1621	5243	13107	26214	52429

In Figure 7.21, 7.22, 7.23 and 7.24, the lower bounds of the failure probabilities vs. match lengths are presented for each number of self strings and string lengths presented in Table 7.20.

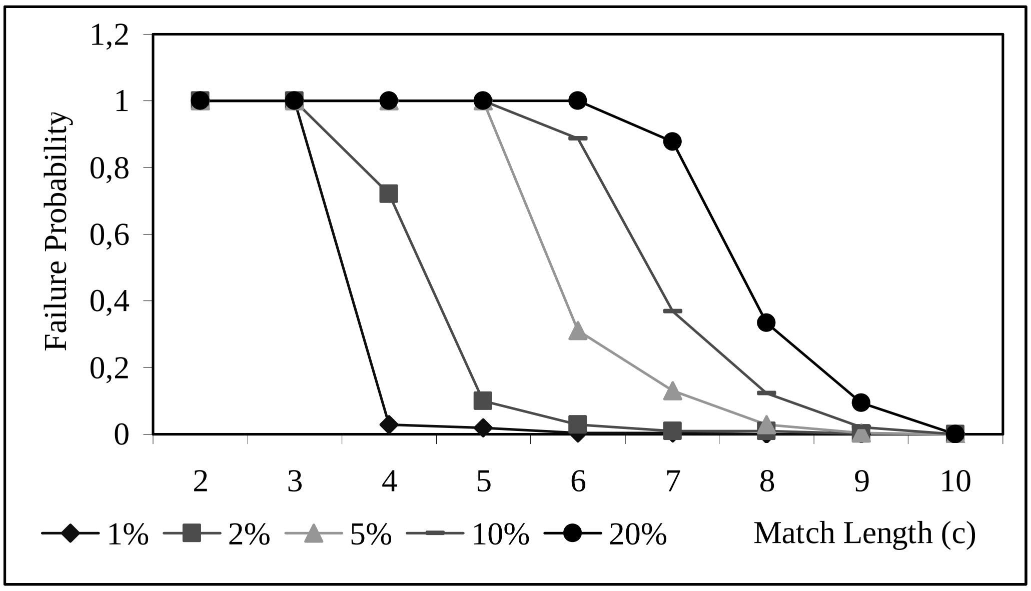


Figure 7.21 Failure probability vs. match length for ℓ=10

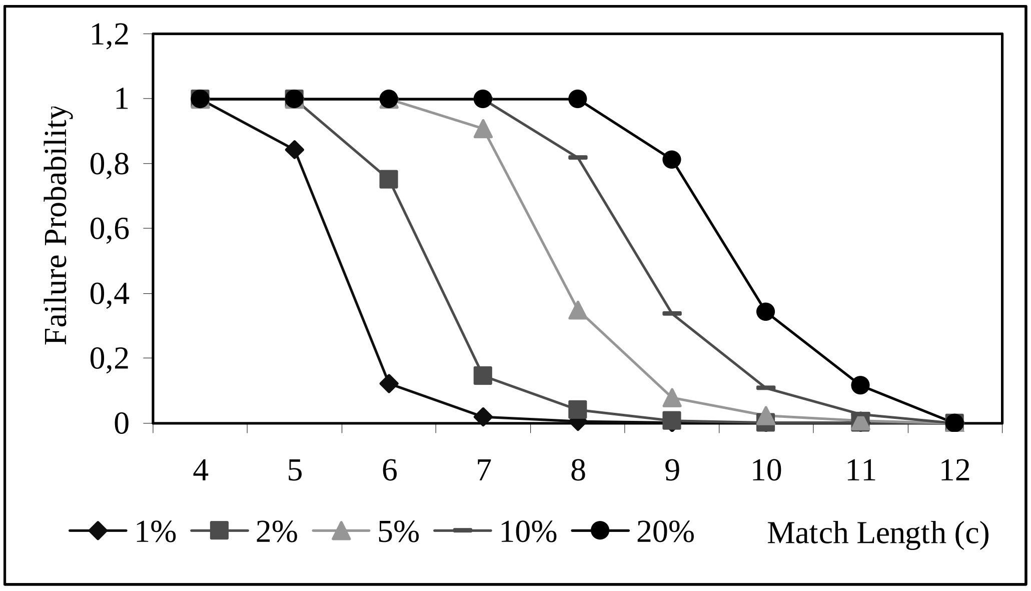


Figure 7.22 Failure probability vs. match length for ℓ=12

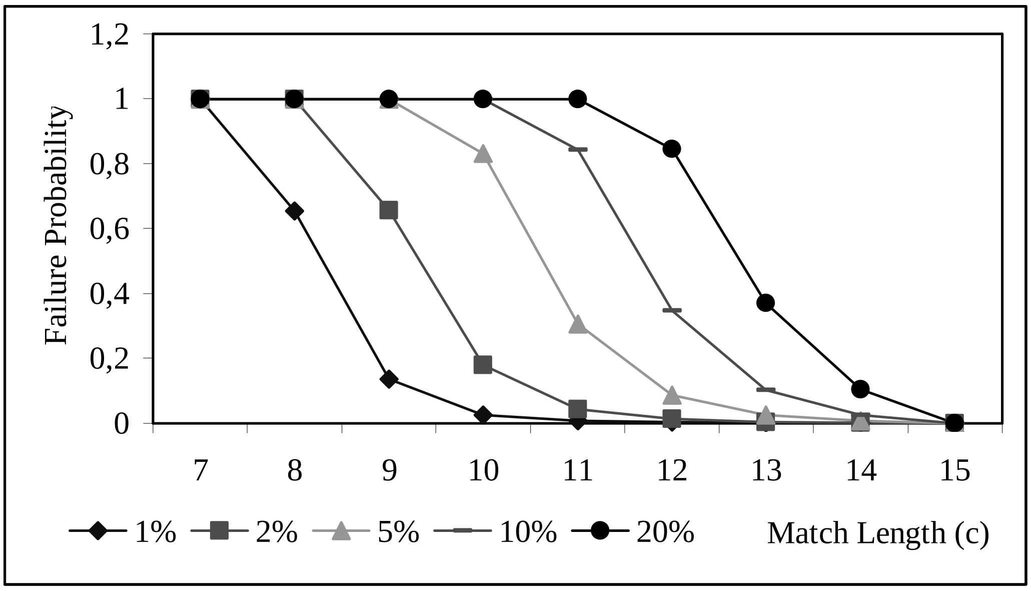


Figure 7.23 Failure probability vs. match length for ℓ=15

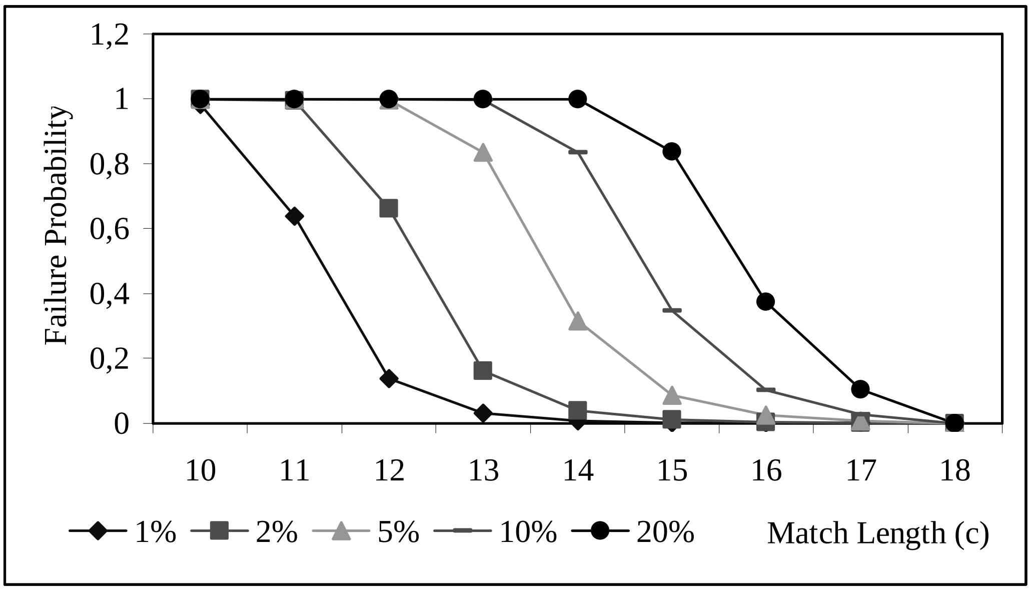


Figure 7.24 Failure probability vs. match length for ℓ=18

From the figures of the randomly generated self strings sets, it can be said that, the failure probability of the system approaches to "0" rapidly if the number of strings in the self strings set is low.

7.5.1 Positive Tolerance Conditions

Positive tolerance conditions generally give better solutions than receptors. Only exceptions are the average number of strings recognized by a single receptor is usually higher than that of positive tolerance condition when c is close to ℓ/2. It is important to state that even in the conditions that previous statement is valid, when complete repertoire is used, positive tolerance conditions recognize more non-self strings than receptors that is because the number of strings in the positive tolerance conditions is more than that of receptor repertoire. Figure 7.21 displays difference between the average number of strings recognized by the single receptor in the receptor repertoire and the average number of strings recognized by the single positive tolerance condition. The results of 0 to 9 BCD counter with bit-string representation of C/R/Previous/Current are used.

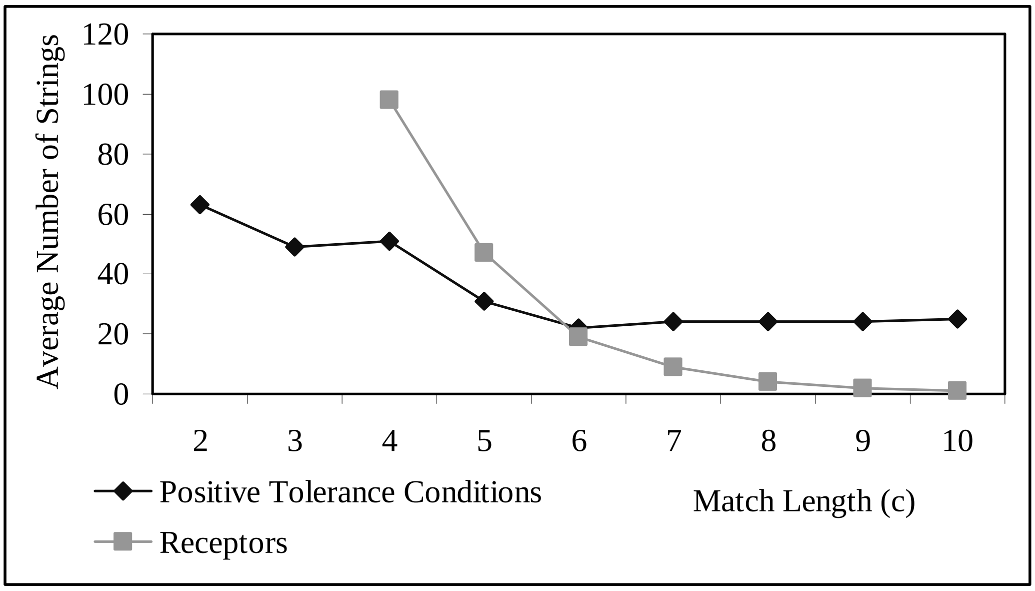


Figure 7.25 Average number of strings recognized by a single positive tolerance condition and a single receptor

Especially for small values of match length, c << ℓ/2 or c close to 1, receptors can not be constructed due to the excessive number of holes and the additional templates. In these cases positive tolerance conditions are the only solution.

CHAPTER 8

CONCLUSION

This study has demonstrated a immunologically inspired approach to hardware fault detection and discussed the architecture for a hardware immune system to detect faults in systems. The acquired immune response in the human immune system is learned through a process of centralized maturation to create a collection of antibodies able to detect the invasion of non-self into the body. This analogy has been applied to the field of electronic hardware error detection to provide FSMs using a generic immunization procedure.

An immunization cycle has been developed that integrates with a typical hardware development cycle to permit any finite-state-based system to be immunized in a methodical way. The system is analyzed, self strings are gathered and tolerance conditions are generated. The match length is chosen to make optimum use of the available tolerance condition storage space. This is carried out currently by hand, although future automation would be straightforward. The architecture also permits a trade-off between the storage space and failure probability, ensuring that the most effective tolerance conditions are always stored first.

The hardware immune system currently goes some way to achieving three of the five original analogies between the human immune system and hardware fault tolerance discussed in Chapter 4.

1. The operational hardware immune system functions continuously and autonomously and is designed to allow full implementation in hardware.

This is facilitated by the simple (compared to tolerance condition generation) search and detection process created through the use of a CAM.

2. The immunotronic error detection mechanisms are trained to differentiate between faulty and fault free transitions. The hardware immune system possesses memory to store the set of tolerance conditions that perform this operation.

3. Detection of invalid conditions is imperfect.

In this study, a new set of tolerance conditions which are called positive tolerance conditions are proposed. Positive tolerance conditions gives better results than receptors if storage requirements to not impose any restriction. As stated in Chapter 7, average number of strings recognized by a single receptor is usually higher than that of positive tolerance condition when match length, c, is close to $\ell/2$.

For c close to ℓ, the complete receptor repertoire constructed by the simulation contains large number of receptors. It is important to note that, there is no meaning to create receptor ensemble that contains more strings than the self strings set. Because, when the number of receptors exceeds the number of self strings set, simply storing all self strings in the memory and using these as a look up table which is actually constructing positive tolerance conditions when match length is equal the length of the strings, guarantees 100% reliable operation.

Complexity of the construction of both positive tolerance conditions and receptors are of order $O(\ell \bullet (\ell - c) \bullet 2^c)$, where ℓ is the length of the strings and c is the match length (threshold value). Because both positive tolerance conditions and receptors are constructed with same algorithm. Only difference between these two is positive tolerance conditions are constructed from self template matrix but the receptors are constructed from receptor template matrix. Receptor template matrix is modified (child and parent check, explained in Section 5.6) form of the self template matrix and the modification (child and parent check) algorithm is of order $O((\ell - c) \bullet 2^c)$.

Current work has not investigated the possibility of further learning while the system is in operation. Such operation would be more truly representative of a real immune system.

REFERENCES

[1] P. Lee and T. Anderson, "Fault-Tolerance: Principle and Practice", Springer-Verlag, Wien-New York, 1990.

[2] J. Von Neumann, "Probabilistic Logic and The Synthesis of Reliable Organisms from Unreliable Components", Automata Studies, C. Shannon and J. McCarthy Eds. Annals of Math Studies, Num 34, Princeton Univ. Press, pp. 43-98. 1956.

[3] S. Forrest, S. A. Hofmeyr, A. Somayaji and T. A. Longstaff, "A Sense of Self for Unix Processes", in Proceedings of the 1996 IEEE Symposium on Computer Security and Privacy, pp. 120-128, May 1996.

[4] S. Forrest, A. S. Perelson, L. Allen and R. Cherukuri, "Self-Nonself Discrimination in a Computer", in Proceedings of the 1994 IEEE Symposium on Research in Security and Privacy, Los Alamitos, CA, IEEE Computer Society Press, pp. 202-212, 1994.

[5] J. O. Kephart, "A Biologically Inspired Immune System for Computers", in Artificial Life IV, Proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems, R. A. Brooks and P. Maes Eds. Cambridge, MA: MIT Press, pp. 130-139, 1994.

[6] D. Dasgupta and S. Forrest, "An Anomaly Detection Algorithm Inspired By The Immune System", in Artificial Immune Systems and Their Applications, D. Dasgupta Eds. Berlin, Germany, Springer-Verlag, pp. 262-277, 1998.

[7] A. Ishiguro, Y. Watanabe and Y. Uchikawa, “Fault Diagnosis of Plant Systems Using Immune Networks”, in Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems, pp. 34-42, Oct. 1994.

[8] A. Ishiguro, T. Kondo, Y. Watanabe and Y. Uchikawa, “Immunoid: An Immunological Approach to Decentralized Behavior Arbitration of Autonomous Mobile Robots”, in Parallel Problem Solving from Nature IV, H. M.Voight Eds. Springer-Verlag, Vol. 1141, Lecture Notes in Computer Science, pp. 666-675, 1996.

[9] S. Xanthakis, S. Karapoulios, R. Pajot and A. Rozz, “Immune System and Fault Tolerant Computing”, in Artificial Evolution, J. M. Alliot Eds. Springer-Verlag, Vol. 1063, Lecture Notes in Computer Science, pp. 181-197, 1996.

[10] D. W. Bradley and A. M. Tyrell, “The Architecture for a Hardware Immune System”, Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware, Long Beach, California, USA, pp. 193-200, 2001.

[11] A. Avizienis, “Towards systematic Design of Fault-Tolerant Systems”, IEEE Computer, 30(4):51-58, April 1997.

[12] G. C. Kassianos, “Immunization, Childhood and Tarvel Health”, 4th Edition, Blackwell Science, USA, 2001.

[13] T. G. Parslow, D.P.Stites, A. I. Terr and J. B. Imboden, “Medical Immunology”, 10th Edition, Lange Medical Books, McGraw-Hill, USA, 2001.

[14] S. Hofmeyr, “An Overview of the Immune System”
http://www.cs.umn.edu/˜immsec/html-imm/introduction.html, 2004.

[15] D.W.Bradley and A. M. Tyrell, “Immunotronics: Hardware Fault Tolerance Inspired By The Immune System”, Proceedings of the 3rd International Conference on Evolvable Systems: From Biology to Hardware (ICES2000), Vol. 1801, Lecture Notes in Computer Science, Springer-Verlag, pp. 11-20, April 2000.

[16] P. Marrack and J.W. Kappler, “How The Immune System Recognises The Body”, Scientific American, pp. 49-55, September 1993.

[17] D. W. Bradley and A. M. Tyrell, “Immunotronics-Novel Finite-State-Machine Architectures with Built-In Self-Test Using Self-Nonself Differentiation”, IEEE Transactions on Evolutionary Computation, Vol. 6, pp. 227-238, June 2002.

[18] D. W. Bradley and A. M. Tyrell, “Hardware Fault Tolerance: An Immunological Solution”, Proceedings of IEEE International Conference on Systems, Man, and Cybernetic, Vol. 1, pp. 107-112, 2000.

[19] P. D’haeseleer, “An Immunological Approach to Change Detection: Theoretical Results”, Proceedings of the 9th IEEE Computer Security Foundations Work-shop, County Kerry, Ireland, June 1996.

[20] S. T. Wierzchoń, “Discriminative Power of The Receptors Activated by k-Contiguous Bits Rule”, Technical Report, Institute of Computer Science, Polish Academy of Sciences, 2000.

[21] J. K. Percus, O. E. Percus, and A. S. Perelson, “Predicting the Size of The T-Cell Receptor and Antibody Combining Region from Consideration of Efficient Self-Non-Self discrimination”, Proc. Natl. Acad. Sci. USA, 1993.

[22] J. H. Holland, “Adaptation in Natural and Artificial Systems”, University of Michigan Press, ANN Arbour, 1975.

[23] S. T. Wierzchoń, “Deriving a Concise Description of Non-Self Patterns in an Artificial Immune System”, Technical Report, Institute of Computer Science, Polish Academy of Sciences, 2001.

[24] S. T. Wierzchoń, “Generating Antibody String in an Artificial Immune System”, ICS PAS Report No. 892, Institute of Computer Science, Polish Academy of Sciences, 1999.

[25] P. D’haeseleer, “Further Efficient Algorithm for Generating Antibody Strings”, Technical Report CS-95-03, The University of New Mexico, Albuquerque, NM, 1995.

[26] M. Morris Mano, “Digital Design”, 2nd Edition, Prentice Hall International Editions, USA, 1984.

APPENDIX

SIMULATOR

PROGRAM