ขั้นตอนวิธีพันธุกรรมสำหรับวงจรเชิงวิวัฒนาการแบบขนาน

นายยุทธนา เจวจินดา

วิทยานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรดุษฎีบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์ จุฬาลงกรณ์มหาวิทยาลัย
ปีการศึกษา 2551

PARALLEL APPROACH TO GENETIC ALGORITHMS FOR EVOLVABLE HARDWARE

Mr. Yutana Jewajinda

A Dissertation Submitted in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy Program in Computer Engineering
Department of Computer Engineering
Faculty of Engineering
Chulalongkorn University
Academic Year 2008

Thesis Title	Parallel Approach to Genetic Algorithms for Evolvable Hardware
By	Yutana Jewajinda
Field of Study	Computer Engineering
Advisor	Professor Prabhas Chongstitvatana, Ph.D.

Accepted by the Faculty of Engineering, Chulalongkorn University in Partial Fulfillment of the Requirements for the Doctoral Degree

[illegible] .... Dean of the Faculty of Engineering

(Associate Professor Boonsom Lerdhirunwong, Dr.Ing.)

THESIS COMMITTEE

[illegible] .... Chairman

(Associate Professor Sartid Vongpradhip, Ph.D.)

[illegible] .... Advisor

(Professor Prabhas Chongstitvatana, Ph.D.)

[illegible] .... Examiner

(Associate Professor Somchai Prasitjutrakul, Ph.D.)

Darida Sutivong .... Examiner

(Assistant Professor Daricha Sutivong, Ph.D.)

[illegible] .... External Examiner

(Associate Professor Kosin Chamnongthai, Ph.D.)

ยุทธนา เจวจินดา : ขั้นตอนวิธีพันธุกรรมสำหรับวงจรเชิงวิวัฒนาการแบบขนาน (PARALLEL APPROACH TO GENETIC ALGORITHMS FOR EVOLVABLE HARDWARE). อ.ที่ปรึกษาวิทยานิพนธ์หลัก : ศ. ดร.ประภาส จงสถิตย์วัฒนา, 111 หน้า

วิทยานิพนธ์นี้นำเสนอขั้นตอนวิธีพันธุกรรมอย่างย่อแบบเซล ซึ่งเป็นขั้นตอนวิธีพันธุกรรมแบบขนานสำหรับประยุกต์ใช้งานกับวงจรเชิงวิวัฒนาการ ขั้นตอนวิธีพันธุกรรมอย่างย่อแบบเซล ใช้การแลกเปลี่ยนโมเดลความน่าจะเป็นของกลุ่มประชากรแทนการแลกเปลี่ยนประชากรโดยตรง อีกทั้งใช้หลักการเก็บประชากรที่ดีที่สุด ขั้นตอนวิธีพันธุกรรมอย่างย่อแบบเซล ใช้หลักการปรับตัวในการรวมโมเดลความน่าจะเป็นของกลุ่มประชากรเข้าด้วยกัน และสามารถใช้แก้ปัญหาขั้นยากที่มีกรอบจำกัดได้ ขั้นตอนวิธีพันธุกรรมอย่างย่อแบบเซลถูกออกแบบมาเหมาะสมสำหรับสร้างเป็นวงจรเชิงเลข มีการนำเสนอสถาปัตยกรรมฮาร์ดแวร์แบบปรับขยายได้ของขั้นตอนวิธีพันธุกรรมอย่างย่อแบบเซล วิทยานิพนธ์ยังได้นำเสนอวงจรเชิงวิวัฒนาการซึ่งใช้งานร่วมกันระหว่างวิธีพันธุกรรมอย่างย่อแบบเซลและโคร่งข่ายประสาทเทียมโดยใช้หลักการสถาปัตยกรรมฮาร์ดแวร์แบบเป็นชั้น ได้ทำการประยุกต์วงจรเชิงวิวัฒนาการซึ่งใช้งานร่วมกันระหว่างวิธีพันธุกรรมอย่างย่อแบบเซลและโคร่งข่ายประสาทเทียมกับปัญหาการแยกสัญญาณคลื่นไฟฟ้าหัวใจ ซึ่งแสดงให้เป็นว่าวิธีพันธุกรรมอย่างย่อแบบเซลและวงจรเชิงวิวัฒนาการที่นำเสนอ สามารถนำไปแก้ปัญหาจริงได้

ภาควิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อนิสิต [illegible]
สาขาวิชา วิศวกรรมคอมพิวเตอร์ ลายมือชื่อ อ.ที่ปรึกษาวิทยานิพนธ์หลัก [illegible]
ปีการศึกษา 2551

4771820721 : MAJOR COMPUTER ENGINEERING
KEYWORDS : EVOLVABLE HARDWARE / GENETIC ALGORITHM / PARALLEL GENETIC ALGORITHM

YUTANA JEWAJINDA : PARALLEL APPROACH TO GENETIC ALGORITHMS FOR EVOLVABLE HARDWARE. ADVISOR : PROF. PRABHAS CHONGSTITVATANA, Ph.D., 111 pp.

The thesis proposes the cellular compact genetic algorithm (CCGA), which is a parallel probabilistic model-building genetic algorithm for evolvable hardware. CCGA replaces traditional migration of individuals with the probabilistic migration. Each CCGA node uses the traditional compact GA with elitism. CCGA employs adaptive combination of probability vectors from its neighbors. CCGA can solve hard problems of bounded difficulty. With parallel approach, CCGA supports scalability. In addition, CCGA is designed for hardware implementation. The scalable hardware architecture for CCGA is proposed. For each node of CCGA, the scalable hardware architecture supports expandable number of variables to be optimized with flexible precision and expandable chromosome length. Evolvable hardware based-on Cellular Genetic Algorithm (CCGA) and Block-based neural network (BBNN) is presented. The layer-based architecture is proposed for integrating CCGA with BBNN in hardware. A hardware design of BBNN neurons is proposed. The link-multiplexed concept is used for hardware design of BBNN neurons. The proposed evolvable hardware based-on CCGA and BBNN is applied to the problem of online ECG signal classification. This demonstrates that CCGA can solve the real-world problems. The proposed evolvable hardware can be implemented in FPGA or ASIC for a portable personalized ECG signal classifications for long term patient monitoring.

Department : Computer Engineering	Student's Signature [illegible]
Field of Study : Computer Engineering	Advisor's Signature [illegible]
Academic Year : 2008	

Acknowledgement

There are many people to whom I am grateful for helping to get all the way to finish up my dissertation. Foremost, I am thankful to the Lord for inspiring me to rethink about beginning my Ph.D. study once again in Thailand after long years of discouraging back here in the land of smiles.

I wouldn't be this far for my Ph.D. study at Chula if it wasn't my thesis advisor, Professor Prabhas Chongstitvatana, who gave me this great opportunity of working with him and other members of the Prof. Prabhas's lab. Prof. Prabhas is a great teacher and a bottomless source of inspiration when I felt like I wanted to give up many times.

Finally, I would like to say "Thank you" to my mother for her supporting.

Contents

List of Tables

List of Figures

CHAPTER I
INTRODUCTION

1.1 Evolvable Hardware

Evolvable hardware (EHW) is a one particular type of hardware whose architecture/structure and functions change dynamically and autonomously in order to improve its performance in performing certain tasks [1]. By contrast to the conventional hardware machines that the structures are irreversibly fixed during design processes, the evolvable hardware machines can adapt to changes in the field or in environments. In recent years, the emerging of the EHW research has been contributed by the progress in reconfigurable hardware and evolutionary computation [1]. The adaptability of EHW machines is performed by reconfigured its own hardware structure dynamically and autonomously by the mean of evolutionary algorithms.

One of the key motivations behind EHW is to learn or adapt itself in environments. Even being in an unknown environment, the EHW devices can adapt and provide the appropriate function intelligently. This vision for a new kind of hardware devices may have not been accomplished nowadays [2]. However, with the progress on EHW research and contribution from EHW researchers around the world, the ultimate goal of intelligent and adaptive hardware could be accomplished in the near future. This thesis is a part of this EWH research community.

Relatively new research field as it is, EHW has delivered some solutions to opened question of the real-world problems. There are real-world needs for evolvable hardware, which are still opened research problems. These are adaptive computing, adaptive signal processing, adaptive communication, adaptive fault-tolerant computing, and adaptive and autonomous devices for space exploration.

1.1.1 Evolvable Hardware: Basic Concepts

The integration of evolutionary computation and configurable hardware devices is the key to adaptability of EHW since the objective of evolvable hardware is the "autonomous" reconfiguration of hardware structure in order to improve performance [1]. The capacity for autonomous reconfiguration with evolvable hardware makes it fundamentally different from conventional hardware, where it is almost impossible to change the hardware's function once it is manufactured. While configurable hardware devices, such as a Programmable Logic Device (PLD) and Field Programmable Gate Array (FPGA), allow for some functional changes after being installed on a printed circuit board, such changes cannot be executed without the intervention of human designers (i.e., the change is not autonomous). With the use of evolutionary computation, however, evolvable hardware has the capability to autonomously change its hardware functions. Figure 1.1 shows the overview of the EH research field which lies between the computer science (evolutionary computation), and electronics engineering.

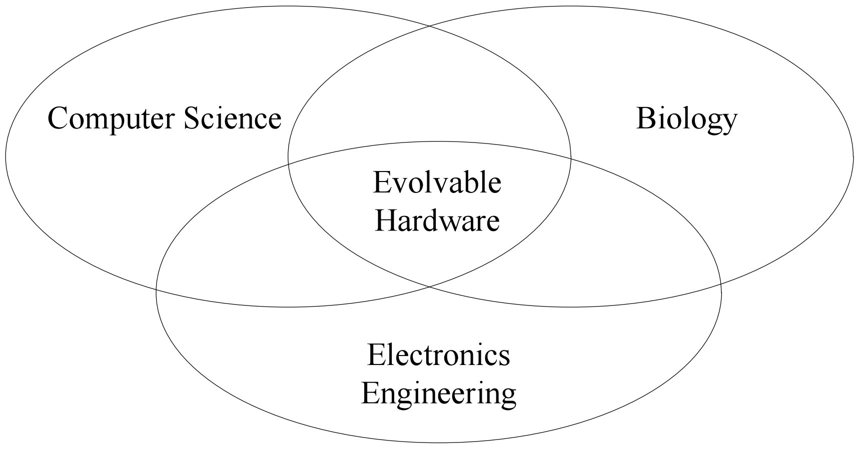


Figure 1.1: Overview of Evolvable Hardware

Although there are different views on what EHW is. In this thesis, we support the view of EHW as hardware that capable of online adaptation through adapting its structure and functions dynamically, and autonomously. Our view is different to the view of EHW as a way of using evolutionary algorithms to circuit synthesis [3]. The evolutionary design of electronic circuits has been around for a long time, especially for logic and circuit synthesis [3]. The evolutionary algorithms have been used to optimize certain processes in electronics design automation such as electronics board layout and VLSI placement and routing.

The origins of our focused evolvable hardware can be traced back to two papers by Daniel Mange [4, 5] and a paper by Tetsuya Higuchi [6]. Mange's research led to bio-inspired machines that aim at self-reproduction or self-repair of the hardware structure rather evolving new structures [7, 8], the Higuchi's work targeted evolvable hardware research utilizing genetic algorithms (GAs) for the autonomous reconfiguration of hardware structure [9, 10]. This dissertation focuses primarily on the GA-based evolvable hardware.

The key concept of the evolvable hardware is to regard the configuration bits of programmable hardware architecture as the chromosomes of GAs. By designing a fitness function to achieve a desired hardware function, the GA becomes a means of autonomous hardware configuration. Configuration bits "evolved" by the GA are repeatedly downloaded into the programmable hardware devices until the evolved hardware performance is satisfactory in term of fitness function values. This evolving process by GA is illustrated in Figure 1.2. From Figure 1.2, in the first step, new population, which represents different topology of circuits, are generated. Then, the circuits are measured for their performance. The GA bits that represent the circuits with higher performance are selected to be parents of the next population. By performing GA operators: crossover and mutation, the new population are generated.

A GA for evolvable hardware is executed either outside or inside the evolvable hardware, depending on its purpose. For example, if the speed of hardware reconfiguration is an important factor which is the case for intrinsic or online evolvable hardware, then GA should be a part of the evolvable hardware.

EHW can be classified into two categories: extrinsic and intrinsic EHW [3]. Extrinsic EHW simulates evolution by software running on an external computer and only downloads the best configuration to hardware in each generation or only once. Intrinsic EHW simulate evolution directly in its hardware, every chromosome will be used to configure the hardware.

There are a number of methods and techniques that propose the Genetic Algorithm (GA) and Evolutionary Algorithms (EA) to be implemented in hardware for intrinsic EHW, especially implement into FPGAs or other reconfigurable devices [11, 12]. However, intrinsically on-line evolving in hardware and to utilize hardware resource efficiently pose a challenging question of how to modify or invent efficient and improve GA or EA algorithms that can be effectively implemented into hardware.

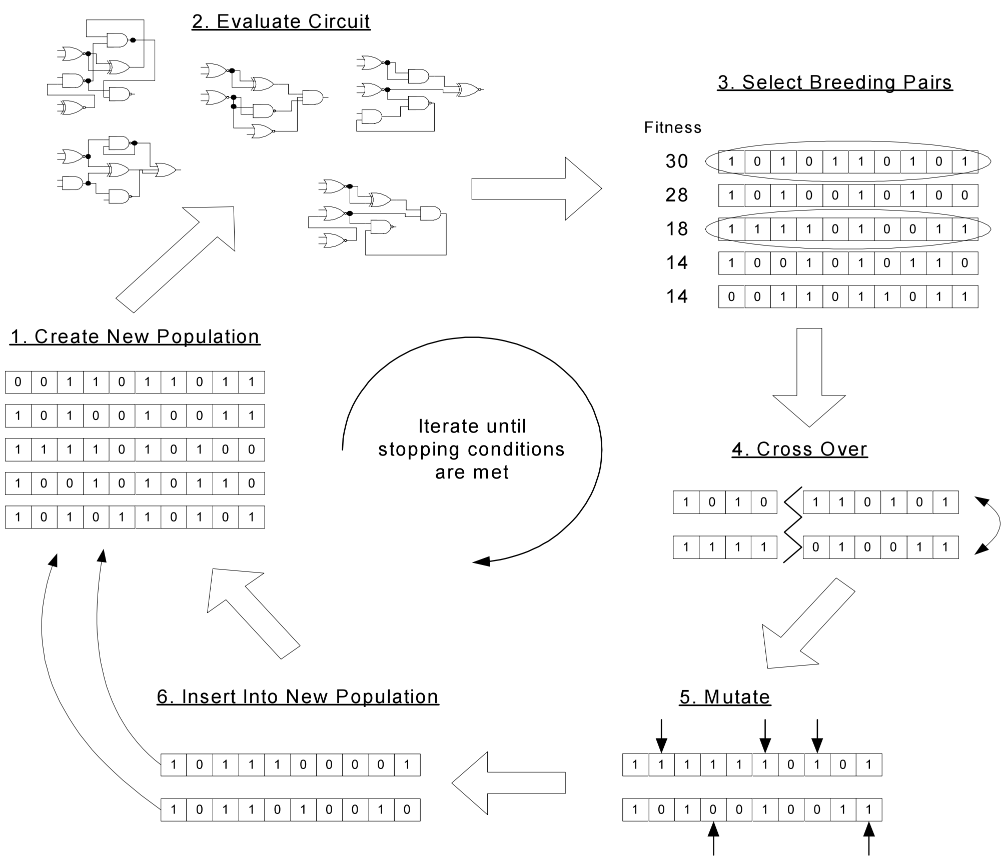


Figure 1.2: Genetic Algorithms as a mean for hardware evolution

In addition, regarding to the increasing of density and price per performance of current FPGAs due to advanced semiconductor process technology [13, 14], there is an opportunity for designers and researchers to use higher density and faster FPGA devices for EHW at reasonable cost. Considering this trend of FPGA technology development, the concept of implementing a group of parallel processing units for EHW into a single or array of FPGA chips are feasible. In this thesis, the concept of parallel genetic algorithms will be explored.

1.1.2 Applications of EHW

Through many years of EHW research, the application of EHW can be classified into three main categories, namely, EHW controllers, EHW recognizers and classifiers, and EHW as optimizers for other problems. These three EHW can be summarized as follows:

a) EHW Controllers. The EHW can be used as intelligent controllers for robots or other devices [8]. The typical EHW controller is shown in Figure 1.3.

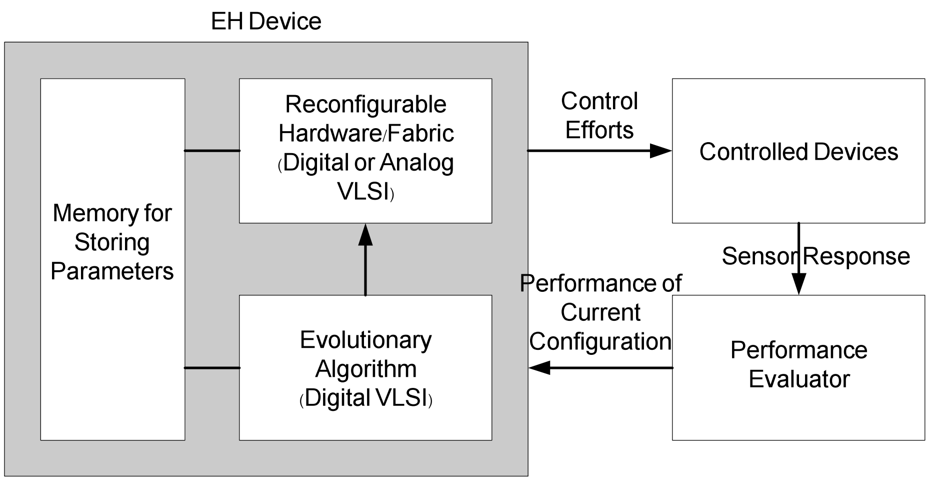

EH Device
Memory for Storing Parameters
Reconfigurable Hardware/Fabric (Digital or Analog VLSI)
Evolutionary Algorithm (Digital VLSI)
Control Efforts
Controlled Devices
Sensor Response
Performance of Current Configuration
Performance Evaluator

Figure 1.3: A typical EH device for control application.

b) EHW recognizers and classifiers. EHW is capable of performing classification on streams of data at faster data rate and lower power consumption. The adaptability of EHW provides flexibility and autonomous behavior to traditional classifiers. Figure 1.4 shows the typical EHW classifiers.

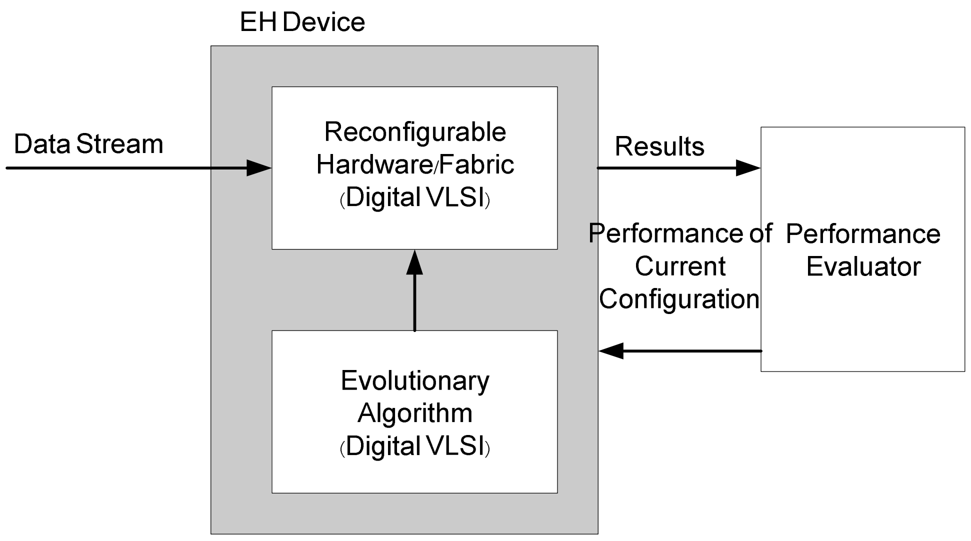

EH Device
Data Stream
Reconfigurable Hardware/Fabric (Digital VLSI)
Evolutionary Algorithm (Digital VLSI)
Results
Performance of Current Configuration
Performance Evaluator

Figure 1.4: A typical EH device for online classification

c) EHW as optimizers for other problems. This class of EHW is for other EHW that can not be classified into EHW controllers and EHW classifiers. There are some research projects that use evolutionary algorithm integrated hardware to provide solution to specific problems [15].

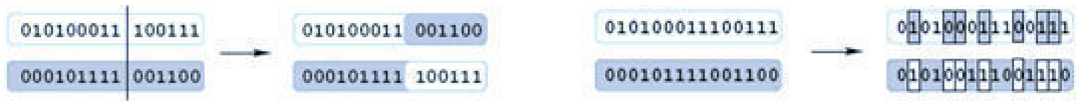
1.2 From Genetic Algorithms to Probabilistic Model-Building Genetic Algorithms

1.2.1 Genetic Algorithms

Genetic algorithms (GAs) [16, 17] are stochastic optimization methods inspired by natural evolution and genetics. GAs approaches optimization by evolving a population of candidate solutions with the operators inspired by natural evolution and genetics. Maintaining a population of solutions, as opposed to a single solution, has several advantages. Using a population allows a simultaneous exploration of multiple basins of attraction. Additionally, a population allows for statistical decision making based on the entire sample of promising solutions even when the evaluation procedure is affected by external noise.

Genetic algorithms represent candidate of solutions as binary strings to vectors of real numbers, to permutations, or even to production rules. There is no strict restriction on representations. However, most GAs uses binary strings representation for each candidate of solutions. The performance of each candidate solution is represented by a real number called *fitness*. The task of GAs is to find the binary string with the highest fitness.

The first population of candidate solutions is usually generated randomly with a uniform distribution over all possible solutions. Each iteration starts by selecting a set of promising solutions from the current population based on the performance of each solution by using various selection operators. There are two methods commonly used: tournament selection and truncation selection. In this thesis tournament selection is preferred. Tournament selection iteratively selects one solution at a time by first choosing a random subset of candidate solutions from the current population and then selecting the best solution out of this subset. Once the set of promising solutions has been selected, new candidate solutions are sampled by applying recombination (crossover) and mutation to the promising solutions. Recombination combines subsets of promising solutions by exchanging some of their parts; mutation perturbs the recombined solutions slightly.



(a) One-point crossover (b) Uniform crossover

Figure 1.5: An illustrative two types of two-parent crossover operators.

In one-point crossover, the tails are exchanged after a randomly chosen position. In uniform crossover, the bits on each position are exchanged with probability 50%. For example, one-point crossover first randomly selects a single position in the two strings and exchanges the bits on all the positions after the selected one (see Figure 1.5 (a)). On the other hand, uniform crossover exchanges bits on each position with probability 50% (see Figure 1.5 (b)).

1.2.2 Probabilistic Model-Building Genetic Algorithms: concepts

Probabilistic model-building genetic algorithms (PMBGAs) use probabilistic modeling of promising solutions to guide the exploration of the search space instead of using the traditional recombination and mutation operators of simple GAs [18, 19]. There are many ways of estimating the probability distribution of promising solutions and each PMBGA deals with the problem of estimating distributions in its own way.

```
(1) set t := 0
    randomly generate initial population P(0)
(2) select a set of promising string S(t) from P(t)
(3) estimate the probability distribution of the selected set S(t)
(4) generate a set of new strings O(t) according to the estimate
(5) create a new population P(t+1) by replacing some strings
    from P(t) with O(t)
    set t := t + 1
(6) if the termination criteria are met, go to (2)
```

Figure 1.6: Pseudo-code of the general PMBGA procedure.

The general procedure of PMBGAs is similar to that of GAs. The initial population of PMBGA is generated at random. In each iteration, promising solutions are first selected from the current population of candidate solutions. The true probability distribution of the selected solutions is then estimated. New candidate solutions are then generated by sampling the estimated probability distribution. The new solutions are then incorporated into the original population, replacing some of the old ones or all of them. The process is repeated until the termination criteria are met. The pseudo-code of the PMBGA procedure is shown in Figure 1.6. The difference between PMBGAs and traditional GAs is in the way PMBGAs process a population of promising solutions to generate new candidate solutions. Instead of performing crossover on pairs of selected solutions with a certain probability and then applying mutation to each of the resulting solutions, the following two steps are performed [20]:

1. Model building. A probabilistic model of promising solutions is constructed.
2. Model sampling. The constructed model is sampled to generate new solutions.

PMBGAs differ in how they cope with the above two steps and in whether they incorporate special selection or replacement mechanisms for processing the populations of solutions. In the literature, PMBGAs are also called estimation of distribution algorithms (EDAs).

1.2.3 Parallel Genetic Algorithms

In order to increase the GA's efficiency, the parallelization of GA has been the active research topic using high performance computer systems [21, 22]. The parallelized GA (PGA) can be categorized into four approaches. These are global, coarse-grained, fine-grained, and hybrid approaches.

1) Global parallelization In this class of model, there is only one group of population. The evaluation of individual and execution of genetic operators are performed in parallel. The evaluation can be parallelized by assigning a group of individuals to a processor node to evaluate and send the results back to the common shared memory or a master node. There is no communication between each processor that evaluates each individual.

2) Coarse grained parallelization This is the popular model for the parallelized GAs. The whole population is partitioned into sub-populations. Within each sub-population, individuals can only mate with others in their own sub-population. However, there is an introduction of migration operator the send some individuals from a local sub-population to other sub-population. There are key parameters for this model: topology, migration rate, and migration interval. The topology defines how each sub-population connects to other sub-populations. The migration rate and migration interval specify how many individuals in each sub-population are migrated and how often they are migrated.

3) Fine grained parallelization The whole population is divided further into even smaller sub-population than the coarse grain model. The ideal case is each individual handled by only one processor node. This ideal case rarely happens in real world implementation in high performance computer systems excepting the implementation into special hardware bit-level. In summary, this model is similar to the massively parallel processors

4) *Hybrid parallelization* This approach is to combine two approaches to solve more difficult problems. For the coarse grained and fine grained approaches, in order to exchange individuals between each sub-population, the question of how to communicate and how costly in term of resources need to be considered. These problems are related to migration parameters: topology, migration interval and migration rate.

1.3 Thesis Objectives

There are five primary objectives in this thesis:

(1) Design a new parallel genetic algorithm capable of feasible hardware implementation.

(2) Extend the proposed algorithm to the hardware implementation

(3) Test the developed algorithm on the designed class of problems

(4) Design an evolvable hardware based-on the proposed algorithm

(5) Test the designed evolvable hardware on a class of problems and real-world applications.

1.4 Contributions

This thesis has made the following contributions:

(1) Proposed the cellular compact genetic algorithm (CCGA), a new parallel genetic algorithm and its hardware architecture for evolvable hardware.

(2) Proposed an evolvable hardware based-on CCGA and block-based neural network (BBNN) to form an evolvable hardware machine.

(3) Proposed using CCGA and BBNN as an evolvable hardware to online heartbeat monitoring and classification.

Specifically, the thesis proposes the cellular compact genetic algorithm (CCGA) and its integrating with block-based neural network (BBNN) to form evolvable hardware machines. Classified as a kind of parallel estimation of distribution algorithms, the CCGA uses the improved compact genetic algorithm with probability model migration. BBNN is an evolutionary neural network in which evolution is another form of adaptation in addition to learning. CCGA and BBNN hardware architecture are proposed. The parallel hardware architecture for integrating CCGA and BBNN presents a class of evolvable hardware.

Empirical evidence is provided to show that CCGA is capable of solving problems decomposable into sub-problem with growing difficulty and in scalable manner.

1.5 Dissertation layout

The thesis is divided into six chapters. The first chapter introduces genetic algorithms, and the basic concepts of probabilistic model-building genetic algorithms (PMBGAs). Chapter II presents the related work to the thesis. Chapter III presents the cellular genetic algorithm (CCGA) and its hardware implementation. Chapter IV presents the evolvable hardware based on CCGA and block-based neural network (BBNN). Chapter V motivates the use of the proposed evolvable hardware to solve the real-world problem of ECG signal classification. The thesis closes by presenting experimental results on two classes of real-world problems, discussing interesting topics for future work, and providing the conclusions. The following subsections present the content of each chapter in greater detail.

1.6 Publications

This thesis has made following publications:

(1) Y. Jewajinda and P. Chongstitvatana, “A cooperative approach to compact genetic algorithm for evolvable hardware,” Proc. IEEE Congress on Evolutionary Computation, Vancouver, Canada, 2006, pp. 624–629.

(2) Y. Jewajinda and P. Chongstitvatana, “Cellular Genetic Algorithm for Evolvable Hardware”, ECTI International Conference, Thailand, 2008.

(3) Y. Jewajinda and P. Chongstitvatana, “FPGA Implementation of a Univariate Estimation of Distribution Algorithm and Block-based Neural Network as An Evolvable Hardware”, IEEE World Congress on Computational Intelligence, Hong Kong, June, 2008.

(4) Y. Jewajinda and P. Chongstitvatana, “FPGA Implementation of a Cellular Genetic Algorithm, NASA/ESA international conference on Adaptive hardware and Systems (AHS-2008), Netherland, July, 2008.

(5) Y. Jewajinda and P. Chongstitvatana, An Adaptive Hardware Classifier in FPGA based-on a Cellular Compact Genetic Algorithm and Block-based Neural Network”, International Symposium on Communications and Information Technology (ISCIT 2008), Vientiane, Laos, October, 2008.

CHAPTER II
LITERATURE REVIEWS

2.1 Evolvable Hardware: Vision and Motivation

One of the major goals of computation intelligence research is to create method and systems that strive towards human level intelligence. Human level intelligence is manifested through adaptive learning, associative memory, pattern recognition, language communication, concept formation, abstract thinking, common sense of knowledge, consciousness [23]. So far, the methods of computational intelligence research have been successfully used in creation of some elements of human intelligence, but not for the creation of human level intelligent machines. We need to gain more knowledge to enable us to achieve a higher-degree of the intelligence in machines. Two approaches to such high level of intelligence are to understand evolving processes of the learning brain and biological organisms. Understanding human intelligence in brain allows us to implement machines closer to the human-level intelligence. In addition, understanding the development process of biological organisms allows us to create machines that capable of cellular differentiation and cellular division which are key mechanism of living organism.

The brain is a dynamic information processing system that evolves its structure and functionality in time through information processing at different hierarchical levels. The term “evolving” is mainly concerned with the development of the structure and functionality of the individual system during its lifetime. Understanding of the interaction through its modeling is the key to comprehend information processing in brain and perhaps the brain as a whole. Building computation models that integrate principles from brain based on connectionist learning may be well towards constructing intelligence machines. These models can be called evolving connectionist system.

The majority of living creatures, with exception of uni-cellular organism, share a common multi-cellular structure in which the organism is divided into a finite number of cells and each group realizing a single function such as muscle, neuron, etc. The process that allows organisms to develop from a single initial cell to a fully-grown individual relies essentially on two mechanisms: cellular division and cellular differentiation. The developing process of cells can inspire us to create artificial machines that are capable of these two key mechanisms. Since the bio-organisms inspire us to create hardware machines that imitate living organisms, we can call this type of machines: bio-inspired machines.

2.1.1 Evolving connectionist system

Evolving connectionist system is an adaptive, incremental learning and knowledge representation system that evolves its structure and functionality, where in the core of the system is connectionist architecture and that consists of neurons and connections between them. Evolving connectionist model operates continuously in time and adapts their structure and functionality through a continuous interaction with environment and with other systems. The adaptation can be defined through:

1. A set of evolving rules
2. A set of parameters that are subject to change during system operation
3. An incoming continuous flow of information, possibly with unknown of distribution
4. Goal criteria that are applied to optimize the performance of the system over time

Evolving connectionist system presented in this thesis is based-on principles from the human brain. It is known that the human brain develops before the child is born. During learning the brain allocates neurons to respond to certain stimuli and develop their connections. Some parts of brain develop connections and also retain their ability to create neurons during person's lifetime.

Adaptive learning and behaviors are keys of human level intelligence. Until recently, most researches on machine intelligence have been carried on in software domain operated on fixed hardware. The emerging field of evolvable hardware and adaptive hardware promise to deliver elements of machine intelligence at hardware level. Principles of brain development and learning can be used as inspiration for the development of evolvable hardware based-on evolving connectionist system:

1. Evolution is achieved through both genetically defined information and learning
2. The evolved neurons in the brain have a spatial-temporal representation where similar stimuli activate close neurons

3. Evolving through interaction with the environment
4. The evolving process is continuous and life long
5. Redundancy is the evolving process in brain leading to the creation of a large number of neurons involved in each learning tasks

Evolvable hardware based-on evolving connectionist system can be proposed as part of the classical evolvable artificial neural networks. Traditionally, evolving connectionist system is based-on software implementation. However, the concept of evolving connectionist system can be applied towards building machine intelligence at hardware level. Since most of evolving connectionist system is implemented in software running on top of the fixed hardware. There opened research opportunities for us to build an adaptive and learning aim at human intelligence level in silicon-based hardware devices.

2.1.2 Bio-inspired Hardware Model

As mentioned in the previous section, the developmental process of biological organisms exploits essentially two mechanisms: cellular differentiation and cellular division.

Cellular division is the process through which each cell achieves its duplication. During this phase, a cell copies its genetic material (i.e. the genome) and splits into two identical daughter cells.

Cellular differentiation defines which function a cell has to realize. This specialization, which essentially depends of the cell's position in the organisms, is obtained through the expression of a part of the genomes.

In order to implement cellular division, new approaches have to be explored and invented for the current hardware technology mainly silicon-based hardware. However, to directly implement cellular division in silicon-based system is not possible. An alternative way is to implement it at logical structures instead of physical level. This is the same concept used in field programmable logic devices (FPGAs). There are many approaches to implement cellular division and differentiation. The notable are self-replicating and embryonic approaches.

2.1.3 Evolvable Hardware Based-on Evolving Connectionist System and Bio-inspired Machines.

The potential of evolvable hardware is its capability to adapt or change its behavior and improve its performance while executing in real physical environment. Acquiring knowledge from the evolving connectionist system and bio-inspired approach provide solid models for realization of the innovative evolvable hardware. Typical design of evolvable hardware from evolving connectionist system and bio-inspired machines has to address following issues:

- *Genotype-phenotype mapping*. It is common knowledge that the information stored in the genome is not sufficient to completely define the structure of the organisms. The current research suggests that genotypes code instructions on how the cells grow. The codes are interpreted the help of applying evolutionary mechanism to their design. The complex genotype-to-phenotype mapping allows reduction in size of the genome, with a consequence increase in its evolvability.
- *Structural adaptation*. Networks of neuron or cells that adapt or self-organize structurally to the environment by adding and removing neurons and connections in the system exploit mechanisms that are similar to the those used in the growth of an organism.
- *Unknown environmental adaptation*. Environment-directed adaptation and development occurs in bio-organisms. Mostly, currently implemented systems, hardly implemented in hardware, only adapt themselves with existed fixed-structure or known environment.
- *Online Adaptation*. The online adaptation means adaptation while the hardware is executing in a real physical environment. In a sense, it can be view as real-time adaptation. Online adaptation requires the adaptive hardware to learn incrementally and responsively. Such requirements do not seem to be met by population-based evolutionary learning. The current algorithms have limitation of re-learning the new and the old information in order to response to the new changed environment. In addition the evolutionary learning tend to require more time to reach the optimum solutions.

- *Disaster Prevention in Real-time online adaptation.* The risk of trial and error of the evolutionary algorithms can create severe damages to physical devices or other devices under control by the adaptive/evolvable hardware.
- *Adaptation to a high-level specification of intended function.* The true useful adaptive systems have to adapt themselves to provide functional without giving details. That is they have to provide users-friendly in setting new high-level adaptation objective.
- *Minimal overhead for adaptation.* The adaptive hardware/systems have to require low overhead in term of limited or expensive resources: times and spaces
- *Timely Adaptation.* Adaptation process has to be as fast as needed and deliver rapid reactions.
- *Decision making at hardware level.* So far, evolvable and adaptive hardware systems do not implement decision making process. The nowadays systems tend to implement the decision process in software for system-level adaptation.
- *Hardware change autonomously apart from software.* Most current implemented system. The hardware handles command sent by software module in order to perform adaptation to software-set objective. The goal is to allow hardware to adapt by interacting with environment directly and then inform software.
- *New model for integrating software and evolvable/adaptive hardware.* Traditional and successful model is in the form Von-Neumann machine in which the hardware provides instruction sets for programmer to create software to control the hardware. With more intelligence and adaptive hardware, the new model of interface between hardware and software can be proposed.
- *Generalization.* Generalization is a key issue for any learning system. How well of the learning capability of the adaptive/evolvable hardware upon the unknown environment or inputs.

2.1.4 Dualism of Evolvable Hardware

It can be observed that there is a kind of dualism in the various proposed evolvable hardware. The dualism consists of a network of topology and a learning algorithm. The network of topology can consists of simple or complex processing elements. The network of topology can support adaptation at its processing unit and its topology or the way they are interconnected. This dualism in a sense is analogous to hardware and software in current computing machines. In order to evolve or adapt efficiently, the learning algorithm and the hardware fabric should provide flexibility to evolve which we can call evolvability. Thus, the research direction for evolvable hardware has to address both sides of this dualism. The more flexible fabric of topology and processing units, the more powerful functions will be derived from the learning algorithm.

2.1.5 Evolutionary Artificial Neural Network and Evolvable Hardware

Evolutionary Artificial Neural Network (EANN) is a special class of artificial neural network that uses evolutionary algorithm to search for its parameters [24]. EANN is a model of evolving connectionist systems (ECOS) [23, 24]. According to, an ECOS is an adaptive, incremental learning and knowledge representation system that evolves its structure and functionality, where in the core of the system is a connectionist architecture that consists of neurons (information processing units) and connections between them. EANN employs the evolutionary algorithms to find important parameter of ANN such as weights, connections, learning rules. In other words, EANN can be a type of EHWs.

The simple model of ANN is the feed forward neural networks. Architecture of feed forward ANN is shown in Figure 2.1. ANN consists of a group of processing element called neuron or nodes. ANN can be described by a transfer functions in the form:

$$y_j = f\left(\sum_{i \in I} w_{ij} x_i + b_j\right), \; j \in J \qquad (1)$$

Where y_i is the output node *i,* x_i is the input, and w_{ij} is the connecting weight between nodes i and j.

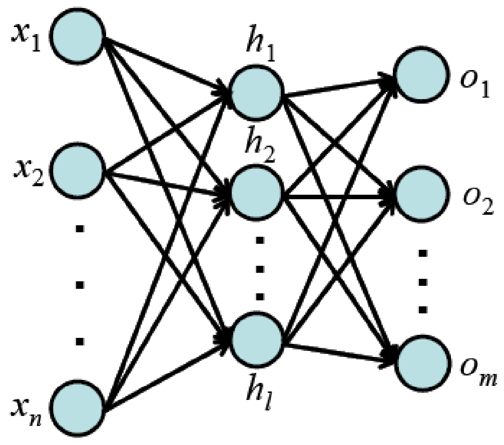


Figure 2.1: Feed forward Artificial Neural Network.

Optimizing the weights of ANN has been traditionally performed using training algorithm such as back-propagation (BP) and conjugate gradient [25]. BP has drawbacks due to its use of gradient descent [26] so it often get trapped in a local minimum of the error function and is incapable of finding minimum if the error function is multimodal and/or non-differentiable. Evolutionary Algorithm (EA) has been proposed to optimize weights of ANNs [27-30].

In this thesis, a special class of ANN called *Block-Based Neural Network* (BBNN) is introduced to be integrated with our proposed genetic algorithm to form evolvable artificial neural network (EANN) [31]. The BBNN consists of a two-dimensional (2-D) array of basic neural-network blocks with integer weights for easier implementation using reconfigurable hardware such as field programmable logic arrays (FPGAs).

BBNNs are represented by fixed-length binary codes, which correspond to network configuration bit strings of FPGAs to determine internal structures. The structure and weights of the BBNN are encoded as a 2-D chromosome for easier partial on-line reconfiguration. A genetic algorithm can evolve configuration bit strings to search for an optimal structure and weights setting of the BBNN among many possible choices of structure and weight combinations.

2.2 Evolvable hardware: Survey

Yao and Higuchi [2] wrote a good survey paper for evolvable hardware, they describe two difference classes of evolvable hardware (EH): evolvable hardware used as alternative for circuit design and evolvable hardware as online adaptive hardware. To achieve online adaptation, evolvable hardware must adapt its architecture while operating in real environment. Many new issues arise when online adaptation is required. This section presents a survey on new development of the field.

The first evolvable hardware chip developed for myo-electric hand control is described in Kajitani et.al [32]. Sakanashi, et al. developed GA-based EHW applied to image compression [33]. Korenek *et al.* have developed and evaluated a specialized architecture to evolve relatively large sorting networks in an ordinary FPGA [34].

Some new and practical applications have also been studied. Martinek et al. have proposed an evolvable image filter that was completely implemented in an FPGA [35]. The system is able to evolve an image filter in a few seconds. Smith et al. [36] have presented an application of GA to evolve new spatial masks for nonlinear image processing operations, which are ultimately to be implemented to evolvable hardware.

Digital evolvable hardware has also been applied to evolving robot controllers. Kim et al have shown that their proposed GA guarantees satisfactory smooth and stable walking behavior in an experiment involving a real biped robot [37].

Stefatos investigated the problem of evolvable FIR digital filter [38]. Using reconfigurable arithmetic architecture evolving with evolutionary algorithms provides promising solution over traditional design [39].

2.3 Probabilistic Model-Building Genetic Algorithms

This section reviews PMBGAs proposed in the past. The methods are classified according to the underlying representation of candidate solutions and the complexity of the class of models they consider. The algorithms that use a probabilistic model are also called the estimation of distribution algorithms (EDAs). There are generally three classes of EDAs that can be applied to problems with solutions represented by fixed-length string over a finite alphabet. The algorithms are classified according to the complexity of the class of models they use [40].

2.3.1 Without Dependency

The basic approach is to assume that the variables represent the distribution of solutions are independent. Figure 2.2 shows the graphical model with no interaction among the variables. There are three key EDAs within this class: population-based incremental learning (PBIL) algorithm [41], univariate marginal distribution algorithm (UMDA) [42], and compact genetic algorithm (cGA) [43].

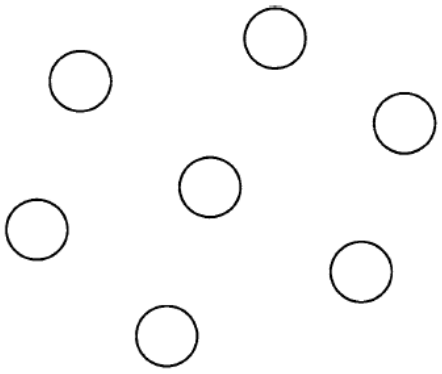


Figure 2.2: Graphical models with no interactions

In the univariate marginal distribution (UMDA), the probability model is represented as factorized of a product of independent univariate marginal distributions. That is:

$$p_l(x) = p\,(x|D_{l-1}^{Se}) = \prod_{i=l}^{n} p_l(x_i) \qquad (2.1)$$

```
$D_0$ ← Generate $M$ individuals (the initial population) at random
**Repeat** for l = 1, 2, … until the stopping criterion is met
    1. $D_{l-1}^{Se}$ ← Select $N \leq M$ individuals from $D_{l-1}$ selection
       method
    2. Estimate the joint probability distribution
           $p_l(X) = p(X|D_{l-1}^{Se}) = \prod_{i=1}^{n} p_l(x_i)$
    3. Generate a new population
       $D_l$ ←  Sample $M$ individuals from $p_l(X)$
```

Figure 2.3: Pseudocode of UMDA

D_l^{Se} is the current data file containing selected individuals of generation l in which each D_{l-1}^{Se} contains many cases of patterns of "one" and "zero". Table 2.1 shows an example of D_l^{Se}. Each univariate marginal distribution is estimated from marginal frequencies:

$$p_l(x_i) = \frac{\sum_{j=1}^{N} \delta_j(X_i = x_i \mid D_{l-1}^{Se})}{N} \tag{2.2}$$

Where

$$\delta_j(X_i = x_i \mid D_{l-1}^{Se}) = \begin{cases} 1 \text{ if in the j}^{\text{th}}\text{case of } D_{l-1}^{Se}, X_i = x_i \\ \quad 0 \quad \text{otherwise} \end{cases} \tag{2.3}$$

Table 2.1: An example of D_l^{Se}

	X1	X2	X3	X4
1	1	0	1	0
2	0	1	0	1
3	1	0	0	0
4	1	1	0	1

```
Obtain an initial probability vector:

$p_0(X) = (\, p_l(x_1), \dots, p_l(x_i), \dots, p_l(x_n))$

**while** no convergence **do**
  **begin**
  1. Using $p_l(x)$ obtain $M$ individuals: $x_1^l, \dots\ x_k^l, \dots\ , x_M^l$
  2. Evaluate and rank $x_1^l, \dots\ x_k^l, \dots\ , x_M^l$
  3. Select the $N$ ($N \le M$) best individuals
  4. Update the probability vector
         **For** i = 1, … n **do**
         $p_{l+1}(X_i) = (1- \propto)p_l(x_i) + \propto \frac{1}{N}\sum_{k=1}^{N} x_{i,k:N}^l$
**end**
```

Figure 2.4: Pseudocode of PBIL

Population-based incremental learning (PBIL) algorithm represents the solution by binary string of fixed length. In PBIL, population of solutions is replaced by the probability vector which is initially assigned each value of each position in the vector to the same probability at 0.5. Each generation the probability vector is shifted towards the desired solutions by using Hebbian learning rules [41]. Figure 2.4 shows the pseudocode of PBIL.

The compact GA (cGA) represents the population with a single probability vector. The pseudocode of the compact GA is shown in Figure. 2.5. At each generation, the two individuals are randomly generated from the probability vector. Then, tournament selection is performed over the two individuals. Each bit of the probability vector is adjusted according to the result of the competition. Eventually, the CGA keeps running until the probability vector is converged.

```
1.  Initialize probability vector
        $p(X) = (p_l(x_1), ..., p_l(x_i), ..., p_l(x_n))$
              = (0.5, ..., 0.5, ..., 0.5)
2.  Generate two individuals from the vector $p(X)$
          a = generate (p);
          b = generate (p);
3   Let them compete
          winner, loser = evaluate (a, b)
4.  Update the probability vector toward the winner
          for = 1 to n do
             if winner[i] != loser[i] then
             if winner[i] = 1 then p[i] += 1/N
             else p[i] -= 1/N
5.  Check if the probability vector has converged
          for i := 1 to n do
              if p[i] > 0 and p[i] < 1 then goto step 2
6.  $p(X)$ represents the final solution
```

Figure 2.5: Pseudocode of compact GA

2.3.2 Pairwise Dependency

By taking dependency between a pair of variables, there are three key algorithms in this class. The mutual information-maximizing input clustering (MIMIC) algorithm, a dependency tree approach proposed, and the bivariate marginal distribution algorithm (BMDA) are key algorithm for pairwise dependency [44]. Figure 2.6 demonstrates the difference of these three algorithms in term of graphical model.

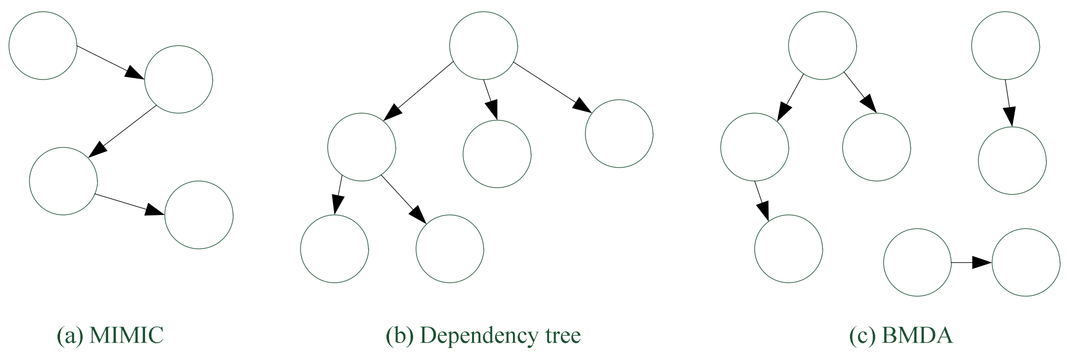


Figure 2.6: Graphical model of pairwire dependency algorithms

The mutual information-maximizing input clustering (MIMIC) algorithm which uses a simple chain distribution as shown in Figure 2.5. MIMIC employs greedy search approach to search in each generation for the best permutation of variables to construct a chain of variables. In this fashion the Kullback-Liebler divergence between the chain and the complete joint distribution is minimized. Since MIMIC uses only a greedy search algorithm, therefore global optimality of the distribution is not guaranteed.

Baluja and Davies [41] use dependency trees to model promising solutions (see Figure 2.5). Similarly as in the PBIL, the population is replaced by a probability vector which contains all pairwise probabilities. The probabilities are initialized to 0.25 and repeatedly adjusted according to new promising solutions acquired on the fly. Estimation of the probability distribution of the selected individuals in each generation is done using a tree structured Bayesian network with a learning algorithm proposed by Chow and Liu [41]. Once the probabilistic model is derived, individuals of population are sampled from it. There are two major advantages of using trees instead of chains. Trees are more general than chains because each chain is a tree.

Moreover, by relaxing constraints of the model, in order to find the best model (according to a measure decomposable into terms of order two), the global optimality of the solution can be derived.

The bivariate marginal distribution algorithm (BMDA) uses a set of dependency graph that need only second-order statistics [41]. A set of dependency graph is similar to a set of trees that not mutually connected. This class is even more general than dependency trees approach because the a dependency tree is a set of dependency graph. Pearson's chi-square test is used to determine which pair of variables should be connected and to construct the final mode. In each generation the factorization obtained with the BMDA is given by:

$$p_l(X) = \prod_{x_r \in R_l} p_l(x_r) \prod_{x_i \in V \setminus R_l} p_l(x_i | x_{j(i)}) \qquad 2.4$$

where V is the set of n variables, R_l denotes the set containing the root variable in generation l, and $X_{j(i)}$ return the variable connected to the variable X_i and added before X_i. The probabilities for the root nodes, $p_l(x_r)$ and the conditional probabilities, $p_l(x_i|x_{j(i)})$ are estimated from database, D_{l-1}^{Se}, containing selected individuals.

2.3.3 Multivariate dependencies

For difficult problems with multivariate or highly-overlapping building blocks, pairwise dependency can not solve those problems efficiently. Researchers propose to use more complex graphical model to handle such problems. However, using more complex model tend to increase computing time and sometimes still do not guarantee global optimality of the resulting probabilistic models. In this section, the extended compact genetic algorithm (ECGA) [44], factorized distribution algorithm (FDA), and Bayesian optimization algorithm (BOA) are presented.

ECGA groups the variables into many clusters. Each cluster is taking as a whole and different clusters are considered to be mutually independent. To discriminate models, ECGA uses a minimum description length (MDL) metric which prefers model that allows higher compression of data.

The factorized distribution algorithm (FDA) uses a factorized distribution as a fixed model throughout the whole computation. The structure of problem which is the distribution and its factorization needs to be given by experts. Unfortunately, this usually not available when solving real-world problems, and therefore the use of FDA is limited to some specific problems.

The Bayesian optimization algorithm (BOA) models the probability using Bayesian networks. BOA uses the Bayesian Dirichlet (BD) equivalence metric to measure the quality of each network. The BD metric does not prefer simpler model to the more complex ones. It uses only accuracy as the criterion. The search for the model is done by using greedy search and it starts each generation from scratch. BOA has been continuously developed and improved [44].

2.4 Test Functions for Genetic Algorithms

This section presents test functions for the proposed GAs. In experiments described in this dissertation, various test problems are used to compare the experimental results with the compact GA which is equivalent to simple GAs [43]. Fitness value (e.g., the number of correct BBs) over generations of population is taken to be performance measure.

2.4.1 Problems Involving Lower Order BBs

A 100-bit one-max problem (i.e., the counting ones problem) and a minimum deceptive problem (MDP) (formed by concatenating ten copies of minimum deceptive function) are considered for evaluating the proposed algorithms on problems involving lower order BBs [45]. The one-max problem and the MDP are representative problems with the order-one BBs and the order-two BBs, respectively.

The MDP problem is defined by

$$f_{MDP} = \sum_{i=1}^{m} f(x_{2i}),$$

$$\text{where } f(x_{2i}) = \begin{cases} 0.7, if\ x_{2i} = 00 \\ 0.4, if\ x_{2i} = 01 \end{cases} \text{ and } f(x_{2i}) = \begin{cases} 0.0, if\ x_{2i} = 10 \\ 1.0, if\ x_{2i} = 11 \end{cases} \quad (2.5)$$

Here, x_{2i}, presents the value (i.e., alleles) of a 2-bit long sub-string.

2.4.2 Problems involving higher order BBs

Fully deceptive problems are used to test the proposed algorithms on problems involving higher order BBs. Deceptive trap functions are used in many studies of GAs because their difficulty is well understood and it can be regulated easily. The deceptive trap is defined by

$$f_{trap}(u,a,b,z,k) = \begin{cases} \left(\frac{a}{z}\right)(z-u), if\ u \leq z \\ \left\{\frac{b}{(k-z)}\right\}(u-z), otherwise \end{cases} \quad (2.6)$$

where u is defined as the number of ones of a sub-string, a and b are the local deceptive and the global optimum respectively, z is the slope-change location, and k is the problem size.

The first deceptive problem is based-on a three-bit trap function. The test problem is formed by concatenating thirty copies of the three-bit trap function for a total chromosome length of 90 bits. Each three-bit trap function has a deceptive-to-optimal ratio of 0.7. That is, the problem is formulated by

$$f_{3-bit} = \sum_{i=1}^{10} f_{trap}(u_{3i}, 0.7, 1, 2, 3) \quad (2.7)$$

where u_{3i}, is a 3-bit long string.

The second deceptive problem is 4-bit trap which is formed by concatenating thirty copies of the 4-bit trap function for a total chromosome length of 120 bits. Each 4-bit trap function has a deceptive-to-optimal ration of 0.7. That is the problem is specified by

$$f_{4-bit} = \sum_{i-1}^{10} f_{trap}(u_{4i}, 0.7, 1, 3, 4) \quad (2.8)$$

2.4.3 Continuous and Multimodal Functions

Many real-world problems do not involve the concatenation of distinct order-k BBs in a simple manner since their solution and search space are continuous and multimodal. The problems can be modeled as an intricate combination of lower and higher-order BBs. In order to investigate the performance on such problems, a circle function and Schaffer's binary function are employed [45]. The functions maybe used for modeling several real-world problems, especially those arising in the emerging areas of wireless networks.

The circle function is investigated. It is defined in (2.9) and plotted in Figure 2.7. This multimodal function has many local optima (i.e., minima) there are located on concentric circles near the global optimal.

$$\text{Minimize } f_c(x) = \left(\sum_{i=1}^{n} x_i^2\right)^{1/4}\left[sin^2\left(50\left(\sum_{i=1}^{n} x_i^2\right)^{1/10}\right) + 1.0\right] \qquad (2.9)$$

$$x_i \in [-32.767\ \ 32.768], \qquad n = 2$$

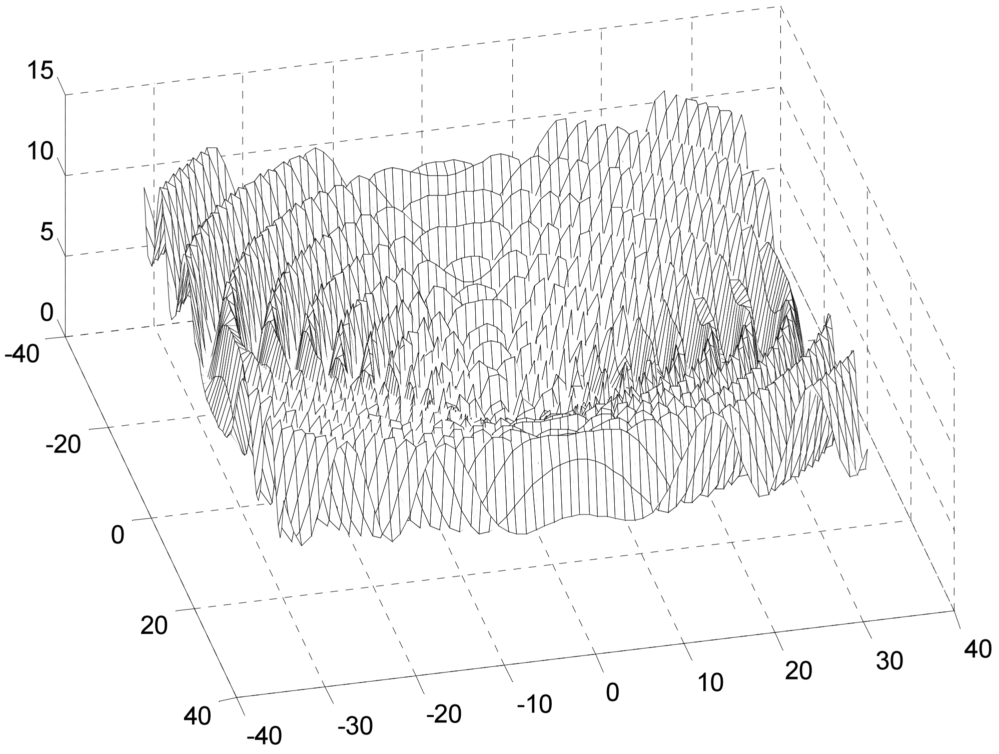


Figure 2.7: Plot of the circle function

Schaffer's binary function is presented in (2.10). The characteristic of this function are easily grasped from it two-dimensional case shown in Figure 2.8. The function is degenerate in the sense that many points share the same global optimal function value (f_{s6}^{*} = 0.99400693). As can be seen in Figure 2.7, the points are located on the highest circle in the crown near the origin.

Maximize $$f_{s_6}(x) = \frac{\sin^2\left(\sqrt{\sum_{i=1}^{n} {x_i}^2}\right)}{1.0 + 10^{-3}\left(\sum_{i=1}^{n} x_i^2\right)^2} \tag{2.10}$$

$$x_i \in [-16.383\ 16.384], \qquad n = 5$$

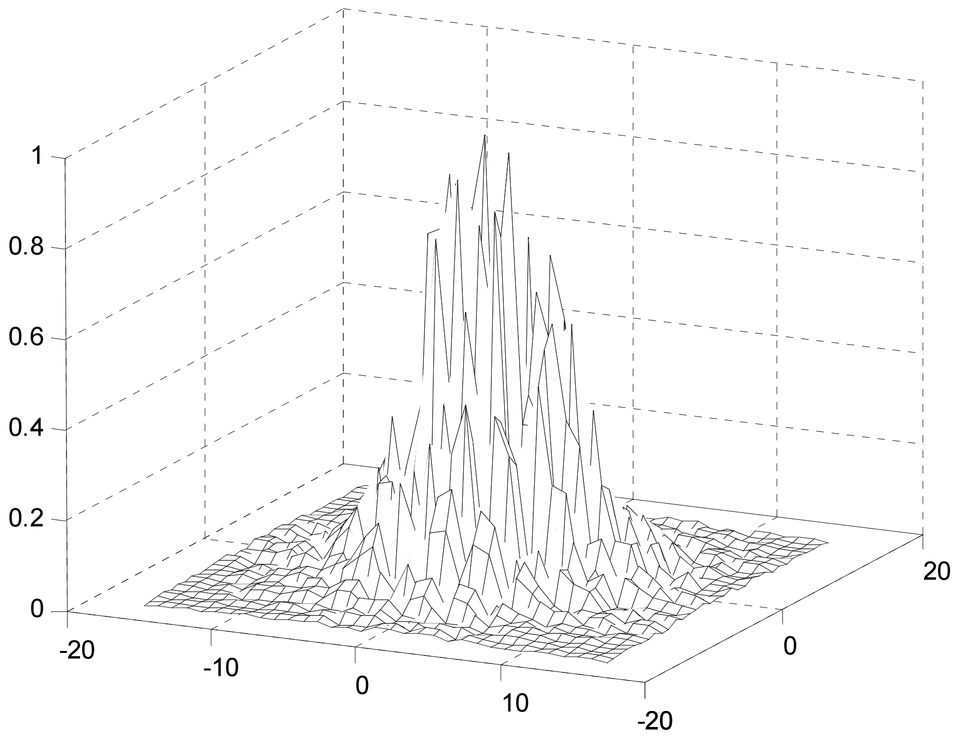


Figure 2.8: Plot of the Schaffer's function with 2-D (n=2)

2.5 Parallel Estimation of Distribution Algorithm with Probability Model Migration

Parallel estimation of distribution algorithms or parallel probabilistic-modeling genetic algorithms is based on the traditional parallel genetic algorithms. However, the key difference is how the EDA manages probability vector instead of individuals in the population. This difference has important effects on how to parallelize the algorithms. In this section, parallel GAs or EDAs with multiple populations is in our attention since the proposed algorithm is developed from it and so many researches has been developed in this class of algorithm. This class of GAs is also called *coarse-grained* or *island model*. The design of multiple-deme parallel GAs involves difficult and related choices. The main issues are to determine (1) the size and number of demes, (2) the topology that interconnect the demes, (3) the migration rate that controls how many individuals migrate, (4) the frequency of migration, and (5) the migration policy that determines which individuals migrate and which are replaced in the receiving demes [46, 47].

Most parallel EDAs concentrate on parallel construction and sampling of probabilistic models in order to speed up the process especially for EDAs with complex model like Bayesian optimization algorithm (BOA) [48, 49]. The idea of the multi-deme estimation of distribution algorithm (PEDAs) based on PBIL algorithm is presented in [50]. In [49], mixture of distribution with Bayesian inference is discussed. The concept of migration of probability parameters instead of individuals was firstly published in [51] where UMDA platform with the convex combination of univariate probability models is investigated for various network topologies. In [52] parallelization of compact GA is presented. Further enhancement of this concept is described in [53] where local search methods are used to identify which parts of migrant model can improve the resident model. Recently, the *island model* of Bivariate marginal distribution algorithm (BMDA) is presented in [54]. In [54], the unidirectional ring topology is used. The cooperation of demes is realized via migration of probabilistic models. It introduces an adaptive learning technique, based on the quality of resident and immigrant subpopulation, which consists of the adaptation of the resident by the incoming neighbor immigrant model. Using parallel EDA based-on compact GA to solve the scalability problem of GA is presented in [55].

2.6 Hardware Implementation of Genetic Algorithms

GA process is a time-consuming. For many real-world applications, GA can run for days, even when they are executed on a high performance workstation. To reduce the execution time of GA, several method have been offered, including parallel and/or distributed processing of GA along with its hardware implementation [56, 57]. A myriad of hardware-based GA have been proposed in recent years. The hardware-based GA can be categorized into three methods: direct implementation from the existing genetic algorithms, parallelized hardware implementation some parts of the algorithm, and new genetic algorithms invented for hardware implementation.

The genetic algorithms and evolutionary algorithms been have implemented in hardware in three areas [58]:

1. A means of implementing the fitness functions of GAs.
2. A platform for implementing the EA/GAs for general optimization problems.
3. An evolutionary engine in intrinsic evolvable hardware

2.6.1 Direct Hardware Implementation of the Basic Genetic Algorithms

For this approach to the hardware implementation of GA, Scott et al. proposed a hardware-based genetic algorithm (HGA), which was implemented on a set of field-programmable gate arrays (FPGAs). The HGA is based on the simple genetic algorithm (SGA) [59].

Yoshida et al. [56, 60] proposed a hardware-based GA called the GA processor (GAP). The GAP is based on a steady-state GA.

Shakleford et al. [61, 62] proposed their original GA. It was called the survival-based GA. It is somewhat similar to the steady-state GA. They implemented a complete GA system using Xilinx XCV3200E chip. Their implementation uses extensive pipelines and parallel fitness evaluation to get performance increase of 320 times when compared to the same algorithm running on a 366Mhz Pentium CPU.

Kajitani et al. [63] proposed GA hardware for evolvable hardware on a single LSI chip.

Wakabayashi et al. [64] also proposed a VLSI implementation of an adaptive GA, called the GA accelerator (GAA) chip as a general purpose GA hardware.

Yamaguchi et al. [65] used an FPGA to implement a coprocessor for evolutionary computation to solve the iterated prisoners' dilemma problem. They reported a 200 times performance speed up in processing the problem on FPGA when compared to a 750MHz Pentium processor.

Graham and Nelson [66] implemented a complete GA system using four FPGAs. Each FPGA was programmed to carry out a different function; selection, crossover, fitness, and mutation. Each FPGA passed its results to the next FPGA, forming a pipeline GA. The performance of their system was compared to a software implementation running on a 125MHz PA-RISC workstation and they gain four times of improvement.

2.6.2 Parallelized Hardware Implementation of the Existing Genetic Algorithm

The early effort to implement the parallel genetic algorithm in hardware is presented in Turton et al [67]. They implemented fine-grained GA for image registration. Each processor node processes only one chromosome. The result was simulated.

Bland et al [57] implemented the genetic operators of the simple genetic algorithm in a systolic array like architecture. Also, four systolic arrays form a macro-pipeline which implements the operators. They proposed the implementation for a system consisted of a general purpose processor augmented with an FPGA.

Tufe and Haddow [64] designed pipeline architecture for the GA called GA Pipeline for intrinsic evolvable hardware. The main functions of GA Pipeline are selection, genetic operation, fitness and sorting which are implemented in separate modules. The pipeline consists of two phases: reproduction and updating.

Cho et al. [68, 69] proposed the genetic algorithm processor (GAP) based on subpopulation architecture. They applied the steady-state GA with modified tournamenet selection, special survival condition and the parallelism of coarse-grain GA. Their implementation keeps a group of best individual in each generation. Ring topology was proposed for interconnection. The Altera FPGA was used to implement one GAP.

Nedjah et al. [70] proposed a massively parallel architecture for hardware implementation of genetic algorithms. They implemented the fitness computation in parallel hardware units which they proposed as a hardware-based neural network. Also, the crossover, mutation and selection are implemented in parallel using multiple crossover, mutation, and selection units. However, this parallel hardware implementation still using large amount of memory because of its implementation is based on a simple genetic algorithm.

Jelodar et. al. [71] designed a parallel GA for an System-on-a-Programmable-Chip (SOPC) called SOPC-based parallel GA. The global GA which has one population was implemented in parallel hardware. The new idea in this paper was to propose using configurable processors with genetics operators together. This approach resembles software-hardware co-design architecture for the global parallel genetic algorithm.

2.6.3 New Genetic Algorithms for Hardware Implementation

To implement GA in hardware from the existing software-based genetic algorithm tends to require more hardware resources and memory. Researchers have proposed solutions to these problems by design new or modified genetic algorithms suitable for hardware implementation [58].

The compact genetic algorithm (CGA) is a probability vector-based evolutionary algorithm that can be efficiently implemented in digital hardware [72]. Even though CGA has advantage for hardware implementation, but unfortunately, the basic CGA lacks of sufficient search power for real world EH applications that require accuracy and faster processing time. Therefore, the CGA is improved by adding more techniques like elitism, mutation, and champion resampling. This modified CGA is called *CGA or *CGA family. Gallagher et al. [58] proposed a family of compact genetic algorithms to be implemented in hardware. They modified a basic compact GA with elitism, mutation, and resampling. The modified compact GA called *CGA performs better than a basic compact GA with reasonable increased hardware resources. The reason behind using the compact GA is that CGA can be implemented in hardware without memory.

To implement a EA based hardware for less search capability, Gallagher also proposed an EA-based hardware called Minipop EA. Recently, MiniPop EA is proposed to be used for EH [73]. The MiniPop EA trades away search-power for the ability to implement the algorithms in small size hardware. However, the *CGA and MiniPop algorithms still perform well on normal EH-control problems [74]. Thus, *CGA and MiniPop currently are the key algorithms for EH applications.

Recently, Zhu et al. [75] presented a new hardware-based GA called OIMGA. OIMGA includes two searches that interact in hierarchical manner, namely a global search and a local search. OIMGA operates on real population in contrary to the compact GA which is probabilistic-based model building GA. The hardware resources used by OIMGA is similar to the basic compact GA but the results shows that OIMGA has more search power than a basic compact GA without elitisms.

2.7 Block-based Neural Network

The block-based neural network (BBNN) was proposed by S. W Moon et al in [76]. The BBNN model consists of a 2-D array of basic blocks. Each block is a basic processing element that corresponds to a feedforward neural network with four variable input/output nodes. Figure 2.9 represents the structure of the BBNN model of *m* x *n* size with *m* row or stage and *n* column labeled as B_{ij} The label *i* denotes the stage. The last stage is denoted *m.* Any block in the BBNN is connected to its neighbors. The first row of blocks B_{11}, B_{12} to B_{1n} is the input layer and the blocks B_{m1}, B_{m2}, ... B_{mn} form the output layer. An input signal $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ propagates through the blocks to produce network output $\mathbf{y} = (y_1, y_2, \ldots, y_n)$.

BBNN can implement both feedforward and feedback network configuration. A feedback configuration of BBNN architecture may cause a longer signal propagation delay. BBNN of size *m* x *n* can represent the input-output characteristics of any Multilayer perception (MLP) network.

A block of BBNN consists of four nodes. These four nodes can be represented by one of the three different types of internal configurations. Figure 2.10 shows three types of internal configurations of one input and three outputs (1/3), three inputs and one output (3/1), and two inputs and two outputs (2/2). The four nodes inside a BBNN block are connected with each other through weights. A weight w_{ij} denotes a connection from node *i* to node *j*. A BBNN block can have up to six connection weights including the biases. For the case of two inputs and two outputs (2/2), there are four weights and two biases. The 1/3 case has three weights and three biases. There are three weights and one bias for 3/1 case. Generalization capability of BBNN network emerges through various internal configuration of a BBNN block.

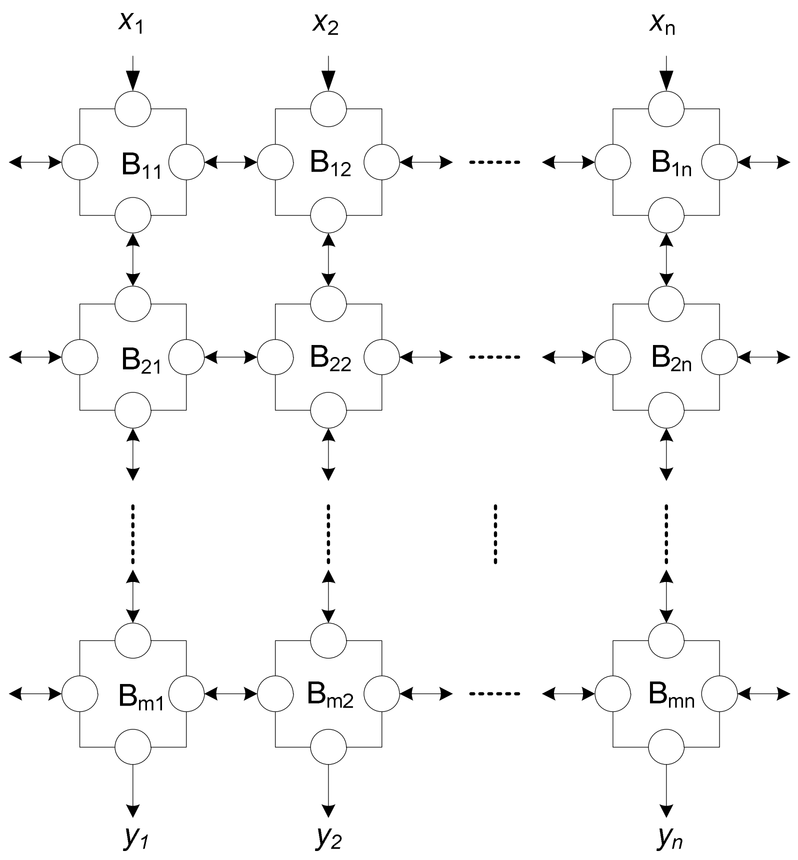


Figure 2.9: Structure of BBNNs

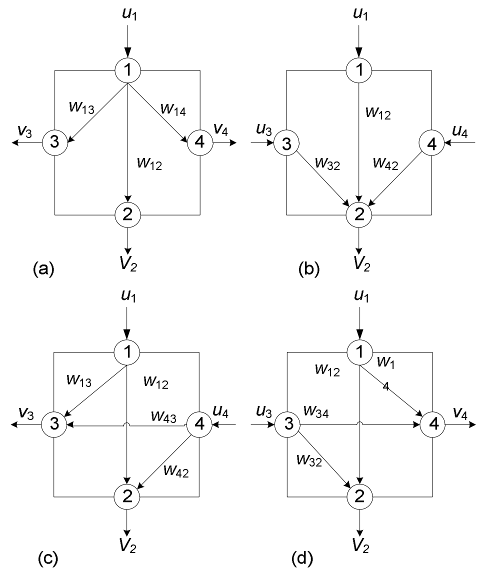


Figure 2.10: Four different internal configurations of a BBNN block (a) 1/3, (b) 3/1, (c) 2/2, (d) 2/2

If signal u_i is the input and v_j is the output of the block. The output v_j of a block is computed with an activation function h as follows.

$$v_j = h\left(\sum_{i \in I} w_{ij} u_i + b_j\right), \qquad j \in J$$

I and *J* are sets for input and output node, respectively. The term b_j is the bias of the *j*th node. The activation function can be linear or nonlinear function. Direct implementation of nonlinear activation function requires more hardware resources. A more practical approach would be a piecewise-linear approximation of the non-linear activation function or using lookup tables.

2.8 Hardware Implementation of Neural Network: Survey

In this section, the digital implementations of artificial neural network are surveyed. The digital neural network hardware implementations are further classified as follows: 1) field-programmable gate array (FPGA)-based implementations, 2) digital signal processor (DSP)-based implementations, and 3) application specific integrated chip (ASIC)-based implementations [77, 78]. DSP-based implementation is like programming on special proposed processors, it offers flexibility but cannot deliver best performance compared to the cost of the devices.

For the past decades, Kuhn et.al presents a good survey of digital hardware for neural network [79]. Dias et. al. wrote a comprehensive survey of commercial hardware of neural network [80]. Zhu and Sutton present a short review of FPGA implementation of neural network [81]. Since ANNs are inherently parallel architectures, there have been several earlier attempts to build custom application specific integrated circuits (ASICs) that include multiple parallel processing units [82]. However, intrinsic to the ASIC design principles, the resulting networks were constrained by size and type of algorithm implemented. More recently, the focus of ANN hardware design shifted toward implementation on reconfigurable hardware [83]. This allows for more flexibility of network size, type, topology, and other constraints while maintaining increased processing density by taking advantage of the natural parallel structure of neural networks. Currently, field-programmable gate arrays (FPGAs) are the preferred reconfigurable hardware platform. Current FPGAs provide performance and logic density similar to ASIC but with the flexibility of quick design/test cycles. Thus, they are superior in research, and, often, industrial applications

Considering digital design techniques, there are two approaches to design digital neural network hardware: bit-serial and bit-parallel approaches. The bit-serial approach has been developed by encoding the weight and inputs to serial bit-stream [84, 85].

In this dissertation, we adopt bit-parallel approach because it is suitable for FPGA implementation since there are more hardware resources in current FPGA devices. There are challenges in design and implement neural network in digital hardware. NN hardware required to implement nonlinear excitation functions and to design suitable hardware architecture to accommodate neurons. It demands higher resources, especially hardware multipliers to implement neurons. Another challenge is the design the suitable arithmetic precision to be implemented in hardware in order to minimize hardware resource. This is called the area versus precision design tradeoff. The tradeoff is to choose, in a data format, the correct balance between the precision required carrying on network functionality, and the size and cost of the FPGA resources consumed.

The traditional implementation of neuron is based on the hardware implementation of multi-layer perceptron (MLP). Figure 2.11 shows the basic structure of MLP. The basic unit of MLP is the neuron. There are generally two ways to implement neural in digital hardware: shared multiply-accumulate (MAC) unit and fully parallel multiply-accumulate. Figure 2.13 and 2.14 show the two types of basic implementation of one neuron.

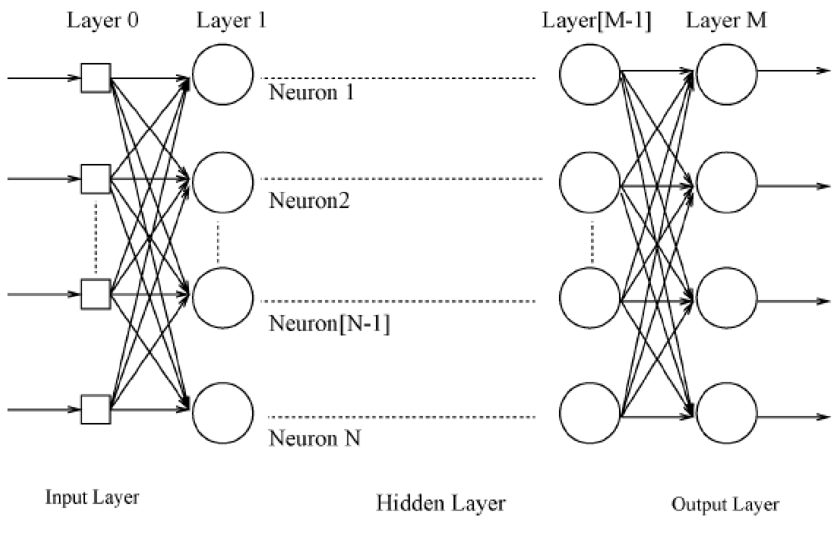


Figure 2.11: Structure of MLP

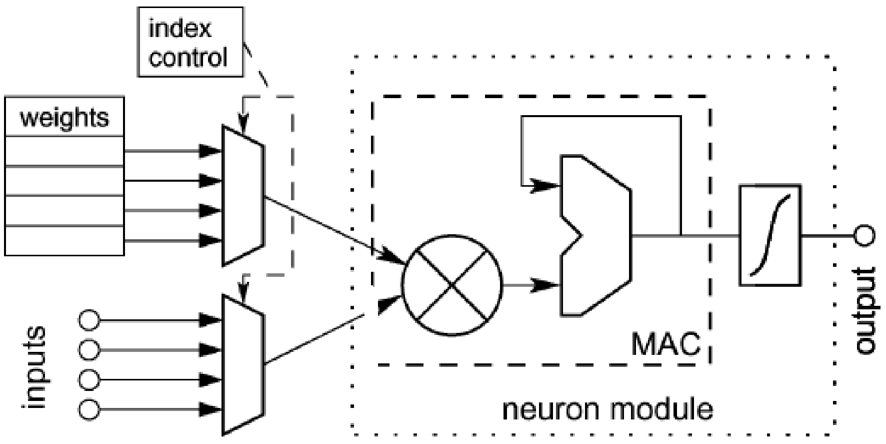


Figure 2.12: Shared multiply-accumulate (MAC) unit

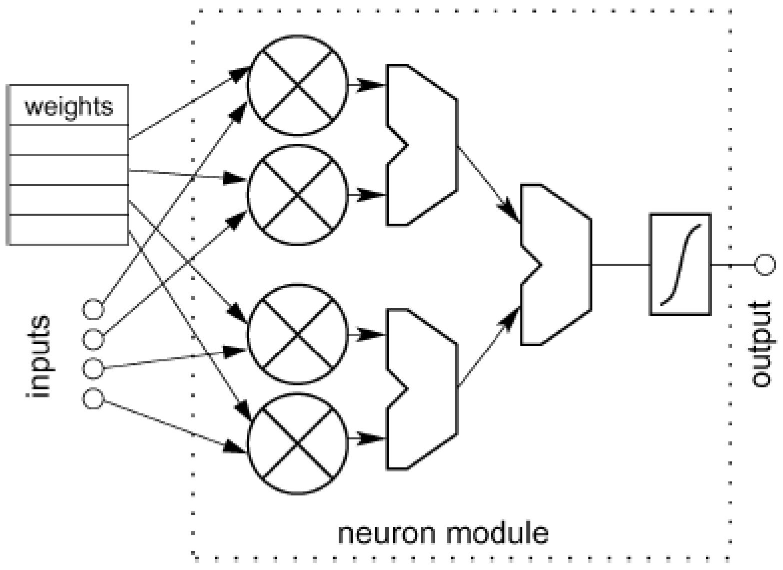


Figure 2.13: Parallel multiply-accumulate (MAC) unit

Numerous works have reported new multiplication algorithms for NN [86-90]. In [86], to avoid the use of multipliers to reduce resource, the weights of the NN have been constrained to integer powers of two. Constraint on weights leads to inferior performance of the network. Distributed arithmetic (DA) has been widely used to improve hardware resource efficiency for multiplication. Lookup tables (LUTs) are used to store the excitation function in order to improve speed and reduce the resource requirement. In addition, implementation of dedicated NNs with few numbers of neurons has been reported [90]. An NN with eight neurons for control of inverted pendulum has been implemented using Xilinx FPGA interfaced to an 8031 microcontroller and a 16-K random access memory (RAM) [91]. A real-time controller with proportional integral derivative (PID) control algorithms implemented in FPGA and NN controller implemented in DSP board have been reported [92]. In [93], the radial basis function neural network and back-propagation algorithm are implemented with floating-points processor in FPGAs. For hardware implementation, back-propagation algorithm is the key algorithm [94].

Even though with high density FPGAs, there are not limitless resources in FPGAs when implementing an ANN and the design poses a number of challenges. One is to determine the most efficient arithmetic representation format. While most general computing microprocessors/software implementations currently implement single (32 bit) and double (64 bit) IEEE floating-point (FLP) formats (IEEE 754-1985) using these formats on FPGAs requires significant resources. On the other hand, using shorter integer, FLP or fixed-point (FXP) formats [78], which consume less FPGA area to process, often means loss of precision. Depending on the data format chosen, efficient implementation of MLP-BP on FPGAs can result in completely different outputs than those of a similar architecture implemented in software using the IEEE FLP formats.

2.9 On-line ECG Signal Classification

The electrocardiogram (ECG) is the electrical manifestation of the contractile activity of the heart and can be recorded with the surface, noninvasive electrodes placed on limbs and chest. Any disturbance in the regular rhythmic activity of the heart (amplitude, duration, and the shape of rhythm) is termed arrhythmia. Physicians interpret the shapes (morphology) of the ECG waveform and decide, whether the heartbeat belongs to the normal (healthy) sinus rhythm or to the appropriate class of arrhythmia. Generally, arrhythmias can be divided into two groups. The first group includes ventricular fibrillation and tachycardia which are life-threatening and require immediate therapy with a defibrillator. Detection of these arrhythmias is well researched and successful detectors have been designed with high sensitivity [95].

In this thesis, the second group which includes arrhythmias that are not imminently life-threatening but may require therapy to prevent further problems is the subject of the studies. Figure 2.14 shows the typical ECG signal with three indicated waves: the P, QRS, and T. The P wave is the result of slow-moving depolarization (contraction) of the atria. This is the low-amplitude wave of 0.1-0.2 mV and duration of 60-120ms. The wave of stimulus spreads rapidly from the apex of the heart upwards, causing rapid depolarization (contraction) of the ventricles. This results in the QRS complex of the ECG, a sharp biphasic or triphasic wave of about 1mV amplitude and approximately last 80-100ms duration. The plateau part of action potential of about 100-120 ms after the QRS is known as the ST segment. The repolarization (relaxation) of the ventricles causes the slow T wave with an amplitude of 0.1-0.3 mV and duration of 100-120 ms. Between T and P wave there is a relatively long plateau part of small amplitude known as TP segment.

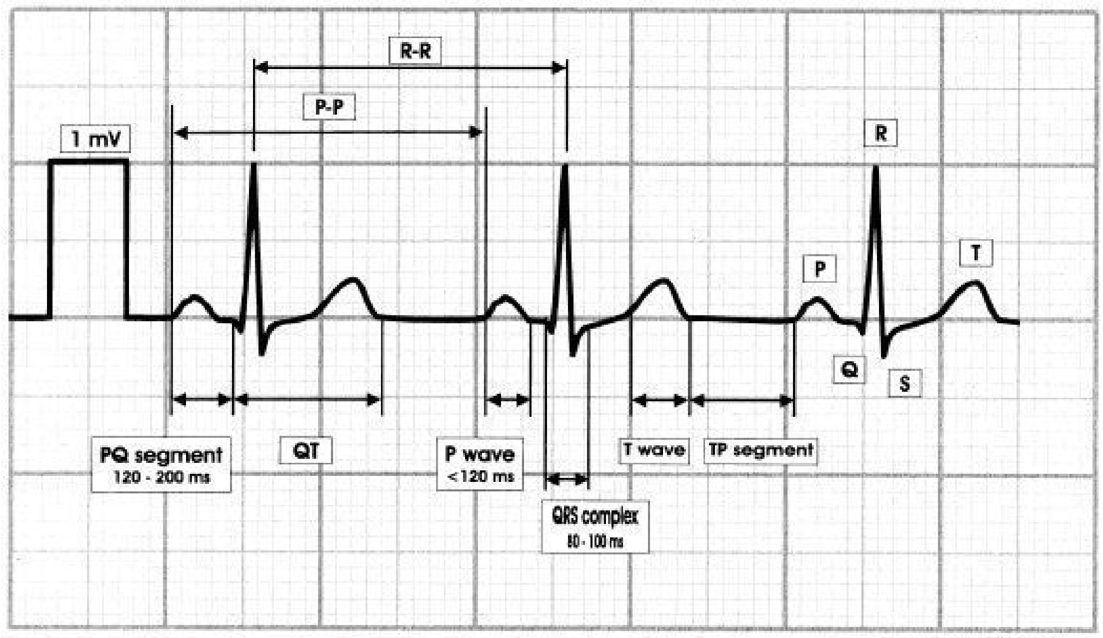


Figure 2.14: Typical ECG waveform.

Many arrhythmias manifest as sequences of heartbeats with unusual timing or ECG morphology. An important step towards identifying an arrhythmia is the classification of heartbeats. Figure 2.15 shows the real recorded ECG signals corresponding to normal beats (N) and premature ventricular ectopic beat (V). The rhythm of the ECG signal can then be determined by knowing the classification of consecutive heartbeats in signal. Classification heartbeats can be very time-consuming and hence any automated processing of the ECG that assists this process would be much needed [96].

The ECG arrhythmia waveforms differ usually with the amplitude and duration of beats. Especially, the QRS complex is the important part. QRS complex reflects the electrical activity within the heart during the ventricular contraction, the time and shape of it occurrence provide much information about the current state of the heart. Due to its characteristic shape, it serves as the basis for the automated determination of heart-rate and diagnosis of cardiovascular diseases. QRS detection provides the fundamentals for almost all automated ECG analysis algorithms.

Within the last decade many new approaches to QRS detection have been proposed. The key algorithm is proposed in [97]. In this thesis, we focus on the classification of ECG beat. The algorithm in [98] is used for QRS detection.

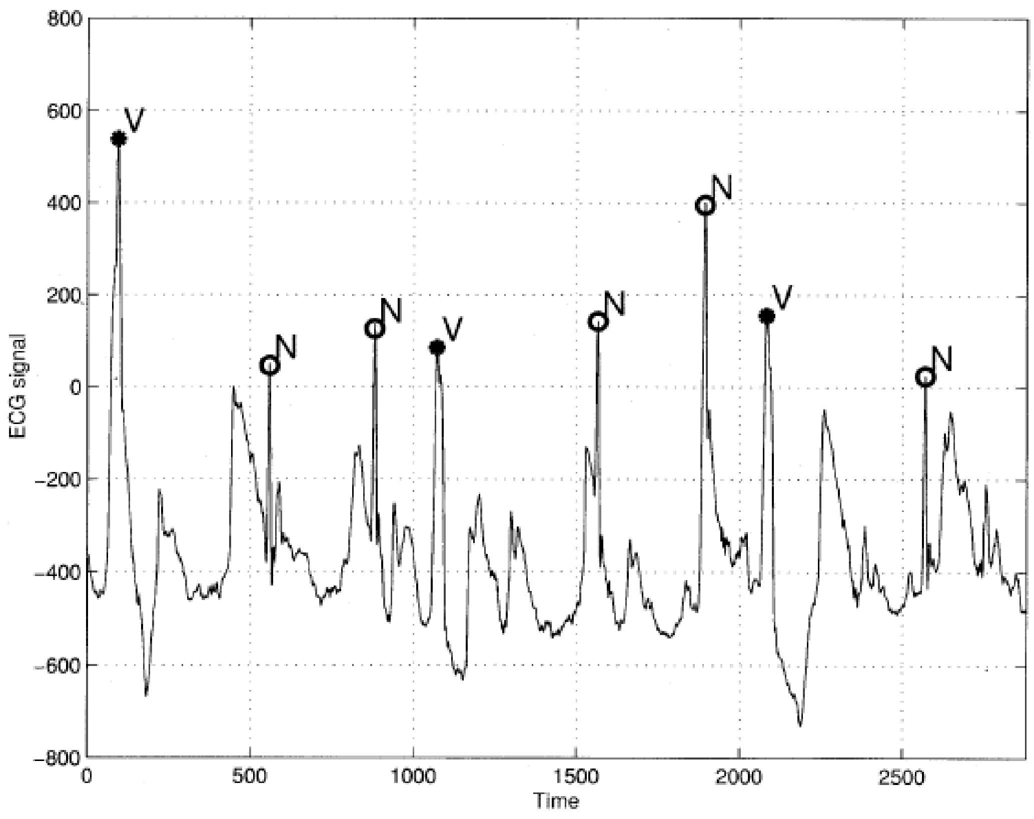


Figure 2.15: Examples of real ECG recording of two different rhythm types

As shown in Figure 2.16, it is divided into a preprocessing or feature extraction stage including linear and nonlinear filtering and decision stage including peak detection and decision logic.

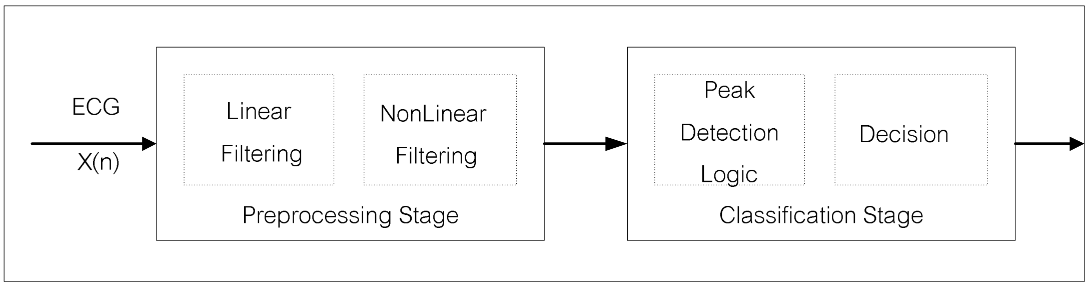


Figure 2.16: Common structure of the QRS detectors.

Within the last decade many new approaches to automatic ECG signals classification have been proposed. In [99-102], the authors present use the Hermite polynomial for feature selection. The method in [99] is based on a hybrid fuzzy neural network that consists of a fuzzy self-organizing network connected in cascade with a multilayer perceptron. In [103], a neuro-fuzzy approach is described. In [104], the authors implemented two classification systems based on support vector machine (SVM) approach. The first uses feature derived from high-order statistics, while the second uses the coefficients of Hermite polynomials. Detection of premature ventricular contractions (PVCs) by means of a fuzzy-neural network classifier with feature derived from a quadratic spline wavelet transformed is proposed in [105]. In [105], different classification systems based on linear discriminant classifiers are explored, together with morphological and timing features. A high spectral analysis techniques is proposed in [106]. In [107], an automatic online beat segmentation and classification system based on a Markovian approach is proposed. In [108], a rule-based rough-set decision system in time domain is presented. In [109], a patient-adapting heartbeat classifier system based on linear discriminant is proposed. The classification system processes an incoming recording with a global classifier to produce the first set of beat annotations. Then, the expert can validate the records. The system then adapt by first training a local classifier using the newly annotated beats. The wavelet and timing features are used to for training using a large data set [110]. The author found that fourth scale of a dyadic wavelet transform with a quadratic spline wavelet together with the pre/post RR-interval ratio is very effective in distinguishing normal and PVC from other beats. In [101], Moon S. and Kong S. proposed the block-based neural network classifier with feature selection using coefficient of Hermite polynomial and RR intervals. The block-based neural networks are trained by evolutionary algorithm and a modified back-propagation algorithm. The authors found a competitive classification results compared to other methods. The support vector machine and particle swarm optimization is proposed in [111]. This method is effective for a limited number of training beats. Those methods that have been presented so far are suitable for implemented on software. In [112], a portable ECG classification system is presented.

The Association for Advancement of Medical Instrumentation (AAMI) has a recommendation for reporting performance results of cardiac rhythm algorithms [113]. AAMI standards are adopted in this thesis. The AAMI recommends that each ECG beat be classified into the following five heart beat types: N (beat originating in the sinus node), S (supraventricular ectopic beats), V (ventricular ectopic beats), F (fusion beats), and Q (unclassifiable beats) [113]. The waveforms of ECG beats from the same class may differ significantly and different types of heartbeats sometime share similar shapes.

Several standard ECG databases are available for the evaluation of QRS detection algorithms. Tests on these well-annotated and validated databases provide reproducible and comparable results. These databases include MIT-BIH database, AHA database, and CSE database. In this thesis, MIT-BIH is used. The MIT-BIH database [114] provided by MIT and Boston's Beth Israel Hospital consists of ten databases for various test purposes; i.e., the Arrhythmia database, the noise stress test database, the ventricular Tachyarrhythmia database. In addition to the AHA database and the Euro-pean ST-T database, the first MIT-BIH databases are required by the ANSI for testing ambulatory ECG devices. Most frequently the MIT-BIH arrhythmia database is used. It contains 48 half-hour recording of annotated ECG with a sampling rate of 360 Hz and 11-bit resolution over a 10-mV range.

CHAPTER III
CELLULAR COMPACT GENETIC ALGORITHMS

Cellular compact genetic algorithm is developed from the cooperative compact genetic algorithm [115] and parallel GAs [46]. The concept of cellular compact genetic algorithm is to parallelize or divide a large problem into smaller tasks and to solve the task simultaneously using multiple genetic algorithms. CCGA is different from a traditional parallel GA since it operates on probability vectors. CCGA is a parallel univariate estimation of distribution algorithms (EDAs) that migrates the probability model instead of individuals [48]. CCGA improves model combination through local search by selecting the better model from neighbors to be combined with the inner model of the cell [53]. The CCGA consists of uniform cellular compact genetic algorithm cells connected in a cellular automata space. Each CGA cell only exchange probability vectors to its neighbors. In the following section, we describe the cooperative compact genetic algorithm, which is the prior version of CCGA. The topology and the algorithm of CCGA are also described.

3.1 Cooperative Compact Genetic Algorithm

For cooperative GAs, the search space can be partitioned by splitting the solution vectors into smaller vectors [116]. Each of these smaller search spaces is then searched by a separate GA. The fitness is evaluated by combining solutions found by each of the GAs representing the smaller sub-spaces. In this section, we present a cooperative approach for CGA. The thrust is to propose the concept of *confident counter* that guides toward search direction along with the traditional probability vectors. The cooperation that we proposed comes from using confident counter as a source of shared knowledge on where is the best search direction because only exchanging the probabilities vectors is not enough to guarantee which one of probability vectors from each neighboring cell is the current best vector. In the cooperative compact genetic algorithm (CoCGA), the individual CGA cell searches in its own sub-population.

The implementation of the coevolutionary genetic algorithm is similar to the concept of parallelized GAs described in the previous chapter. Targeting hardware implementation, we propose CoCGA to be applied to the evolvable hardware. In our case, the individual CGA cell sends its search result through its probability vector together with its confident counter to the leader cell. The leader cell consolidates this information and decides the bias for the search direction which it then sends back to all neighboring CGA cell as the new current best probability vector.

3.1.1 CoCGA Topology

Figure 3.1 shows the topology of the cooperative compact GA (CoCGA). The topology of the proposed CoCGA resembles the cellular automata (CA) systems that cells only interact with their neighbors. However, the interactions between CA cells occur by exchanging the probability vectors instead of mating between individuals of sub-population directly. With this proposed CA topology, the hardware realization of the algorithm is straight forward and not too complicated to be implemented, considering in term of scalability and signal wiring that greatly contribute to the performance of the hardware circuit. In addition, CA architecture is proposed for EH that has capability of self-evolving and self-replicating [117]. Moreover, CA-like architecture can be practically and efficiently implemented into FPGAs or other reconfigurable devices that their architectures consist of array of logic blocks [118]. Therefore, CA-like architecture is proposed for CoCGA. Each coarse grained CoCGA cell has a probability vector and sub-population. There is a group leader for each group of these coarse grained CoCGA cells. In Figure 3.1, the four-neighbor cell with the leader cell in the middle cell is shown. The probability vectors are exchanged through the leader cells. Each cells except the leader cells keep adjusting their own probability vector to the best probability. The confidence counter (CC) is introduced to help the group leader evaluates which probability vectors from its neighbors are likely to converge to a good solution.

3.1.2 CoCGA Algorithm

Figure 3.2 shows pseudocode of the normal CoCGA cell. After probability vectors of each cell is initialized to the mid-point range, two individuals are generated from the probability vector, then compete similar to a normal compact GA. The proposed algorithm is different from the normal compact GA in two ways: (1) the probability vectors are passed to the group leader cells. (2) the confidence toward the better probability vector is calculated as confident counter passed to the group leader cells. In Figure 3.2, the step 3.2 and 4 are inserted into the normal Compact GA.

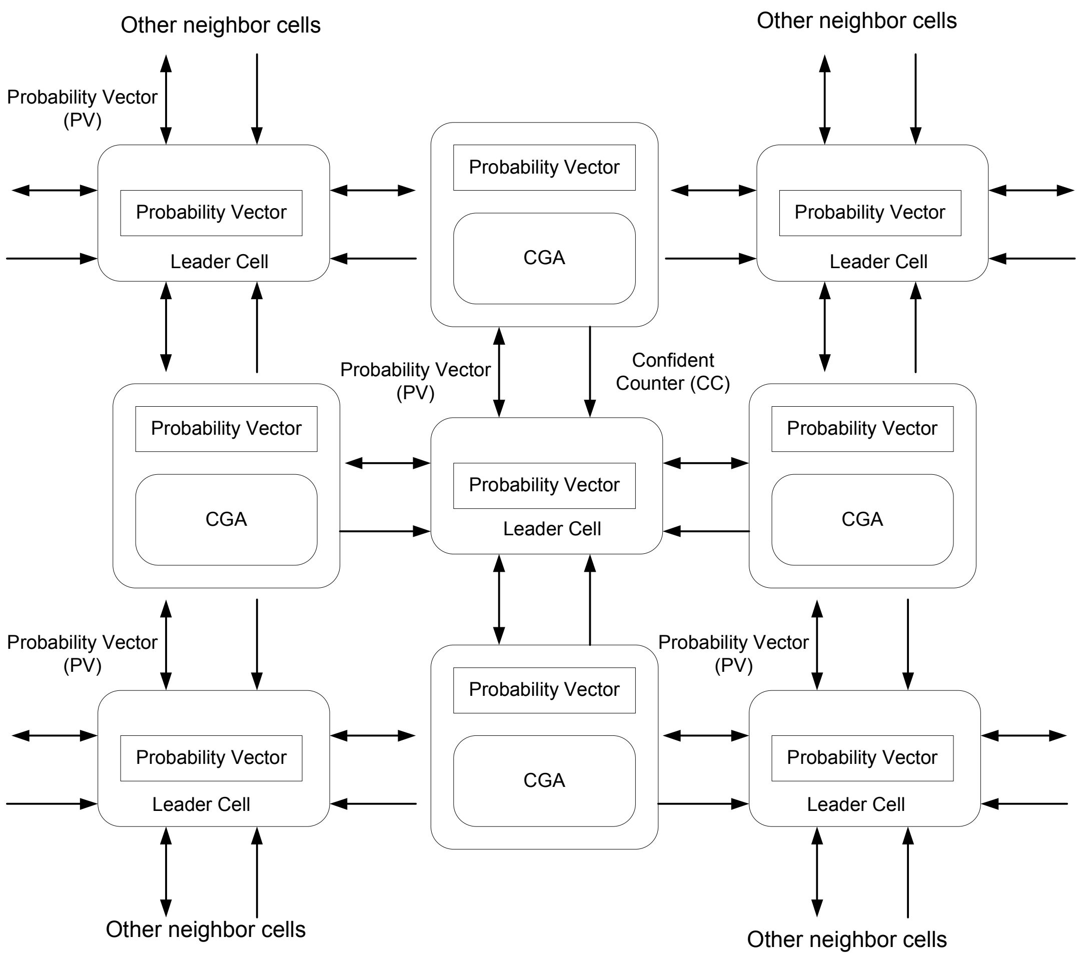


Figure 3.1: Topology of Cooperative Compact GAs with the leader in the middle

The group leader cell only keeps the probability vector but does not implemented the normal compact GA. The group leader updates the probability vectors of its neighbor cells asynchronously because the updating process will occur after the confident counters of the neighboring cells get the new values. For each neighboring cells, the confident counter is incremented asynchronously because it depends on when the current probability vector of each sub-population gives the current best individual. During search process if the better individual is found the confident counter is incremented. Therefore, the group leader uses an asynchronous updating policy. The group leader keeps checking if confident counter (cc) for each one of its neighbor are updated to higher value. Once the confident counter for one of its neighboring updated, then the group leader evaluates value of each confident counter and identifies the current highest value in step two. Next, in step three, the group leader, update its own probability vector with the vector from the neighbor that has the highest confident counter. Then, the new best probability vector is passed from the group leader to its neighbors in step four.

3.1.3 CoCGA Algorithm Benchmark Problems and Experimental Results

We used One-Max and the De Jong test functions (F1, F2, F3) to compare the performance of the CGA and the proposed CoCGA. For De Jong's test functions, the solution quality is measured by the objective function value. The function evaluations are performed by modules coded in Verilog-HDL using behavioral modeling with the same precision as described in [119]. For our experiment, the CoCGA has two neighboring cells and one leader cell. The reason to use De Jong's test functions because the De Jong's test function were originally proposed as a means to measure the abilities of search algorithms and used in [68]. The functions are fully described in [119]. We used 32-bit chromosome length for One-Max problem. For De Jong test function F1 and F2, we used 30-bit chromosome length. For F3 function, the 50-bit chromosome length was used.

De Jong's functions F1-F3 are shown below:

$$F1(x) = \sum_{i=1}^{3} x_i^2, \qquad -5.12 \leq x_i \leq 5.12 \tag{3.1}$$

$$F2(x) = 100\,(x_1^2 - x_2)^2 + (1 - x_1)^2, \quad -2.048 \leq x_i \leq 2.048 \tag{3.2}$$

$$F3(x) = \sum_{i=1}^{5} integer\,(x_i), \qquad -5.12 \leq x_i \leq 5.12 \tag{3.3}$$

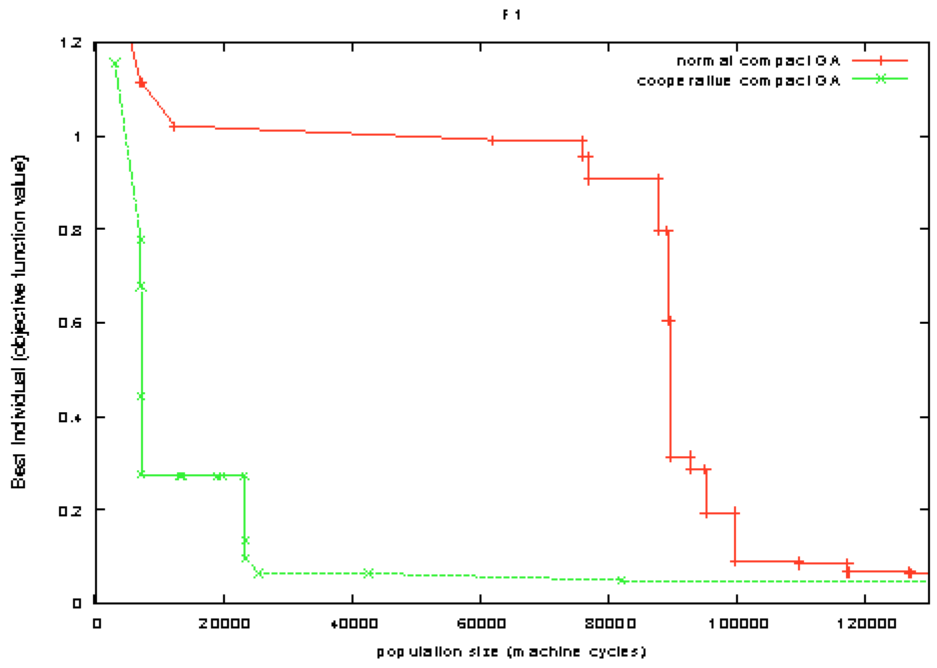
In our experiment, the two neighboring cells and one leader cell are used to validate the efficacy of the proposed approach. We coded both CGA and CoCGA in Verilog-HDL. The simulation was run on Pentium 4 machine with 512MB memory.

```
L is chromosome length
M is number of neighbor cells
cc is Confident Counter
CA is Cellular Automata space

for each cell I in CA do in parallel
    Initialize each p[I]
        For  i   := 1  to L do
            $p_I$[i] := 0.5;
end parallel for
for each group leader cell  i  in  CA do in
parallel
  while not done do
    1.  Read cc from neighboring cells

    2.  Select the highest cc of all neighbors
          $cc_{max}$ := 0;
          for i := 1 to M do
            if (cc[i] > $cc_{max}$ )
                $cc_{max}$ := cc[i];
    3.  Update $p_1$ with $p_{cc}$ with $cc_{max}$
          for i := 1 to L do
            $p_1$[i] := $p_{ccmax}$[i]
    4.  Update new updated $p_l$ to all normal Cell
          for each neighbor cell  of leader cells
          do in parallel
            for i := 1  to L do
              p[i] := $p_1$[i];
          end parallel for
    5.  Check if the vector has converged
          for i := 1 to L do
            if p[i] > 0 and p[i] < 1 then
            goto step 1
    6.  $p_l$ represents the final solution
  end while
end parallel for
```

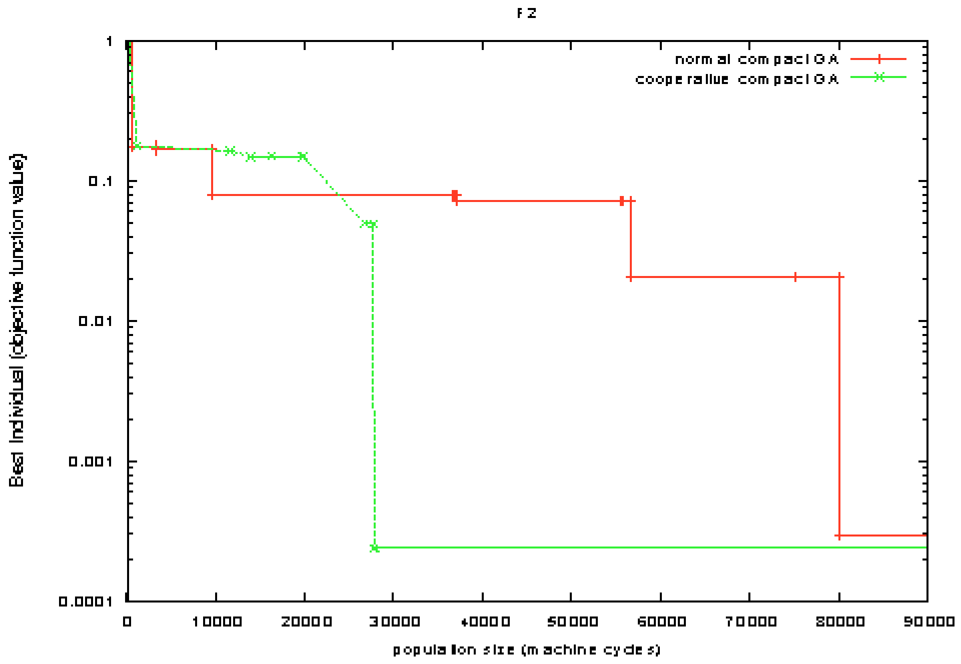
```
L is chromosome length
N is population size
cc is Confident Counter
CA is Cellular Automata space

for each cell I in CA do in parallel
    Initialize each p[I]
        For  i  := 1  to L do
            [i] := 0.5;
    Initialize cc
            cc := 0;
end parallel for
for each cell  i  in  CA do in parallel
  while not done do
    1.  Generate two individual from the vector
            a := generate ();
            b := generate ();
    2.  Let them compete
            Winner, loser := compete (a, b);
    3.  Update the probability vector toward
        better one and
        Increment Confident Counter
        3.1. Update probability vector
          for i := 1 to L do
          if winner[i]  != loser[i] then
            if winner[i] = 1 then
              p[i] += 1/N
            else p[i] -= 1/N

        3.2 Increment Confident Counter
          if winner[i]  != loser[i] then
            cc: = cc + 1;
    4.  Check if cc is incremented then
        Send p and cc to the group leader cell
    5.  Check if the vector has converged
          for i := 1 to L do
            if p[i] > 0 and p[i] < 1 then
            goto step 1
    6.  p represents the final solution
  end while
end parallel for
```

Figure 3.2: Pseudocode of CoCGA (left for leader cell, right for normal cell)

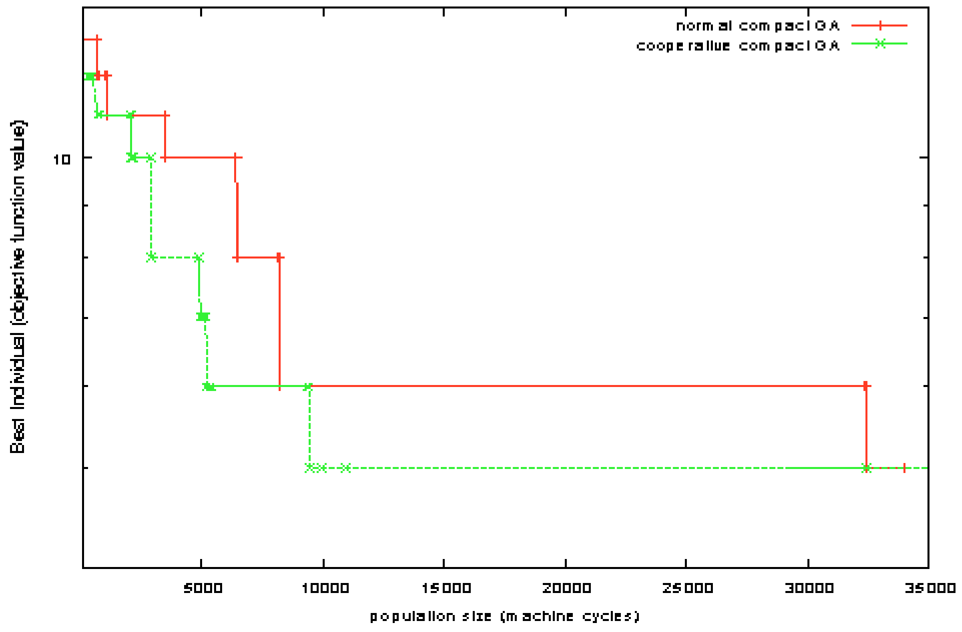
The ModelSim Verilog-HDL, an industrial strength simulator, from Mentor Graphics was used to perform the simulation. We evaluated CoCGA by comparing its performance with the performance of a normal CGA. The graphical plot in Figure 3.3 shows the best individual, the maximum value for One-Max problem and minimum value for F1-F3. Both algorithms were terminated when each of them converged to minimum values. In all cases CoCGA significantly outperformed the normal compact GA both in the minimum values found and in the speed of convergence. The Figure 3.3 shows significance of the experimental results. Table 3.1 shows the speedup comparison between CGA and CoCGA. CoCGA with two neighboring cells is at least three times faster than CGA.


F1
normal compact GA
cooperative compact GA
Best Individual (objective function value)
population size (machine cycles)
0
0.2
0.4
0.6
0.8
1
1.2
20000
40000
60000
80000
100000
120000

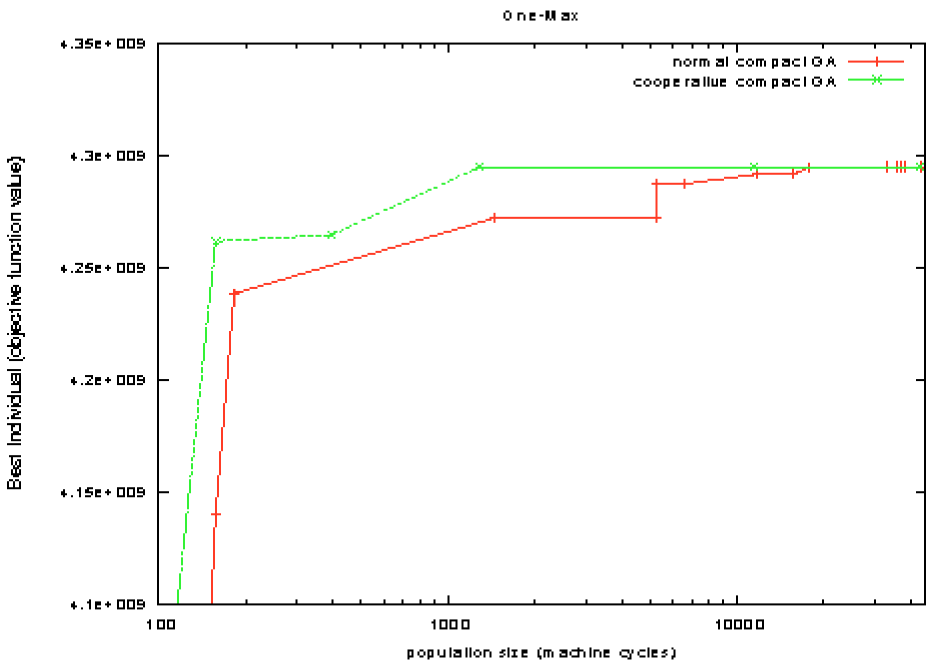
(a) F1


F2
normal compact GA
cooperative compact GA
Best Individual (objective function value)
population size (machine cycles)
0.0001
0.001
0.01
0.1
1
0
10000
20000
30000
40000
50000
60000
70000
80000
90000

(b) F2



(c) F3



(d) One Max

Figure 3.3: Comparison of CGA and CoCGA performance

Table 3.1: The performance comparison in term of number of clock cycles

	One -Max	F1	F2	F3
CGA	43362	126967	80027	32427
CoCGA	11492	25542	27757	9407
Speedup	3.77	4.97	2.88	3.44

3.1.4 Hardware Design and Implementation of CoCGA

For performance evaluation of the proposed CoCGA in comparison with the normal compact GA in hardware, we designed and implemented two hardware circuits for the normal compact GA and the proposed CoCGA into synthesizable Verilog HDL codes and implemented into an FPGA chip. The CoCGA was designed by adding additional modules to the hardware of the normal CGA hardware. By designing our own hardware for both the CGA and CoCGA, we can perform performance comparison fairly since both hardware circuits designed by the same designers and based on the same Verilog HDL codes.

CGA hardware design for a normal compact GA is similar to the design in [58], Figure 3.4 shows CoCGA bit module that consists of the following blocks:

1) RNG is the random number generator.

2) PV is register that keeps the probability vector.

3) GEN_A and GEN_B: The random number generator RNG provides the random number in 8-bit to GEN_A and GEN_B. If the 8-bit value from RNG greater than value of PV, the output of GEN_A and GEN_B will be "0". In the other hand, if value from the random generator is less than the value of probability vector, then the output of GEN_A and GEN_B will be "1".

4) UPDATE PV: The result from the fitness evaluation controls the UPDATE_PV to increment or decrement the register that keeps probability vector.

5) BestFitNess is a register that keeps the current best fitness.

Figure 3.5 shows the hardware design of the N-bit module of the CoCGAEach cell. CoCGA bit-module is based on the designs proposed in [58, 72] integrated with the communication unit (COMM) and the confident counter unit (CC). In Fig. 6, the hardware design consists of three main blocks. The first block is the CoCGA bit-module which can be cascaded to form N-bit chromosome. The second block is the additional units for CoCGA which has the confident counter (CC) and the communication unit COMM. The third block is a finite state machine acts as the main controller for the whole block. The detail of these three additional modules is described below:

1) COMM is a finite state machine that controls the process of sending and receiving the probability vector as parallel 8-bit package to and from the normal cells to the lead cell. For a chromosome of N-bit length, the compact GA needs to have N-bit of probability vector which each bit of the probability vector sizes 8-bit. Thus, for N-bit length chromosome, N packages of parallel 8-bit will be sent and received between the lead and normal cells by the COMM units of each cell.

2) CC is the confident counter designed as a 5-bit counter. During fitness evaluation, the counter is incremented every time when the fitness of the winner is better than the current best fitness which is inside the fitness evaluation block. The value of the counter is passed to the lead cell with the current probability vector.

3) FSM CONTROL is a finite state machine that controls and synchronizes COMM, CC, and the Bit Module.

We implemented the CGA and CoCGA in Verilog-HDL and synthesized he code into FPGA. For CoCGA, we used one leader cell and two neighbor cells. The verilog design was then synthesized and implemented in to Xilinx Vertex-4 FPGA. Table II shows the FPGA implementation results after the codes synthesized, placed and routed using Xilinx software. From Table II, we can notice that the CoCGA cells only require slightly higher hardware resources than normal CGA cell. This is because we only added the communication unit and confident counter unit to the original implementation of CGA [58, 72].

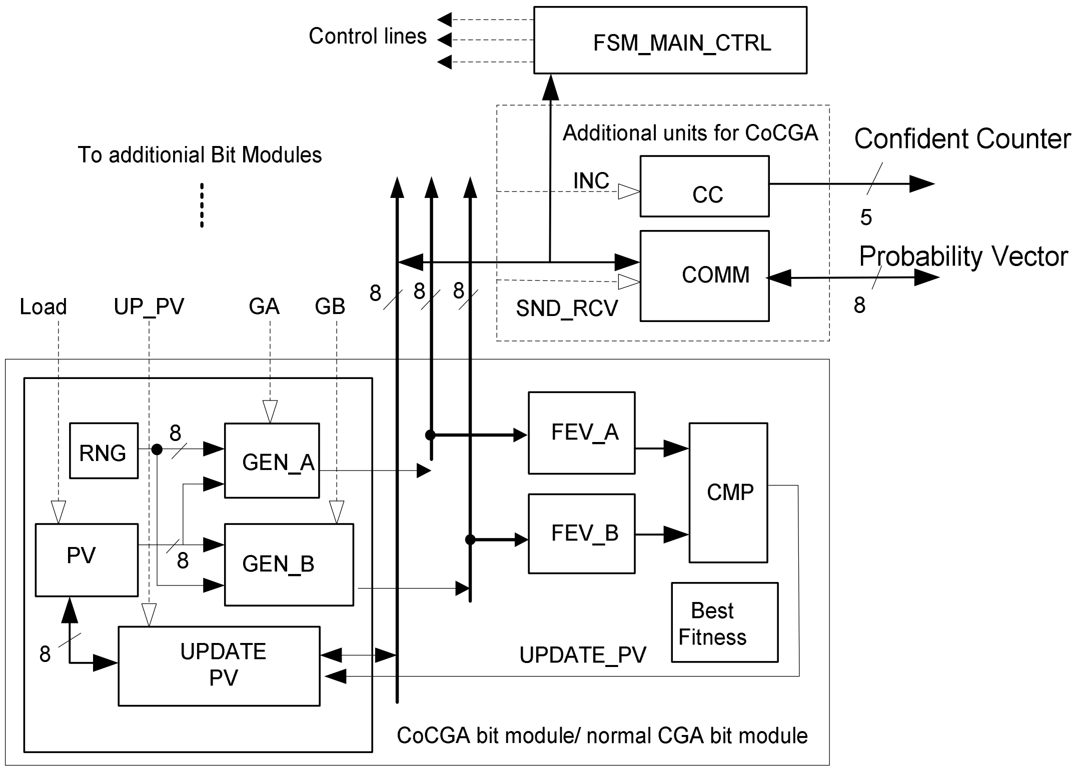


Figure 3.4: Hardware block diagram of a normal CoCGA cell

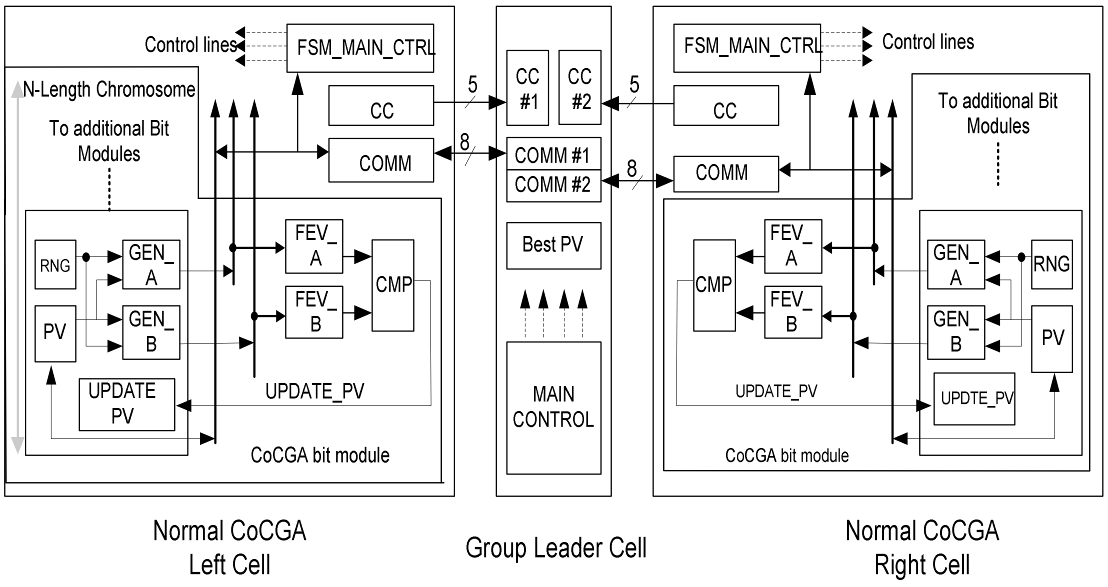


Figure 3.5: Hardware design of two-neighbor cells

Table 3.2: The performance comparison in term of FPGA hardware resources.

	CGA	CoCGA Normal cell	CoCGA Leader cell
Family	Vertex-4	Vertex-4	Vertex-4
Device	Vlx25-sf363	Vlx25-sf363	Vlx25-sf363
No. of Flip Flops	541	598	168
4-input LUT	1065	1295	359
Total equivalent gate count	12602	17034	4651
Maximum Frequency	136.325Mhz	134.421 Mhz	145.423 Mhz

3.2 Cellular Compact Genetic Algorithm

Cellular compact genetic algorithm is developed from cooperative compact genetic algorithm described in the previous section and parallel GAs [46]. The concept of cellular compact genetic algorithm is to parallelize or divide a large problem into smaller tasks and to solve the task simultaneously using multiple genetic algorithms. CCGA is different from a traditional parallel GA since it operates on probability vectors. CCGA is a parallel univariate estimation of distribution algorithms (EDAs) that migrates the probability model instead of individuals [50]. CCGA improves model combination through local search by selecting the better model from neighbors to be combined with the inner model of the cell [53]. Being difference from the cooperative compact GA, the CCGA consists of uniform cellular compact genetic algorithm cells connected in a cellular automata space by each CGA cell only exchange probability vectors to its neighbors. In this section, the topology and the algorithm of CCGA are also described.

3.2.1 CCGA Topology

Figure 3.6 illustrates the topology of the cellular compact GA. The topology of the proposed CCGA resembles the cellular automata (CA) system that cells only interact with their neighbors [117]. When each local CA cells in cellular automata space operates together, the global states of computation can emerged [117]. With this proposed CA topology, the hardware realization of the algorithm is straight forward and can be practically and efficiently implemented into FPGA because of the architecture of array of logic block [118].

Each coarse grained CCGA cell has a probability vector which represents a sub-population. Every CCGA cell is identical. In Figure 3.6, Each CCGA cell with four neighbors exchanges probability vectors and key information between its neighbors. Every CCGA cell keeps adjusting its own probability vector to the better probability. The confidence counter (CC) is introduced to help each cell evaluates how to recombine the probability vectors coming from its neighbors. The key parameters for CCGA topology is the number of the neighbors of each cell.

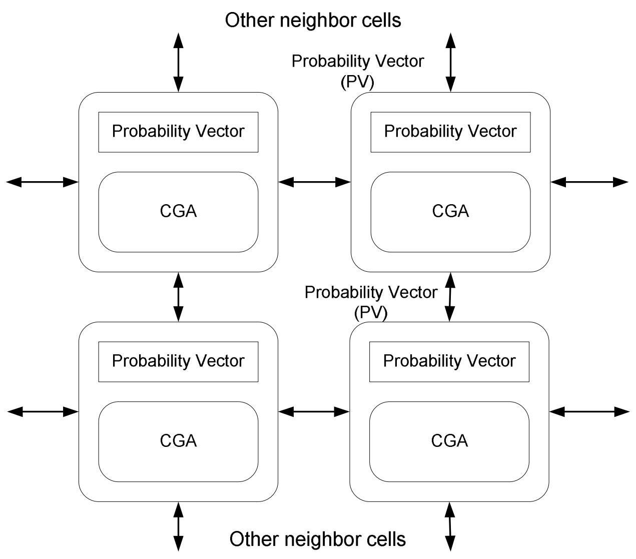


Figure 3.6: Topology of cellular compact GA

3.2.2 CCGA Algorithm

The concept of the CCGA is the cooperative compact GA [115] which is based-on the compact GA. However, the compact GA does not provide acceptable solutions to difficult problems like deceptive problems or multimodal problems, because it does not have memory to retain the required knowledge about non-linearity of the problems. The compact GA with higher selection pressure was proposed in [45] and by adding elitism to the compact GA. In summary, the elitism-based compact GAs can be described into two approaches as shown in Figure 3.7 and Figure 3.8. The non-persistent compact GA performed slightly better than persistent approaches [45]. To apply the elitism compact GA to our cellular compact GA, we modified the non-persistent compact GA from [54] in step 3 by simply discarding the elite chromosome when the allowable length of inheritance reached. With this approach hardware implementation will be more efficient.

Figure 3.9 shows the pseudocode of the cellular compact GA. Each cell of the CCGA has the identical algorithm as shown in Figure 3.9. For each cell, one bit of the GA is represented by a probability vector. There are eight steps in the algorithm. Since the CCGA employs elitism-based compact GA, all of the eight steps are modified from the standard compact genetic algorithm. At first, the probability vector of each CCGA cell in the cellular automata space is initialized to the initial values that can be set to same for every cell to the mid-point range or set to different initial values for different cells in order to assign each CCGA cells to start searching in different area of the problems. This strategy can be applied to more difficult problems and more number of CCGA cells.

In the first step, for each generation, one individual is generated from the probability vector except for the first generation that newly generated individual is assigned to be the elite chromosome. To avoid premature convergence of strong elitism, CCGA employs non-persistent or re-sampling of the current elite chromosome from the initialized probability vector. Thus, a parameter η is introduced as the allowable length or the allowable generations of the elite chromosome to be passed on. This parameter retrains the length of inheritance or the number of generations that the elite passed on, thereby avoid the populations to reach equilibrium very fast and maintain genetic diversity.

```
Parameter:  e : elite chromosome,
            n : new chromosome

2.  Generate one individual from the probability
vector
    If the first generation then
        e := generate(p);
    n := generate(p);
3   Let the elite and new chromosome compete
   Winner, loser := evaluate(e, n);
```

Figure 3.7: Modification of compact GA to realize persistent elitism

In the second step, the loser chromosome is always replaced by a newly generated individual from the current probability vector. However, the winner can be passed on to the next generation as the elite, only when θ, the present generation length, less than the allowable pass-on length of the elite which is the parameter, η. That is if $\eta \geq \theta$ then the elite is passed on to the next generation.

```
Parameter:  $\eta$ : the allowable length of
inheritance

2.  Generate one individual from the probability
vector
    *If* the first generation *then*
        e := generate(p);
        $\theta$ := 0;
    n := generate(p);
3   Let the elite and new chromosome compete
  *if* $\theta \leq \eta$ *then*
    Winner, loser := evaluate(e, n);
    $\theta := \theta + 1$;
  *else*
    e := generate(prob. := 0.5);
    $\theta$ := 0;
```

Figure 3.8.: Modification of compact GA to realize non-persistent elitism

In addition to elitism in step two, the confident counter, cc is introduced as an important parameter to indicate how well the current search process of a CCGA cell is carried on. The higher the confident counter, the more confident toward finding the better solution of this current CCGA cell. Each CCGA cell uses the confident counter, cc, as the indicator to other neighbor cells so that each CCGA cell can compare its own confident counter with other neighbor's confident counter. In other word the confident counter keeps track of the number of the newly generated chromosome that is better than the current elite. The confident counter is incremented when the elite chromosome is assigned a new value.

In the third step, the elite and a newly generated individual from the probability vector compete similar to a normal compact GA. The probability vector of each cell is updated as shown at step 3 in Figure 3.9. If the confident counter of each cell reaches a certain level, then the probability vector and the confident counter of each cell are passed to its neighbors in step four.

In fifth and sixth step, once a cell receives the probability vector and confident counter from its neighbor, the cell performs local search by selecting the best probability vector from the incoming vectors. Then, the new inner probability vector is calculated from the adaptive combination weighted by the β value, which derives from the best confident counter shown in step 6.2 in Figure 3.9.

Using β for vector recombination in step 6.3, the CCGA can avoid local minima of the greedy search by shifting search direction gradually toward the better one. This feature of the CCGA contributes to the better performance. Finally, the CCGA keeps running until the probability vector is converged.

The proposed CCGA algorithm is different from the normal compact GA and other concept of probability model migration as follows:

(1) CCGA employs the elitism-based compact GA which is suitable for hardware implementation than UMDA or BMDA in [50, 54]. In addition, elitism-based compact GA performs better than the normal compact GA.

(2) With uniform cells, the probability vectors are passed directly to neighbor cells.

(3) The confidence toward the better probability vector is calculated as confident counters and passed to neighbor cells. In Figure 3.9, the step 2, 4 and 5 are inserted into the normal Compact GA.

(4) Improved probability vector combination is implemented by local search and adaptive combination in step 6 in Figure 3.9. This combination scheme proposed to provide a solution to the greedy search characteristic of the cooperative compact genetic algorithm [53, 115]. The local search is implemented through selecting the best probability vectors among its neighbor and the confident counter that keeps frequency of the updating to the probability vector of each cellular compact GA cell.

The higher *confident counter* values contribute to higher chance to reach the better solution. The probability vector combination refers to the following equation:

$$\mathcal{P}_i^{t+1}(x) = \beta\,\mathcal{P}_i^t(x) + (1 - \beta)\mathcal{P}_r^t(x)$$

Where β is the adaptive weight calculated from the best *confident counter* among neighbors. The better confident counter will provide the lower β which increases the influence of the incoming model from the neighbors.

$\mathcal{P}_i^{t+1}(x)$ is a new inner probability vector of a CCGA cell

$\mathcal{P}_r^t(x)$ is the best incoming probability vector from neighbors

```
L is chromosome length, N is population size
cc is Confident Counter,  CA is Cellular Automata space
C is initial value of the probability vector

for each cell  in CA do in parallel
    Initialize each p[i]
        For  i  := 1   to L do
            p[i] := C;
    Initialize cc
            cc := 0;
end parallel for
for each cell  i  in  CA do in parallel
  while not done do
  1.  Generate two individuals from the vector
          if the first generation then
                E := generate(p);
                ɵ := 0
                N := generate(p);

  2.   Let them compete and update the cc counter
          winner, loser := compete (E, N);
          if winner != loser &
             winner != E       then
              cc  := cc + 1;
          if $ɵ \le \eta$ then
            E := winner;
            ɵ := ɵ + 1;
          else
            e := generate(p := C);
            ɵ := 0;

  3.   Update the probability vector toward
       better one and Increment Confidence Counter
          for i := 1 to L do
          if winner[i] != loser[i] then
          if winner[i] == 1
          then p[i] += 1/N
          else p[i] -= 1/N

   4.  Check if cc reaches a target level then Send p and cc to
      the neighboring cell
              cc := 0;
   5.  Receives  p and cc from neighbors
   6.  Use local search and the adaptive  convex recombination  with the
       received p from the neighbor  and its own p
       6.1  Find the highest cc among neighbors' cc
            $cc_{max}$ := 0;
            for i := 1 to M do
               if (cc[i] > $cc_{max}$ )
                    $cc_{max}$ := cc[i];
            $p_{ccmax}$ := p[i]

       6.2  Convert $cc_{max}$ to $\beta : 0 \le \beta \le 1$
            $\beta$ := 1 / $cc_{max}$ ;

       6.3. Update $p_1$ with $\beta$
            for i := 1 to L do
              $p_1$[i] := $\beta p_1$[i] $+ (1 - \beta)\, p_{ccmax}$[i]

  7.   Check if the vector has converged
            for i := 1 to L do
               if p[i] > 0 and p[i] < 1 then
                  goto step 1
   8.  p represents the final solution
  end while
end parallel for
```

Figure 3.9: Pseudocode of cellular compact GA

(5) Asynchronous migration rate of probability vector for each CCGA cell is achieved by using *confident counter* since the updating rate of each CCGA cell to its *confident counter* is different. This contributes to the different rate to exchange the probability vector.

3.2.3 Benchmarks Problems and Experimental Results

In this section, the performance of CCGA with 4-node, 2-node, 1-node, and normal compact GA are compared. For 1-node case, it is equivalent to no communication between each CCGA node. All results were averaged over 10 runs. Each experiment is terminated when the PV converse to a solution.

(a) Problem Involving Lower and Higher Order BBs

A 100-bit one-max problem and a minimum deceptive problem (MDP) are used for evaluating the CCGA algorithm. A 100-bit one-max problem is specified in chapter II. We investigate the number of correct building blocks (BBs) or the solution quality. Figure 3.10 shows the performance of the CCGA on OneMax problem. After performing around 200 of generations, CCGA with 4-node can find the best solution. It takes almost 900 generations of CGGA with 1-node to get 100% correct BBs while the compact GA performs poorly.

For the 100-bit MDP problem, Figure 3.11 shows the performance of CCGA compared to the normal compact GA. The same trend as Figure 3.10 is found; however 1-Node CCGA suffer from lack of wider search strength in the solution space. As a result, 1-Node cannot find the best solution within 1000 generations. The compact GA gradually improves the search solution.

(b) Problem Involving Higher Order BBs

Fully deceptive problems are used to test the CCGA on the problems involving higher order BBs. The first deceptive problem is three-bit trap which is concatenated to form the 90-bit problem. The 4-node CCGA achieves better solution than 1-Node CCGA, and much better than the compact GA. From Figure 3.12, 4-node CCGA can find the best solution within 200 generations. For 1-Node CCGA, it is difficult to find the best solution. The communication between CCGA nodes contributes to the better results.

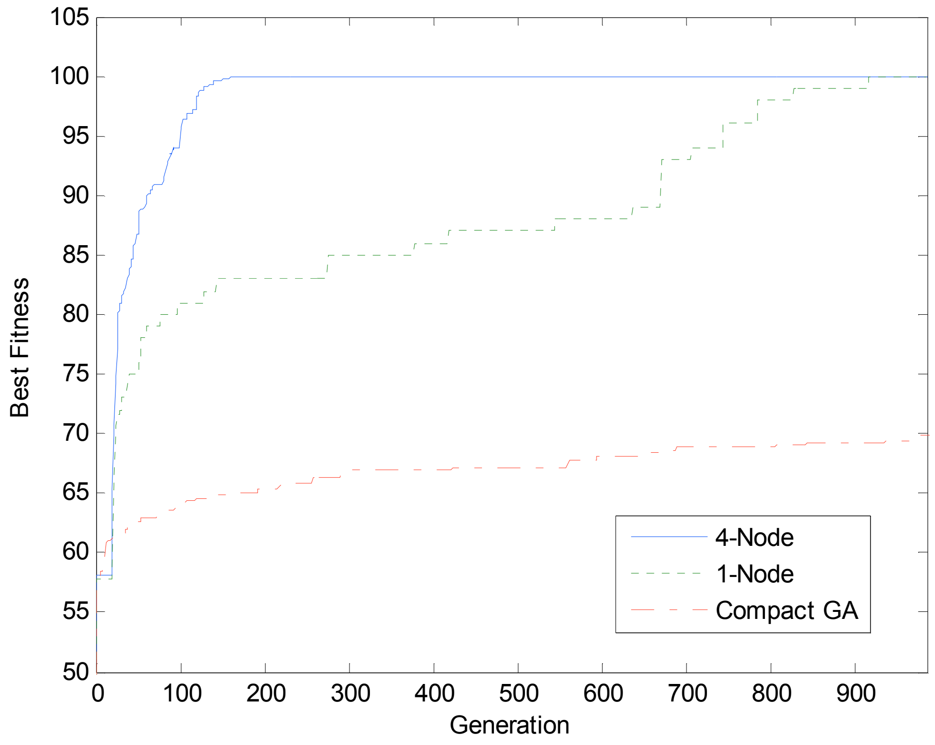


Figure 3.10: Performance of the CCGA on OneMax

The second deceptive problem is formed by concatenating the four-bit trap function to 120-bit. The results are compared in Figure 3.13. The 4-Node CCGA outperforms the 1-node and the compact GA. It can be noticed from Figure 3.12 and 3.13 that the four-bit trap problem poses higher challenge for the algorithms than the three-bit trap problem.

(c) Experimental Results for Continuous and Multimodal Functions

A circle and Schaffer's binary function described in section 2.2 are used to test the performance of CCGA. For the circle function, there are two variables which are represented or sampled to 15-bit precision. Figure 3.14 shows results of minimizing the circle function. The CCGA with 4-node significantly outperforms the compact GAs and the 1-Node of CCGA with regard to convergence speed and quality of solutions.

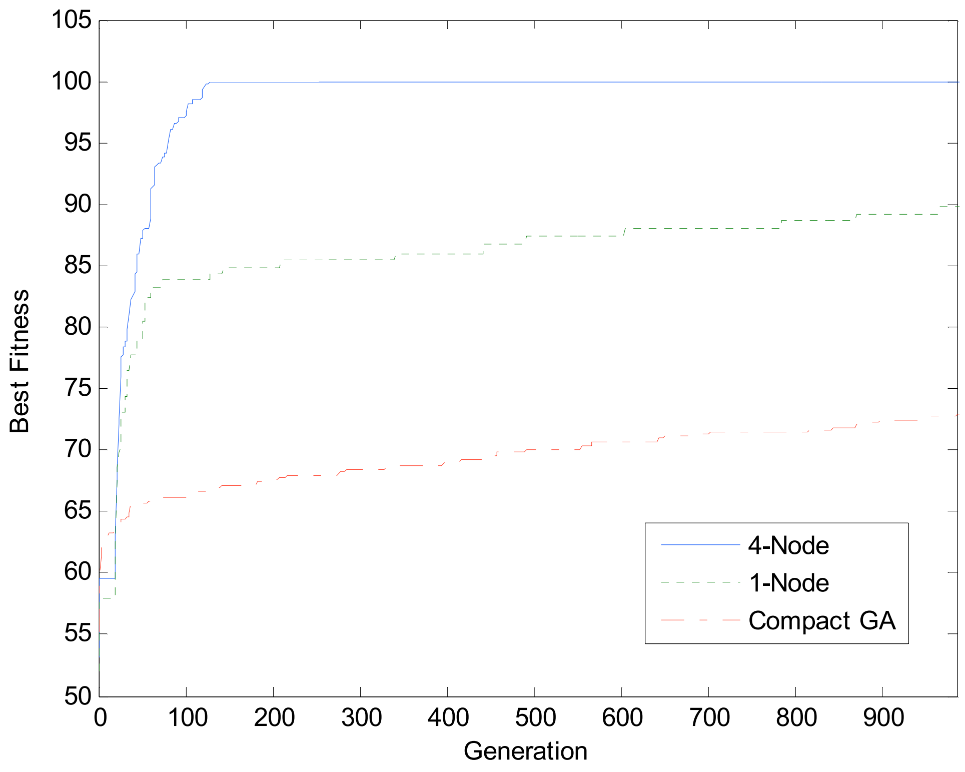


Figure 3.11: Performance of the CCGA on MDP

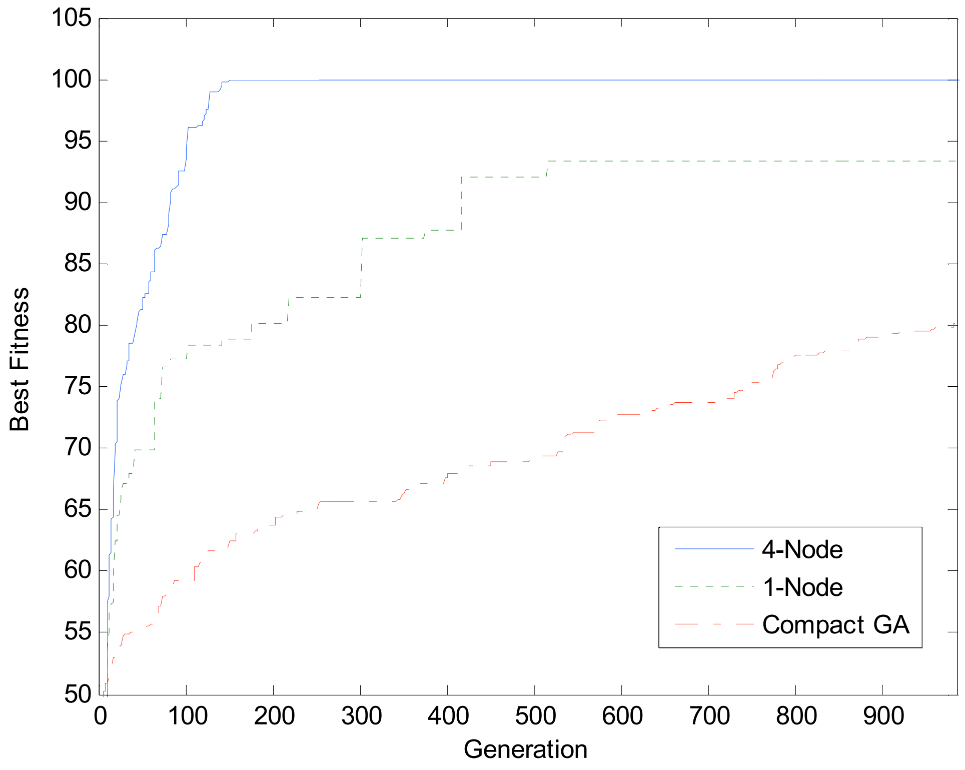


Figure 3.12: Performance of the CCGA on 90-bit of the three-trap problem

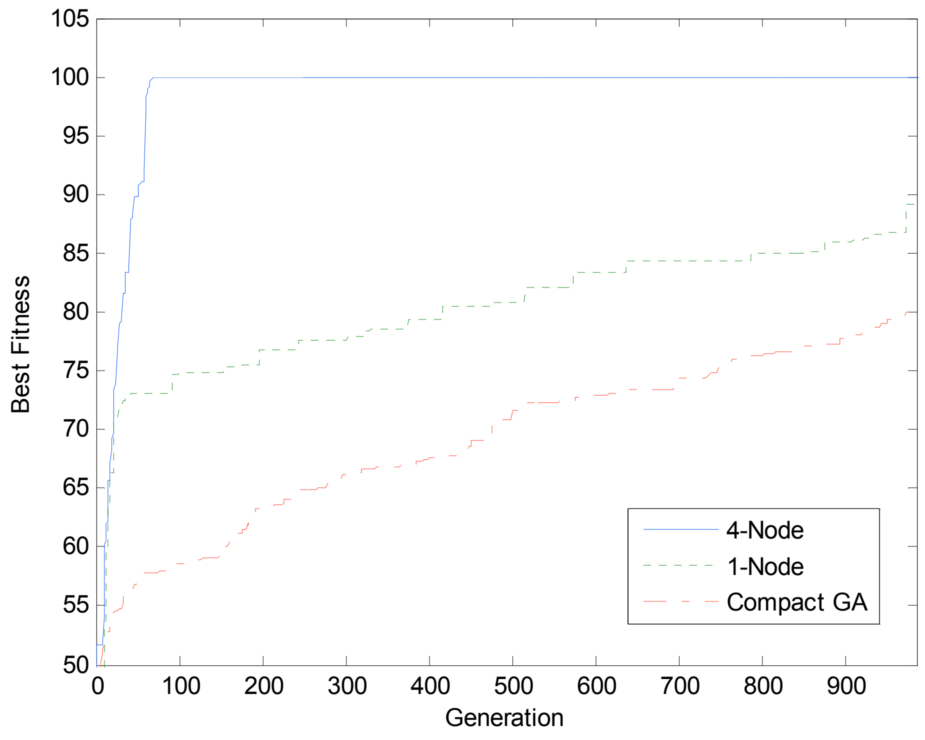


Figure 3.13: Performance of the CCGA on 120-bit of the four-trap problem

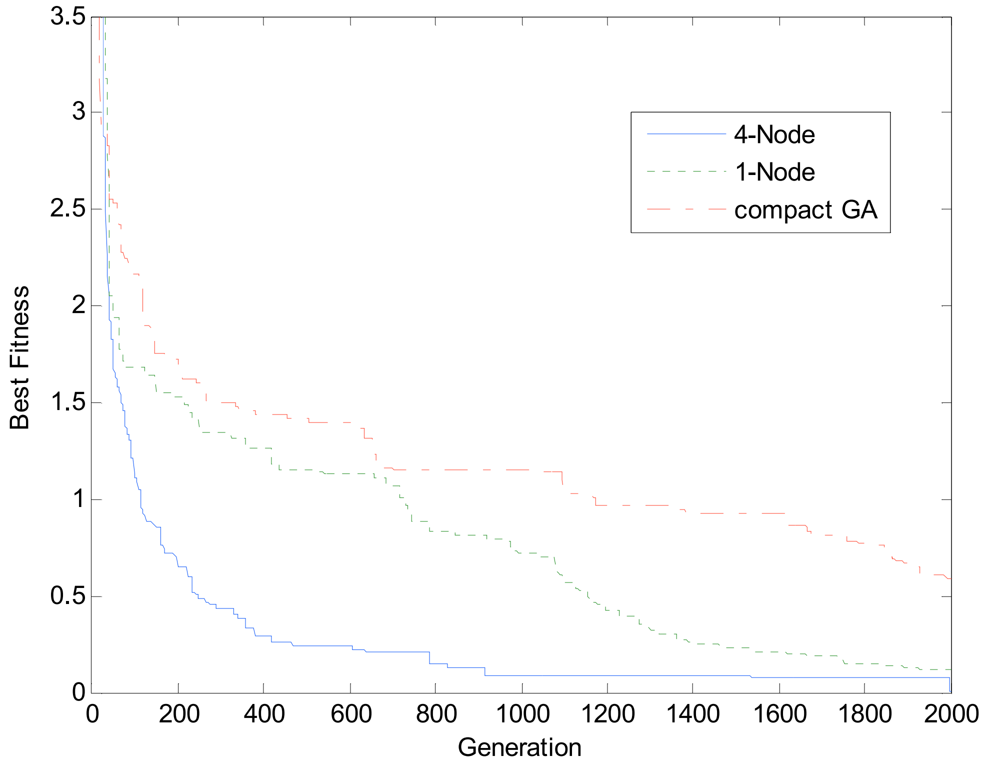


Figure 3.14: Performance of the CCGA on 30-bit of the circle function

The result of Schaffer's binary function is shown in Figure 3.15. The function has 5 variables. The function is degenerate in the sense that many points share the same global optimum function value (0.99400693). The global optimum points are located on the highest tip neat the origin. There many sub-optimal in the fitness landscape. In Figure 3.15, the 4-Node CCGA without communication between each node performs well better than 1-Node and the compact GAs.

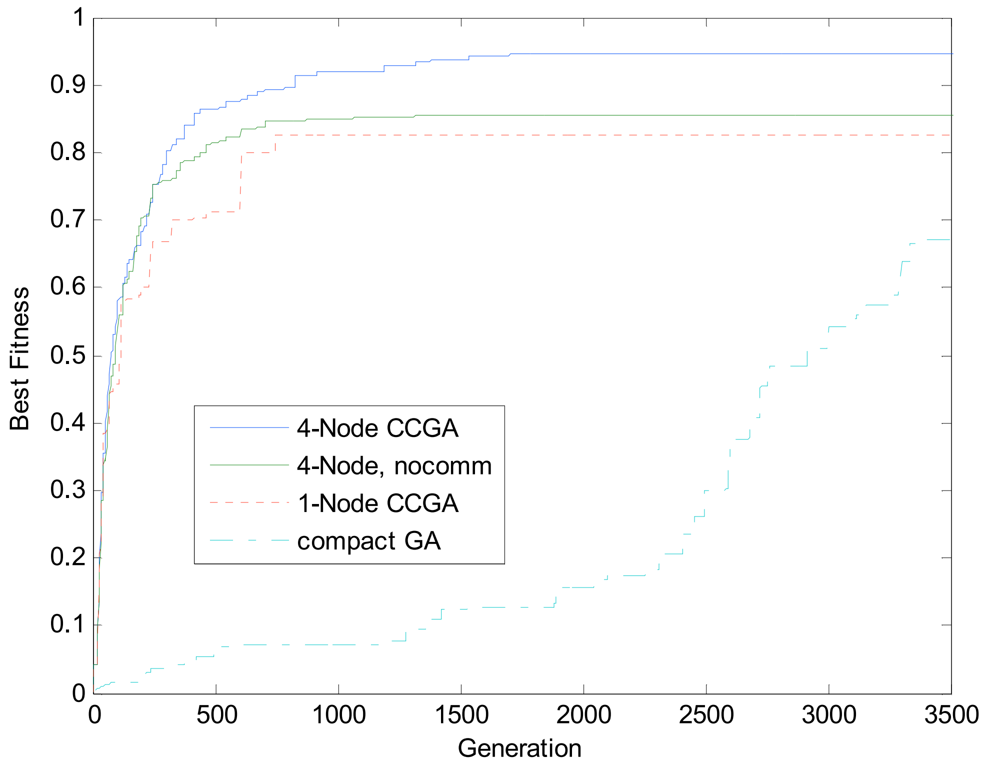


Figure 3.15: Performance of the CCGA on 75-bit of the Schaffer function

3.2.4 Hardware Architecture and Implementation of CCGA

In this thesis, we propose the hardware architecture of the CCGA that supports scalable precision and scalable number of variables. The architecture can be scaled to the problem size with little modification to the hardware. The proposed hardware architecture CCGA is based-on the architecture of a single CCGA cell. Each CCGA cell consists of four main blocks: Bit Module (BM), Package Switch Box (PSB), Fitness Evaluation (FE), and Main Controller (MC). Fig 3.16 shows the hardware architecture of CCGA.

The Bit Module (BM) is the basic building block of CCGA. In order to support the scalable number of variables or the problem size, the Random Access Memory (RAM) with N bit depth and W bit wide is used in the BM. In addition to RAM, two shift registers are used to store each bit of the currently generated chromosome and the current best bit position or the elite bit. The two shift registers have the size of 1-bit wide and N-bit depth. For example, for the problem of two variables and each probability vector is represented by 8-bit, the RAM size 8x2 (W=8, N=2) and the shift register size 1x2 will be used.

Each Bit Module (BM) consists of six sub-blocks: Confident Counter (CC) block, Probability Vector (PV) block, Update Block, Gen A block, RAMs and two shift registers. There are two key registers in Bit Module. These two registers are probability vector register within PV block and confident counter register within CC block.

In CC block, the *confident counter* is designed as an 8-bit counter. During fitness evaluation, the counter is incremented every time when the fitness of the winner is better than the current best fitness. The value of the counter is passed to the neighbor CCGA cells with the current probability vector.

The Update block is the hardware block that implements the step 6 of the pseudocode the Fig. 3.9. A hardware part of the block consists of comparators and multiplexers for comparing incoming *confident counter.* The best confident counter will be selected among the incoming confident counters of the neighbors. The confident counter (cc) is converted to β by using fractional number (1/cc). The multiplication of β with the probability vector is implemented using shift register instead of using multipliers which occupy more hardware resource. With shift register implementation, the β value which is equal to 1/*cc,* will be scaled down to multiple of 2. From the equation of vector combination of CCGA algorithm, after multiplication, the value of both probability vectors will be added using 8-bit adder.

The Package Switch Box consists of five First-In-First-Outs (FIFOs) that store sending and receiving the probability vector as an N-bit package between each CCGA cell. For a chromosome of M-bit length, the CCGA needs to have M number of probability vector which each probability vector sizes N-bit. Thus, for M-bit length chromosome, M packages of N-bit will be sent and received between each CCGA cell by through the FIFOs.

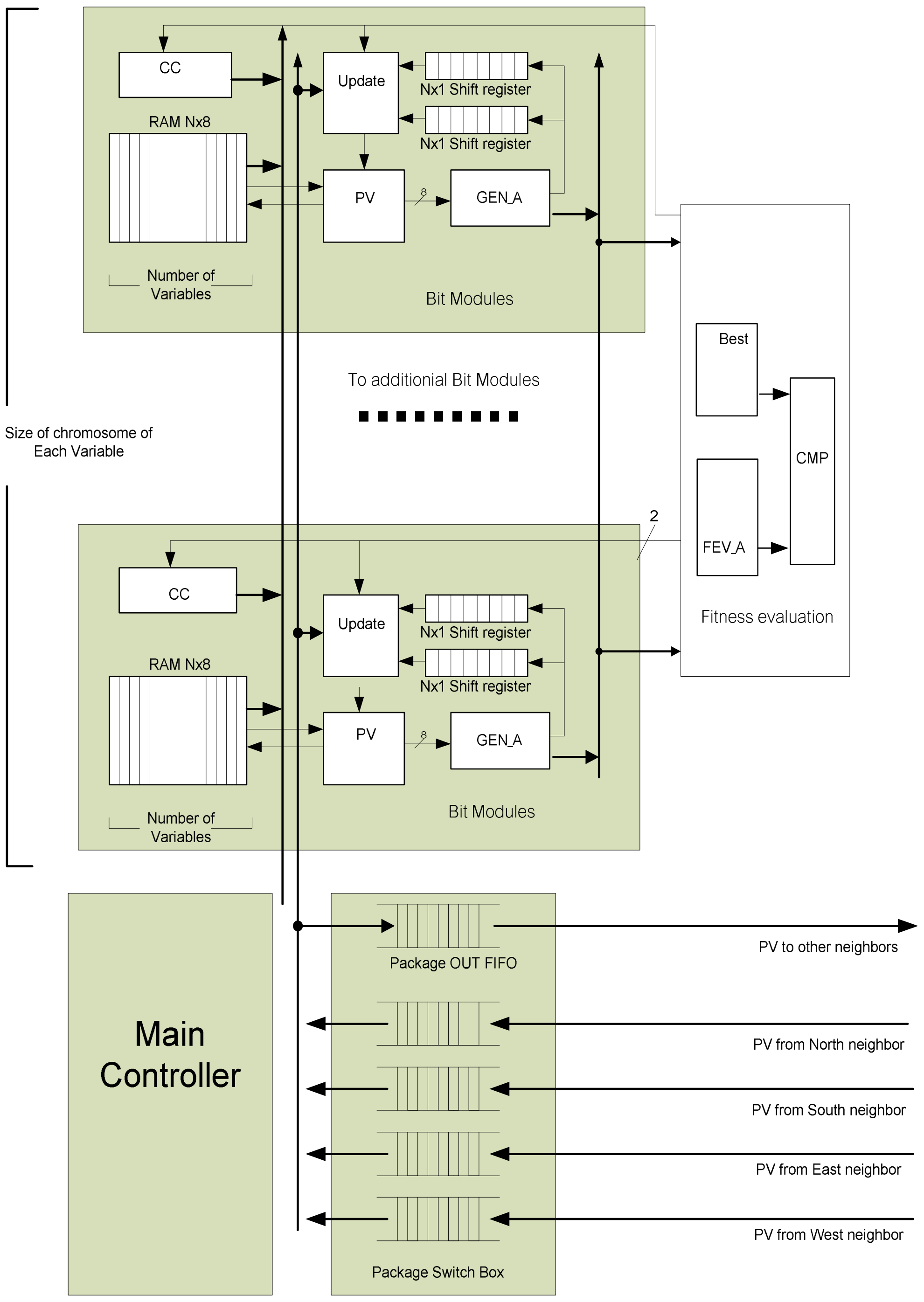


Figure 3.16: The proposed hardware architecture for CCGA

The Main Controller is a finite state machine that controls the four datapath blocks. The CCGA-bit module takes four clock cycles for generating each chromosome for tournament selection and updating probability vectors. The Package Switch Box module takes sixteen clock cycles for sending and receiving probability vectors; however, the number of clock cycles depends on the size of the chromosome for a specific problem. The Update block takes three clock cycles for latching probability vector to the internal registers and perform shifting and addition.

We implemented the CCGA with four nodes in Virtex-5 LX50 device. The code was designed and written in synthesizable Verilog HDL. ModelSim Version 6.2 and Cadence NC-Sim were used for simulation. Xilinx ISE 9.1 was used for FPGA implementation.

Table 3.3 shows the FPGA implementation of one node and when CCGA is scaled up to four nodes. From Table 3.3, the speed of the CCGA is not related to the number of the nodes which demonstrates that CCGA can be scaled up to a problem size in FPGA hardware. The comparison of FPGA resources is shown in Table 3.3. CCGA occupies the same amount of FPGA resources as others CGA. However, it's more practical to FPGA implementation since it has uniform cell type.

The comparison in term of speed and hardware resources to others compact GA implementation is shown in Table 3.4. CCGA delivers the same speed and requires the compatible hardware resources.

3.3 Discussion

The CCGA provides more efficient in hardware implementation than CoCGA since the CCGA uses only one type of cells. For scalability of CCGA, there is a limitation of increasing number of nodes to solve more difficult problems. For a particular problem, there are an optimum number of nodes that can solve the problems. This means that increasing number of nodes beyond a certain number will deteriorate the performance of CCGA. We can conclude that more difficult problems will require more number of CCGA nodes and that number should not exceed the optimum number of nodes. From the experimental results, using elitism in CCGA enhances the performance of CCGA over CoCGA drastically.

The proposed scalable hardware architecture of CCGA provides more flexibility and ease of hardware implementation at the cost higher FPGA resources. The CCGA node requires more resources especially the LUT that needs to be used as distributed random access memory (SRAM). However, the proposed architecture does not require the large size of SRAM. This advantage allows us to save the large macro SRAM inside FPGA for other purposes or lower cost FPGA can be used.

3.4 Summary

In this chapter, the CCGA is presented. The results provide initial evidence that CCGA can outperform the normal compact GA and can provide compatible results to the CoCGA with more applicable to FPGA implementation due to unified cell type. The CCGA delivers a more search performance with the adaptive probability vector recombination. For intrinsic evolvable or adaptive hardware, the CCGA can be used for a hardware GA for real-time evolution and adaptation with increased quality of search results. In addition, CCGA can address a scalability issue of genetic algorithm with problem size since CCGA can scale up with problem size by increasing network size as shown in Table 3.3.

TABLE 3.3: FPGA Hardware Resource Xilinx Virtex-5 LX50

Network size	FPGA resources for CCGA with 32-bit chromosome on Xilinx Vertex-5 LX50	
		CCGA
1	Slice Registers used Flip-Flops	621
	Slice LUTs used as Logic	1,932
	Total equivalent gate count	18,224
	Maximum Frequency	290Mhz
2x2	Slice Registers used Flip-Flops	1,642
	Slice LUTs used as Logic	5,506
	Total equivalent gate count	49,204
	Maximum Frequency	280Mhz

TABLE 3.4: Comparison of FPGA resources

	CGA [10]	CoCGA normal [10]	CoCGA leader [10]	CCGA
No. of flip-flop	541	598	168	1021
4-input LUT	1065	1296	359	2432
Total equivalent gate count	12602	17034	4651	30224
Max. frequency	330 Mhz	330 Mhz	300 Mhz	300 Mhz

CHAPTER IV
EVOLVABLE HARDWARE BASED-ON BBNN AND CCGA

Evolutionary artificial neural networks (EANN) use evolution to adapt the network structure and internal configuration as well as the parameters in dynamic environment [24]. Since the goal of evolvable hardware is to change hardware architecture and functions without human intervention, EANN can be regarded as a class of evolvable hardware. Block-based neural network (BBNN) model provide simultaneous optimization of network structure and connection weights. The BBNN consists of a two-dimensional array with support integer weights. The BBNN structure is suitable to be implemented in hardware especially using field programmable logic arrays (FPGAs). In addition, BBNN is successfully evolved using genetic algorithms to optimize weight and structure. However, in the past research, the genetic algorithm for BBNN was done in software off-line on computer system or on-chip embedded processor. This approach requires higher cost and larger FPGA since it needs on-chip processor.

This thesis presents an FPGA implementation of a cellular compact genetic algorithm and Block-based neural network. We propose a new integration in hardware between cellular compact genetic algorithm and Blocked-base neural network. The layer-based architecture for evolvable hardware consists of three layers. The top and bottom layers are for cellular genetic algorithms for evolving weight and structure. The middle layer is the Block-based neural network. With the cellular-like models of both Block-base neural network and cellular compact genetic algorithm, this layer based approach for hardware implementation provides modular block design in hardware and reduces interconnection length since the each cellular cells of both Block-based neural network and cellular compact genetic algorithm only interact with their neighbors even on the same layer or the another layer.

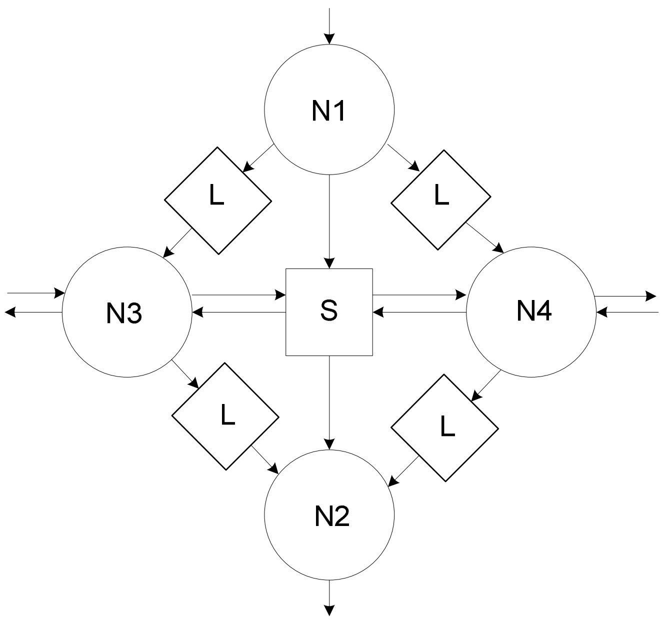
BBNN can implement both feedforward and feedback network configuration. A feedback configuration of BBNN architecture may cause a longer signal propagation delay. BBNN of size *m* x *n* can represent the input-output characteristics of any Multilayer perception (MLP) network [76].

In [120], SM. Merchant et al. propose the design of Smart Block-based Neuron (SBbN) that can be configured on-the-fly to emulate four types of the internal configuration modes of a BBNN neuron. However, the detail design of the SBbN is not presented. SBbN supports *gray mode* which the block is inactive and only pass the inputs to outputs.

4.1 The Proposed Hardware Architecture for BBNN

In this section, we propose the link-multiplexed block-based Neuron (LMBbN). This idea is similar to layer-multiplexed in [77]; however, in [77] the goal is to multiplex each layers of feed-forward network. Our design uses fixed-point fractional number format [78]. The LMBbN is derived from the analysis of a basic block of block-based neural network which is shown in Figure. 4.1. In Figure 4.1, there two types of routing switch: L and S types. These switches and four nodes form seven paths or "link" between two nodes. The architecture of the link-multiplexed BBNN is shown in Figure 4.2. With this LMBbN, we can reduce number of multipliers by sharing it with other links since there are a limited number of hard-macro multipliers inside DSP blocks of FPGAs. Hardware design of the general neuron is shown in Figure 4.3. The main controller receives configuration type. With configuration type, the general neuron can be configured to support case a, b, c, and d. of a BBNN block as shown in Figure 4.3. From Figure 4.4, if we use fixed-point fractional number size 10-bit for weights. The total number of bits to configure one BBNN block with the proposed hardware implementation is 102 bits.

- Seven of 10-bit for weights
- Three of 10-bit for biases
- 2-bit for config-type



S – S type switching box
L – L type switching box
N – neuron

Figure 4.1: a BBNN block

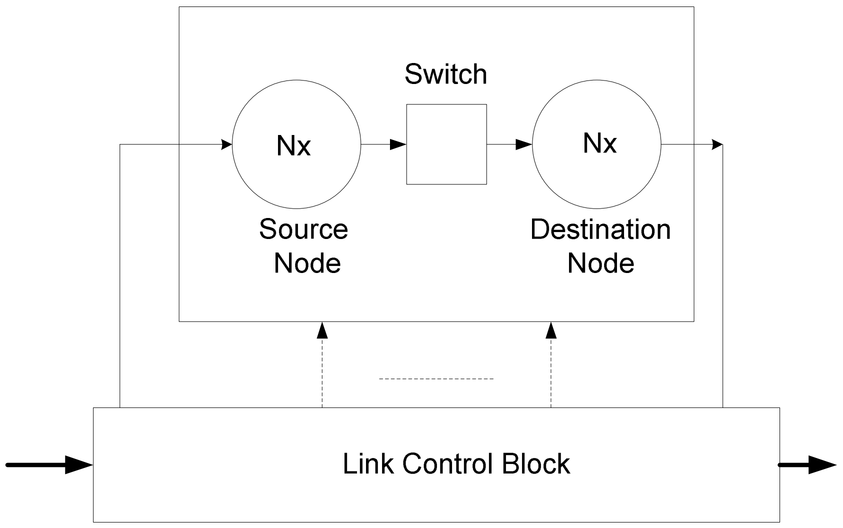


Figure 4.2: Link-multiplexed BBNN block

4.2 The Layer-based Architecture for integration of CCGA and BBNN

In this section, we propose the Layer-based architecture for evolvable hardware based on the block-based neural network and the cellular compact GA. To evolve the block-based neural network with one block, we need 132 bits genetic algorithms. However, to apply a block-based neural network to solve real-world problems, the more number of BBNN block will be required, for example if a BBNN network sized 3 x 4, it requires 1584 bits GA to evolve. The larger size of BBNN, the wider bits requires for GA. This turns to be the scalability problem for genetic algorithm. To solve this problem, we propose to use cellular genetic algorithm implemented in hardware since the CCGA can scale up with problem size since it has array-like characteristic as the BBNN model. Figure 4.4 shows the concept of the layer-based architecture.

From Figure 4.4, the BBNN occupies different layer from the cellular compact GA. These two layers interact to each other between nodes of BBNN and CCGA. Figure 4.5 shows the layer-based architecture with the eight nodes for each BBNN and CCGA layers. If l is the problem size that is the number of node of the BBNN, with n_p CCGA nodes, then each CCGA node contains l / n_p probability vectors.

4.3 FPGA Implementation

Table 4.1 shows the FPGA hardware resources required for implementing BBNN and CCGA. Each BBNN block was implemented using 10-bit fractional number. There is one 25x18 multiplier with a Finite State Machine (FSM) to control how to multiplex computation between the four sub-nodes of a basic BBNN block. There are ten data inputs to a BBNN blocks. These are seven weights (w13, w12, w14, w43, w42, w34, and w32) and three biases (b2, b3, and b4). We implemented each CCGA that has 100-bit which each bit represented by an 8-bit probability vector. Since one CCGA node supports 100-bit, the size of one CCGA node is about four times the size of a BBNN node as shown in Table 4.1.

From Table 4.1, the speed for each BBNN and CCGA node is about 300Mhz regardless of the size of matrix like 1x1, 2x2, or even 3x3. The reason is that each node of BBNN and CCGA is quite independent in term of hardware implementation especially CCGA since each 8-bit probability vector of one bit of CCGA was parallelized in hardware implementation. In our implementation, each BBNN only requires one DSP hard macro in Xilinx FPGA. This saving can allow implementing more nodes of BBNN in one FPGA.

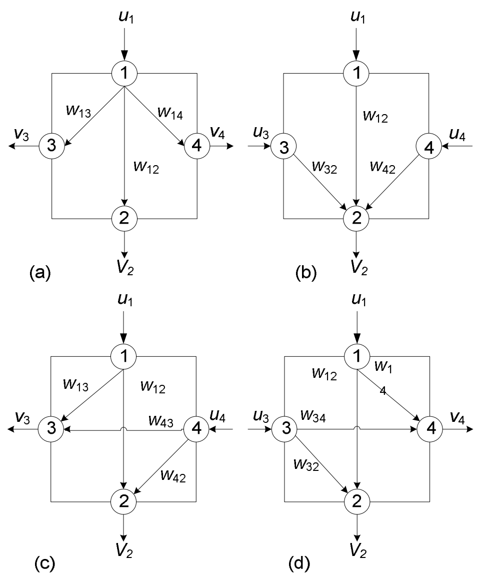


Figure 4.3: Four configurations of a BBNN block (a) 1/3, (b) 3/1, (c) 2/2, (d) 2/2

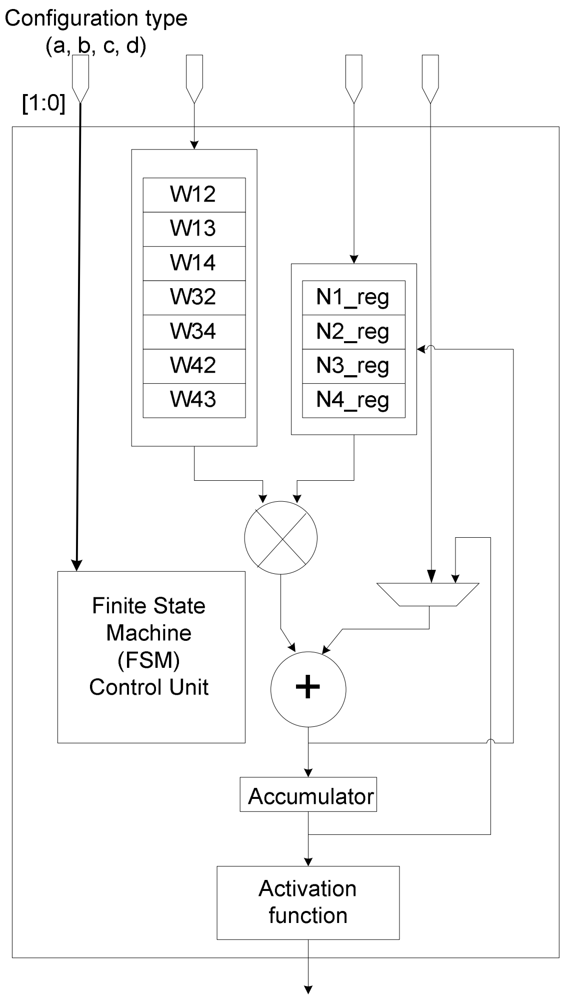


Figure 4.4: Hardware design of general neuron three inputs and three outputs

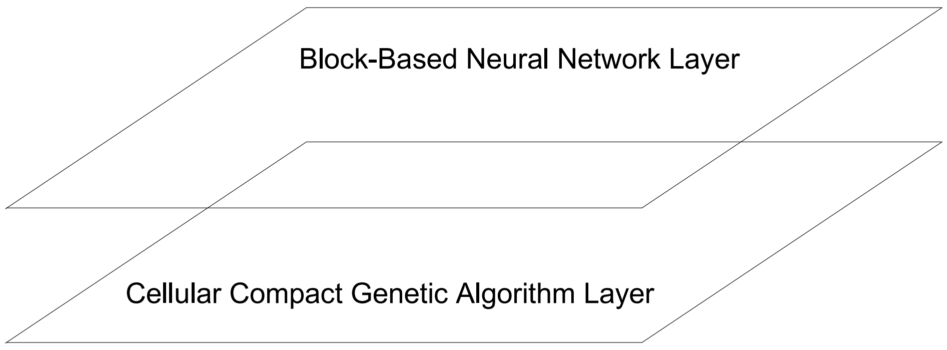

Block-Based Neural Network Layer
Cellular Compact Genetic Algorithm Layer

Figure 4.5: Layer-based architecture for evolvable hardware based on BBNN and CCGA

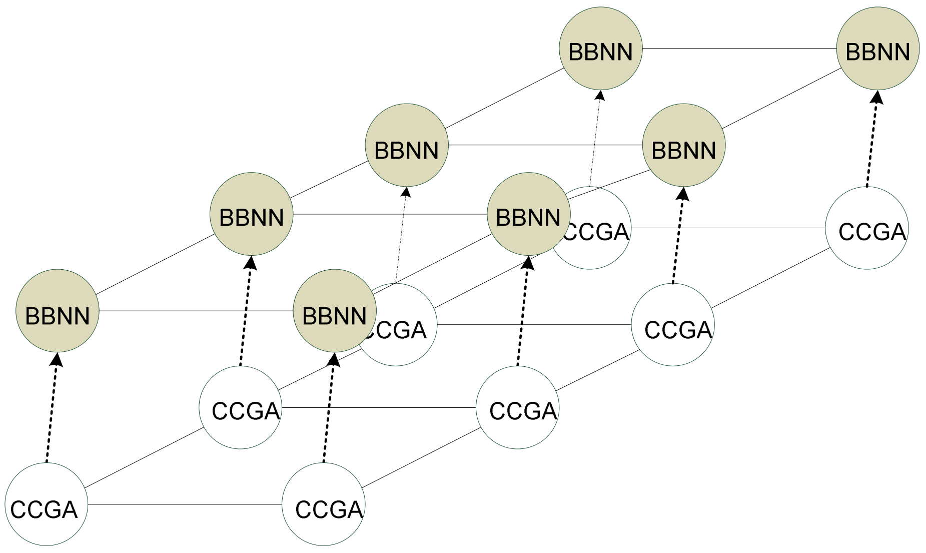


Figure 4.6: 2 x 4 BBNN layer and CCGA

4.4 XOR problem

To demonstrate the capability of integration between BBNN and CCGA, we implemented a 2 x1 BBNN network which has three BBNN nodes and evolving the weights with three nodes of CCGA to solve the XOR problem with two inputs, x1 and x2, and one output, y1. The “off” is when x or y has value < 0.0625 and > 0.9375 when “on”. Each CCGA has 102-bit which supports ten outputs; each has 10-bit, for seven weights and three biases for one BBNN node and two bits for four configuration types. In Figure 4.7 shows the block diagram of BBNN and CCGA to solve the XOR problem. The error calculation block computes using following equations:

TABLE 4.1

FPGA Hardware Resource Xilinx Virtex-5 LX50

Network size	FPGA resources for BBNN and CCGA on Xilinx Vertex-5 LX50		
		BBNN	CCGA
1x1	Slice Registers used Flip-Flops	341	621
	Slice LUTs used as Logic	263	1,932
	DSP48Es	1	0
	Total equivalent gate count	4,562	18,224
	Maximum Frequency	290Mhz	290Mh
2x2	Slice Registers used Flip-Flops	1326	1,642
	Slice LUTs used as Logic	974	5,506
	DSP48Es	3	0
	Total equivalent gate count	17,317	49,204
	Maximum Frequency	280Mhz	280Mhz
3x3	Slice Registers used Flip-Flops	3,262	5,130
	Slice LUTs used as Logic	2,300	16,549
	DSP48Es	9	0
	Total equivalent gate count	36,952	147,614
	Maximum Frequency	270Mhz	270Mhz

$$Fitness = \frac{1}{1+e} \tag{1}$$

$$e = \frac{1}{N\,n_o}\sum_{j=1}^{N}\sum_{k=1}^{n_o} e_{jk}^2 \tag{2}$$

$$e_{jk} = d_{jk} - y_{jk}(x) \tag{3}$$

Where, N and n_o are number of training data and output. d_{jk} and y_{jk} are desired and actual output respectively.

Figure 4.6 shows the hardware simulation results of training of the XOR problem. For one training pattern, the BBNN takes 60 clock cycles while CCGA takes only 3 clock cycles. At 549 epoch, the training was achieved fitness 0.998 with 18-bit precision fixed point arithmetic. The number of bit in neural network has an impact on the performance of the hardware [102].

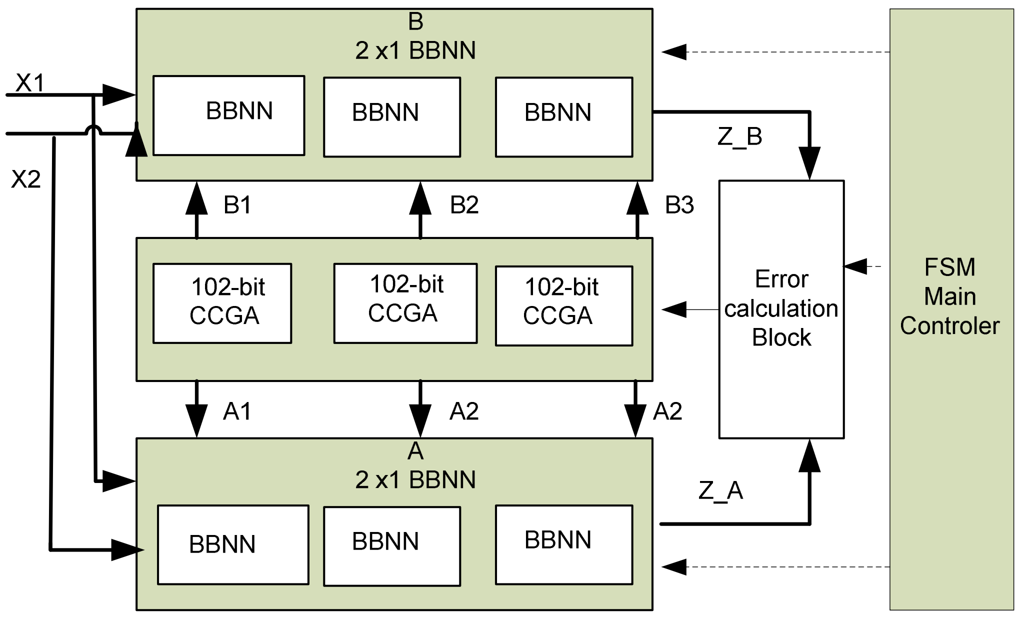


Figure 4.7: Block diagram of BBNN and CCGA for XOR

4.5 Discussion

The FPGA implementation results shown in Table 4.1 confirm the strength of the hardware implementation of the proposed BBNN and CCGA hardware architecture. In term of speed, increasing the number of nodes in BBNN and CCGA does not decrease the speed the FPGA implementation. The constant speed occurs due to the parallelized architecture of both BBNN and CCGA which employ point-to-point interconnect model. The FPGA resource requires by a BBNN node is less that required by CCGA nodes because the proposed multiplexed neurons. We can see from Table 4.1 that the BBNN node utilizes the hardwired macro DSP block in FPGA that support fast multiplication, subtraction, and addition. From Figure 4.9, the arithmetic precision impacts the quality of optimization in FPGA. The higher precision provides more room to perform optimization for CCGA.

4.6 Summary

In this section, an approach to training BBNN in hardware using the cellular compact genetic algorithm which is a kind of EDAs is presented. We propose the cellular compact GA and the layer-based architecture for integration between the block-based neural network and cellular genetic algorithm in hardware. With the layer-based architecture, evolvable hardware based-on the integration between BBNN and CCGA is feasible and effective since both have array- like architecture. This approach provides a solution for scalability of genetic algorithm since CCGA can scale up to the size of the BBNN by adding more CCGA nodes without sacrifice the speed in term of clock period and cycles. The XOR problem was used as an example of the approach. It has been implemented in hardware and can classify the data successfully. The more difficult classification problems can be solved in real-time with this kind of evolvable hardware. We believe that the more hardware resource in future FPGA will create more applications of the block-based neural network and the cellular compact GA for real world problems.

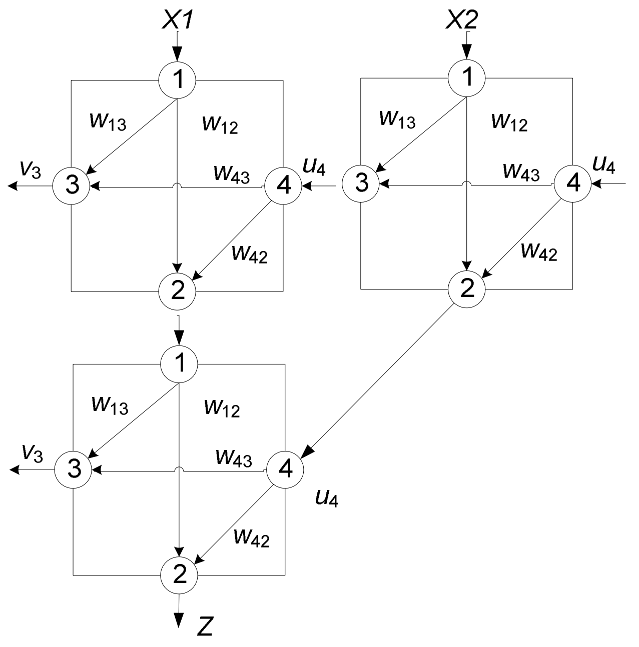


Figure 4.8: a structure of XOR that has fitness value of 0.998

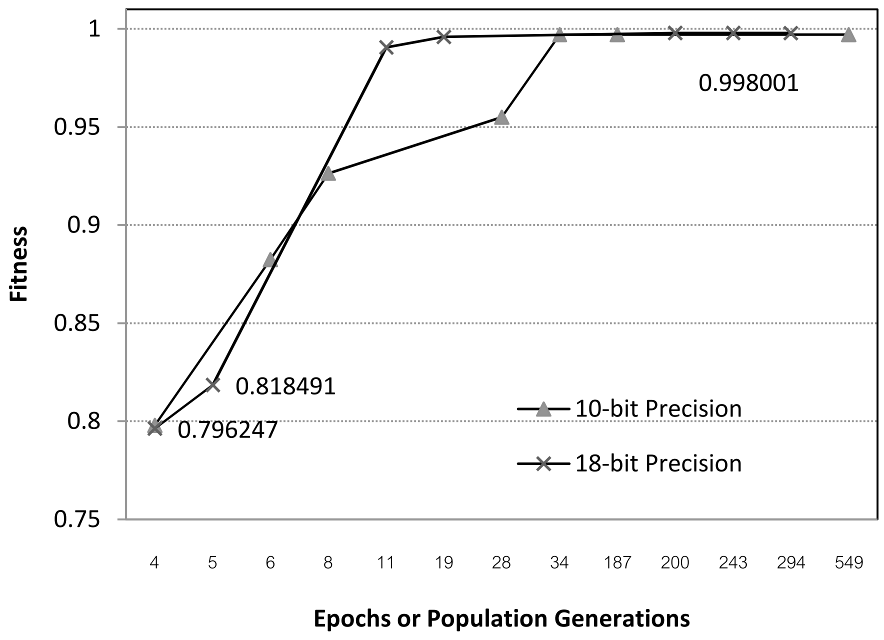


Figure 4.9: Fitness value in training process of XOR

CHAPTER V
EVOLVABLE HARDWARE FOR ECG SIGNAL CLASSIFICATION

In this chapter, the proposed evolvable hardware based-on CCGA and BBNN is applied to solve the problem of online ECG signal classification. The proposed system for on-line ECG beat recognition approach is shown in Figure. 5.1. It consists of two step processes: feature selection and classification. The feature selection consists of QRS detection and Hermite basis functions coefficient extraction, delivering these coefficients as the features to the input of classification process. The classification process uses block-based neural network (BBNN) learning by cellular genetic algorithm (CCGA). In this thesis, the feature selection process was carried on by our own functions created in the Matlab software (Mathworks Inc.,Nattick, MA) for which suitable solving Hermite polynomial and performing QRS window construction.

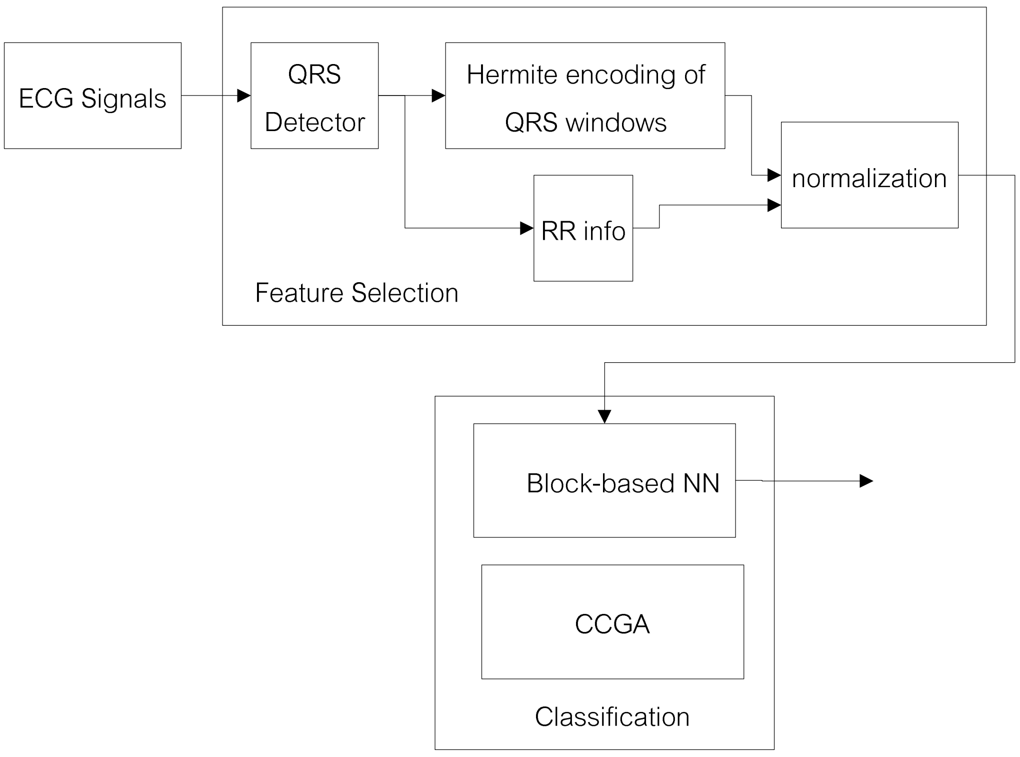


Figure 5.1: Our method of ECGs signal classification.

From chapter II, the AAMI recommends that each ECG beat be classified into the following five heartbeat types: beats originating in the sinus node, supraventricular ectopic beats, ventricular ectopic beats, fusion beats, and unclassifiable beats [114]. A large inter-individual variation of ECG waveforms is observed among individuals and within patient groups [101]. The waveforms of ECG beats from the same class may differ significantly and different types of heartbeats sometime possess similar shapes. Hence, the sensitivity and specificity of ECG classification algorithms are often unsatisfactorily low. For instance, a recent method [101] reports a sensitivity rate of 77.7% for VEB detection and 75.9% for SVEB detection.

5.1 ECG Data

The MIT-BIH arrhythmia database [114] is used in the experiment. The database contains 48 records obtained from 47 different individuals (two records came from the same patient). Each record contains two-channel ECG signals measured for 30 min. Twenty-three records (numbered from 100 to 124, inclusive with some numbers missing) serve as representative samples of routine clinical recordings. The remaining 25 (numbered from 200 to 234, inclusive with some numbers missing) records include unusual heartbeat waveforms such as complex ventricular, junctional, and supraventricular arrhythmias.

The data are bandpass filtered at 0.1–100 Hz and sampled at 360 Hz. There are over 109 000 labeled ventricular beats from 15 different heartbeat types. Table 5.1 lists the heartbeat types. Table 5.2 lists the annotation codes of the database. Table 5.3 summarizes the contents of the database. These tables show some of the contents of the database, reference should be made for more details [114]. The table 5.3 lists the beat types for each half-hour record in its entirety. The largest class is “Normal beat” (NORMAL) with over 75 000 examples and the smallest class is “Supraventricular premature beat” (SPC) with just two examples.

In agreement with the AAMI recommended practice, the four recordings containing paced beats were removed from the analysis. Paced beats refer to ECG signals generated by the heart under the help of an external or implanted artificial pacemaker for the patients whose electrical conduction path of heart is blocked or the native pacemaker is not functioning properly. The remaining recordings were divided into two datasets with each dataset containing ECG data from 22 recordings.

Table 5.1: Mapping the MIT-BIH arrhythmia database heartbeat types to the AMMI heartbeat classes

AAMI heartbeat class	**N**	**S**	**V**	**F**	**Q**
Description	Any heartbeat not in the S, V, F, or Q classes	Supraventricular Ectopic beat	Ventricular Ectopic beat	Fusion beat	Unknown beat
Heartbeat Types	Notmal beat (NORMAL) Left bundle branch block (LBBB) Right bundle branch block beat (RBBB) Atrial scape beat (AESC) Nodal (junctional) escape beat (NESC)	Atrial premature beat (APC) Aberrated atrial premature beat (ABERR) Nodal (junctional) premature beat (NPC) Supraventricular premature beat (SVPB)	Premature ventricular contraction (PVC) Ventricular escape beat (VESC)	Fusion of ventricular and normal beat (FUSION)	Paced beat (PACE) Fusion of paced and normal beat (PFUS) Unclassified beat (UNKNOWN)

Table 5.2: Heartbeat annotation codes in MIT-BIH arrhythmia database

Beat annotation codes	**Symbol**	**Meaning**
NORMAL	· *or* N	Normal beat
LBBB	L	Left bundle branch block beat
RBBB	R	Right bundle branch block beat
BBB	B	Bundle branch block beat (unspecified)
APC	A	Atrial premature beat
ABERR	a	Aberrated atrial premature beat
NPC	J	Nodal (junctional) premature beat
SVPB	S	Supraventricular premature beat
PVC	V	Premature ventricular contraction
RONT	r	R-on-T premature ventricular contraction
FUSION	F	Fusion of ventricular and normal beat
AESC	E	Atrial escape beat
NESC	j	Nodal (junctional) escape beat
SVESC	n	Supraventricular escape beat (atrial or nodal)
VESC	E	Ventricular escape beat
PACE	/	Paced beat
PFUS	f	Fusion of paced and normal beat
NAPC	x	Non-conducted P-wave (blocked APB)
UNKNOWN	Q	Unclassifiable beat
LEARN	?	Beat not classified during learning

Table 5.3: Summary of the database and beat types (entire records)

Record	**N**							**V**	**F**	**N**		**E**	**P**	**F**	**O**	**Q**
	.	**L**	**R**	**A**	**a**	**J**	**S**	**V**	**F**	**e**	**j**	**E**	**P**	**f**	**p**	**Q**
100	2239	-	-	33	-	-	-	1	-	-	-	-	-	-	-	-
101	1860	-	-	3	-	-	-	-	-	-	-	-	-	-	-	2
102	99	-	-	-	-	-	-	4	-	-	-	-	2028	56	-	-
103	2082	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
104	163	-	-	-	-	-	-	2	-	-	-	-	1380	666	-	18
105	2526	-	-	-	-	-	-	41	-	-	-	-	-	-	-	5
106	1507	-	-	-	-	-	-	520	-	-	-	-	-	-	-	-
107	-	-	-	-	-	-	-	59	-	-	-	-	2078	-	-	-
108	1739	-	-	4	-	-	-	17	2	-	1	-	-	-	11	-
109	-	2492	-	-	-	-	-	38	2	-	-	-	-	-	-	-
111	-	2123	-	-	-	-	-	1	-	-	-	-	-	-	-	-
112	2537	-	-	2	-	-	-	-	-	-	-	-	-	-	-	-
113	1789	-	-	-	6	-	-	-	-	-	-	-	-	-	-	-
114	1820	-	-	10	-	2	-	43	4	-	-	-	-	-	-	-
115	1953	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
116	2302	-	-	1	-	-	-	109	-	-	-	-	-	-	-	-
117	1534	-	-	1	-	-	-	-	-	-	-	-	-	-	-	-
118	-	-	2166	96	-	-	-	16	-	-	-	-	-	-	10	-
119	1543	-	-	-	-	-	-	444	-	-	-	-	-	-	-	-
121	1861	-	-	1	-	-	-	1	-	-	-	-	-	-	-	-
122	2476	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
123	1515	-	-	-	-	-	-	3	-	-	-	-	-	-	-	-
124	-	-	1531	2	-	29	-	47	5	-	5	-	-	-	-	-

200	1743	-	-	30	-	-	-	826	2	-	-	-	-	-	-	-
201	1625	-	-	30	97	1	-	198	2	-	10	-	-	-	37	-
202	2061	-	-	36	19	-	-	19	1	-	-	-	-	-	-	-
203	2529	-	-	-	2	-	-	444	1	-	-	-	-	-	-	4
205	2571	-	-	3	-	-	-	71	11	-	-	-	-	-	-	-
207	-	1457	86	107	-	-	-	105	-	-	-	105	-	-	-	-
208	1586	-	-	-	-	-	2	992	373	-	-	-	-	-	-	2
209	2621	-	-	383	-	-	-	1	-	-	-	-	-	-	-	-
210	2423	-	-	-	22	-	-	194	10	-	-	1	-	-	-	-
212	923	-	1825	-	-	-	-	-	-	-	-	-	-	-	-	-
213	2641	-	-	25	3	-	-	220	362	-	-	-	-	-	-	-
214	-	2003	-	-	-	-	-	256	1	-	-	-	-	-	-	2
215	3195	-	-	3	-	-	-	164	1	-	-	-	-	-	-	-
217	244	-	-	-	-	-	-	162	-	-	-	-	1542	260	-	-
219	2082	-	-	7	-	-	-	64	1	-	-	-	-	-	133	-
220	1954	-	-	94	-	-	-	-	-	-	-	-	-	-	-	-
221	2031	-	-	-	-	-	-	396	-	-	-	-	-	-	-	-
222	2062	-	-	208	-	1	-	-	-	-	212	-	-	-	-	-
223	2029	-	-	72	1	-	-	473	14	16	-	-	-	-	-	-
228	1688	-	-	3	-	-	-	362	-	-	-	-	-	-	-	-
230	2255	-	-	-	-	-	-	1	-	-	-	-	-	-	-	-
231	314	-	1254	1	-	-	-	2	-	-	-	-	-	-	2	-
232	-	-	397	1382	-	-	-	-	-	-	1	-	-	-	-	-
233	2230	-	-	7	-	-	-	831	11	-	-	-	-	-	-	-
234	2700	-	-	-	-	50	-	3	-	-	-	-	-	-	-	-

Table 5.4: Heartbeat types associated with the extracted beat for first five minutes and first part of the training set.

AMMI Heartbeat class	Heartbeat Type	Number of beats in first 5 Minutes	Number of beat in The first part of Training set
N	NORMAL , AESC, NESC	13206	6885
N	LBBB	854	506
N	RBBB	1217	778
S	APC	257	254
S	ABERR	7	4
S	NPC	15	15
S	SPC	0	0
V	PVC	1110	716
F	FUSION	180	85
Q	PFUS	5	5

Pace beats are not included in the classification experiments in this thesis as well as in the previous works [95, 103] chosen for comparison. The remaining records are divided into two sets: testing sets and training sets as shown in Figure 5.2. However, the training set is divided into two sub-groups. The training set consists of two data sets: the common set and patient specific set.

From Table 5.5, the common set is from the first set contains the 22 records, with each recordings period of total 30 min., randomly sampled. Since the number of normal beats is as high as ten times more than the other beat types, no NORMAL beat type is selected, but 5% of type-V (64), 30% of type-S (58) and all type-F (13), and type-Q (7) beats are sampled to have total of 142 beats in the common set. The goal to have the common set is to provide common representative of limited number of beat types in the training data.

From Figure 5.2, the second part of the training set is composed of 22 records listed in Table 5.5. For the second set or patient specific set, the heartbeats are from

22 recording for the first 5 min of the ECG recording of the patients, which conforms to the AAMI recommended practice that allows at most 5 min of recordings from a subject to be used for training purpose. The remaining beats of the record are test patterns.

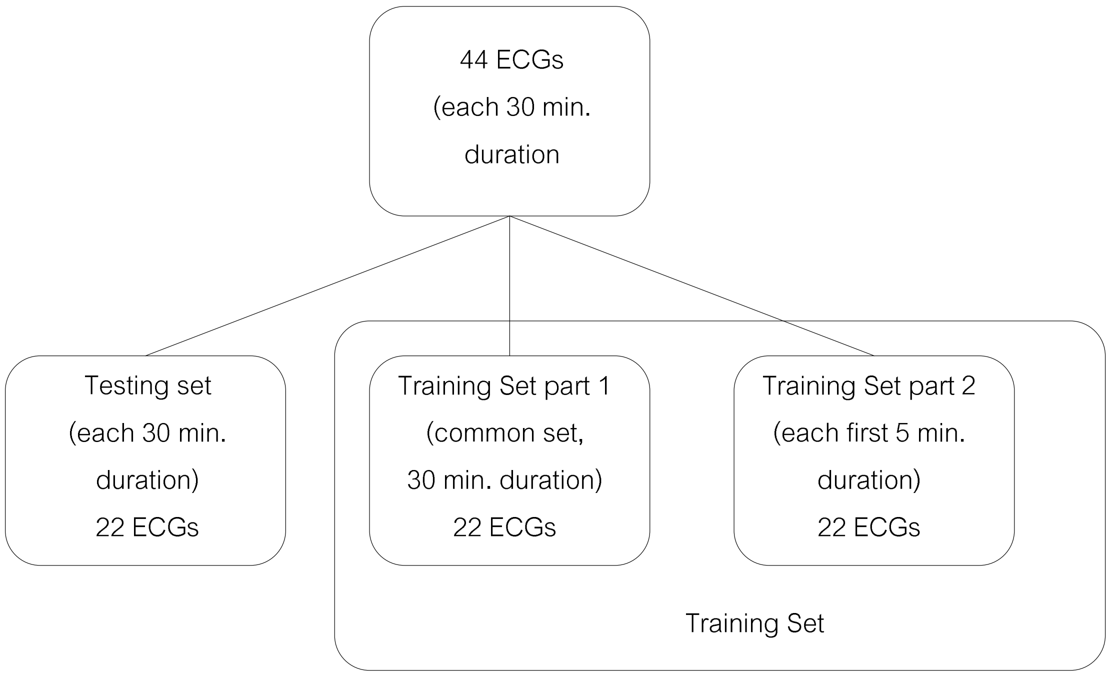


Figure 5.2 Division of the MIT-BIH arrhythmia database into training and testing sets

Table 5.5: Records in training and testing sets from the MIT-BIH arrhythmia database

Records in Testing Set	Records in Training Set
101, 103, 105, 111, 112, 113, 117, 121, 122, 123, ,200, 202, 210, 212, 213, 214, 219, 221, 222, 228, 231, 234	100, 106, 108, 109, 114, 115, 116, 118, 119, 124, 201, 203, 205, 207, 208, 209, 215, 220, 223, 230, 232, 233

5.2 Feature Selection

Basis function representations have been shown to be an efficient feature extraction method for ECG signals [104, 121]. Hermite basis functions provide an effective approach for characterizing ECG heartbeats and have been widely used in ECG signal classification [121-123]. Hermite basis function expansion has a unique width parameter that is an effective parameter to represent ECG beats with different QRS complex duration. The coefficients of Hermite expansions characterize the shape of QRS complexes and serve as input features. Let us denote *x(t)* be the discrete time QRS complex of ECG curve. The expansion of x(t) into Hermite series may be presented in the following way:

$$x(t) = \sum_{n=0}^{N-1} a_n \emptyset_n(t,\sigma) \tag{5.1}$$

where a_n (n = 0, 1, 2, …, N – 1) are the expansion coefficients while $\emptyset_n(t,\sigma)$ is the Hermite basis function defined as

$$\emptyset_n(t,\sigma) = \frac{1}{\sqrt{\sigma 2^n n! \sqrt{\pi}}} e^{-t^2/2\sigma^2} H_n(t/\sigma) \tag{5.2}$$

The functions $H_n(t/\sigma)$ are the Hermite polynomials. With $H_0(x) = 1$ and $H_1(x) = 2x$, the Hermite polynomials are defined recursively by

$$H_n(x) = 2xH_{n-1}(x) - 2(n-1)H_{n-2}(x) \tag{5.3}$$

For example, $H_2(x) = 4x^2 - 2, H_3(x) = 8x^3 - 12x$. Figure 5.2 shows Expanded QRS complex, $x(t)$ with the Hermite basis functions $\emptyset_n(t,\sigma)$ as a function of time for different orders.

The higher in the order of the function, the higher its frequency of changes within time domain and the better its capability to reconstruct the quick changes of the ECG paradigms.

To illustrate the way, in which Hermite polynomials approximate the ECG curve, the QRS segment of ECG signal is handled in a window with 91 data points around the R peak (45 points before and 45 after). At the data sample rate of 360 Hz, this gives a window of 250 ms, which is long enough to cover most of QRS signals. They have been also expanded by adding 45 zeros signals to each ends of the beats. After that all ECG signals are normalized by linear scaling to the range of and substracting the mean level of the first and the last data points. An example of normalized QRS complex of ECG signal is presented in Figure 5.3

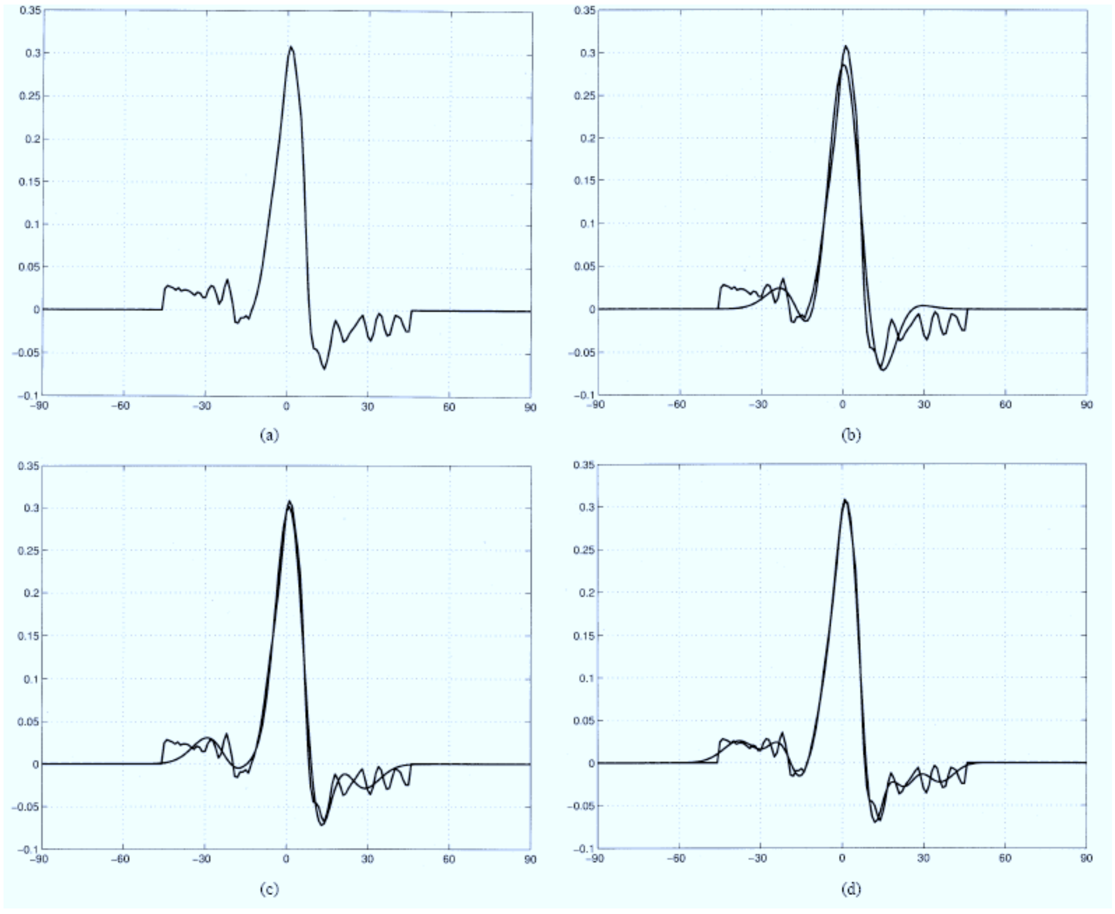


Figure 5.3: Expanded QRS complex of (a) ECG waveform and its estimation using (b) 6, (c) 9, and (d) 15 Hermite basis functions.

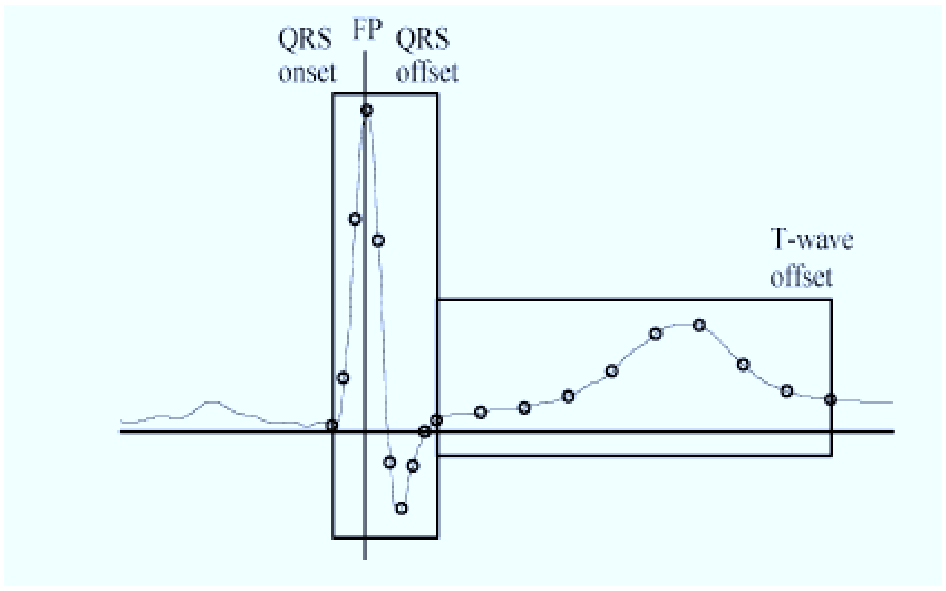


Figure 5.4: time-sampling methods for extracting ECG morphology features.

The modified QRS complex is decomposed onto a linear combination of Hermite basis functions. In the experiments the width σ has been set up in order to make almost half of the Hermite basis function signal values are close to 0 in the considered range. The expansion coefficients are obtained by minimizing the sum squared error, defined as follows:

$$E = \left\| x(t) - \sum_{n=0}^{N-1} a_n \emptyset_n(t,\sigma) \right\|_2^2 \qquad (5.4)$$

This error function represents the set of linear equations versus a_n in practice it can be solved by using SVD decomposition and pseudoinverse technique which can be done in a Microcontroller or it can be solved using the concept of adaptive filter and finding the coefficients by using the cellular genetic algorithm. In this thesis at this point, we find the coefficient of the Hermite function using the Matlab functions. However, finding coefficient of Hermite polynomial with the adaptive filter techniques with the cellular genetic algorithm can be an another add-on research topic and can provide the complete ECG signal classification using evolvable hardware approach on a single chip without processor core.

Different numbers of expansion coefficients can be used to approximate a QRS complex. The approximation error depends on the number of coefficients. There is a tradeoff between approximation error and computation time. More coefficients lead to smaller errors, but result in heavy computation burden. Furthermore, a small number of coefficients are able to approximate the heartbeats with small representation errors. Five Hermite functions allow a good representation of the QRS complexes and fast computation of the coefficients [121-123]. Besides the basis function coefficients and width parameter , the time interval between two neighboring R-peaks is included to discriminate normal and premature heart beats.

5.3 Classification Process

For the classification process, the two nodes of combined BBNN and CCGA are used to perform the experiment. Each node consists of 7x2 BBNN and two CCGAs. In BNNNs, the number of columns is equal to or greater than the number of input features. The number of rows needs to be determined so that the network has sufficient complexity for a given problem. A small network size is preferred, provided that it achieves the desired performance. Too big a network runs the risk of overfitting that causes poor generalization performance, and requires a more complex optimization process because of higher degree of freedom in the search space. Thus, a 2x7 network was selected as a minimum-size BBNN that can take seven input features. For CCGA, the first CCGA is for topology optimization since it requires only 2-bit per BBNN node which is 16x2 bits for this one combined 7x2 BBNN and CCGA. The second CCGA is for weight optimization which supports fixed-point 15-bit fraction number precision. The optimization of problems for the second CCGA is the size of 140 variables since each node of BBNN requires 10 parameters to be optimized. Each variable has the precision of 15-bit, so that it’s the problem of total chromosome length of 2100 bit for each of the CCGA that find the weights of BBNN. Fitness function evaluates the quality of the solutions. The fitness of an individual BBNN is defined as

$$Fitness = \frac{\beta}{1 + \frac{1}{n_0 M_1} \sum_{l=1}^{M_1} \left\| d_c^l - y_c^l \right\|} + \frac{1-\beta}{1 + \frac{1}{n_0 M_2} \sum_{l=1}^{M_2} \left\| d_c^l - y_c^l \right\|} \quad (5.5)$$

where M_1 and M_2 are the numbers of samples in the common and patient-specific training data, respectively.n_0 donote the number of output blocks. β controls relative importance of the common and patient-specific training in the final fitness, and a small value less than 0.5 (0.2 in this paper) gives more weight for correctly classifying patient-specific patterns. Both common and patient-specific training patterns are considered in the fitness function. While patient-specific data may serve as the training data for evolving BbNN specific to a patient, the inclusion of common training data is useful when the small segment of patient-specific samples contains few arrhythmia patterns [114].

5.4 Experimental Results

Figure 5.4 shows the results of the training experiment. The dotted and solid line corresponds to using 4-Node CCGA with communicating between CCGA nodes and without communicating between CCGA nodes. The evolution stops when the desired fitness is met after approximately 2200 generations. From Figure 5.4, the solid line indicates the occurrences of near-optimum structure and weights. The others represent four non-optimal. The 4-node CCGA with communicating between nodes perform significantly better than then non-communicating one.

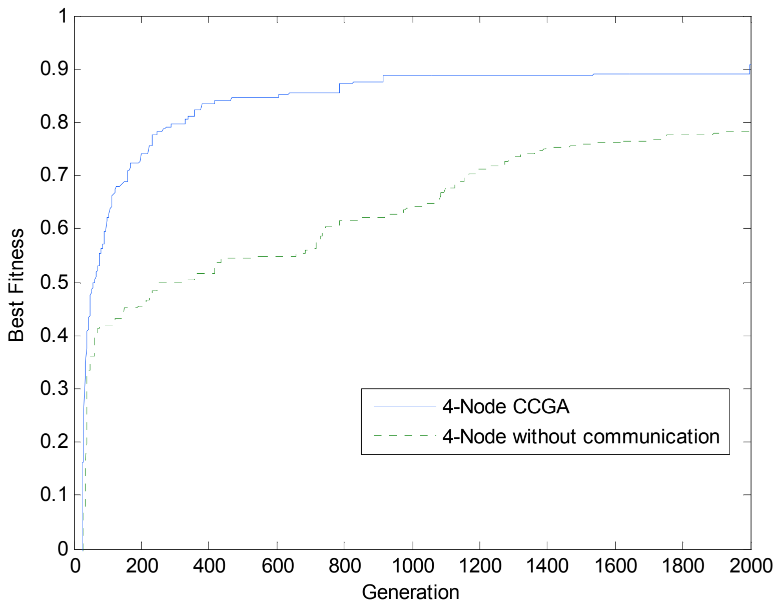


Figure 5.5: Fitness and number of generation.

Table 5.6: Summary of beat-by-beat classification results for the five classes.

Truth	Classification Result				
	N	S	V	F	Q
N	23049	340	256	10	0
S	603	641	48	1	0
V	232	86	1978	7	1
F	71	37	105	107	0
Q	4	0	2	1	0

Table 5.6 summarizes classification results of ECG heartbeat patterns for test records. Two sets of performance are reported: the detection of VEBs and the detection of SVEBs, in accordance with the AAMI recommendations. Four performance measures, classification accuracy (Acc), sensitivity (Sen), specificity (Spe), and positive predictivity (PP), are defined in the following using true positive (TP), true negative (TN), false positive (FP) and false negative (FN). Figure 5.5 summarizes how to calculate classification accuracy. Classification accuracy is defined as the ratio of the number of correctly classified patterns (TP and TN) to the total number of patterns classified. Sensitivity is the correctly detected events (VEB or SVEB) among the total number of events and is equal to TP divided by the sum of TP and FN. Specificity refers to the rate of correctly classified nonevents (non-VEBs or non-SVEBs) and is, therefore, the ratio of TN to the sum of TN and FP. Positive predictivity refers to the rate of correctly classified events in all detected events and is, therefore, the ratio of TP to the sum of TP and FP. The classification of ventricular fusion or unknown beats as VEBs does not contribute to the calculation of classification performance according to AAMI recommended practice. Similarly, performance calculation for detecting SVEBs does not consider the classification of unknown beats as SVEBs.

For VEB detection, the sensitivity was 85.8%, the specificity was 98.7%, the positive predictivity was 86.6%, and the overall accuracy was 97.7%. For SVEB detection, the sensitivity was 49.6%, the specificity was 98.2%, the positive predictivity was 58.06%, and the overall accuracy was 96.05%. From the results, the performance of SVEB detection is not as good as VEB detection, and the possible reasons include the more diverse types in class and the lack of class training patterns in patients [2], [5].

Table 5.7 summarizes sensitivity, specificity, positive predictivity, and overall accuracy of the four methods. The proposed method overall outperforms the methods in [112, 118]. The method in [128] performs slightly better than the proposed method.

Table 5.7: Performance comparison of VEB and SVEB Detection (in percent)

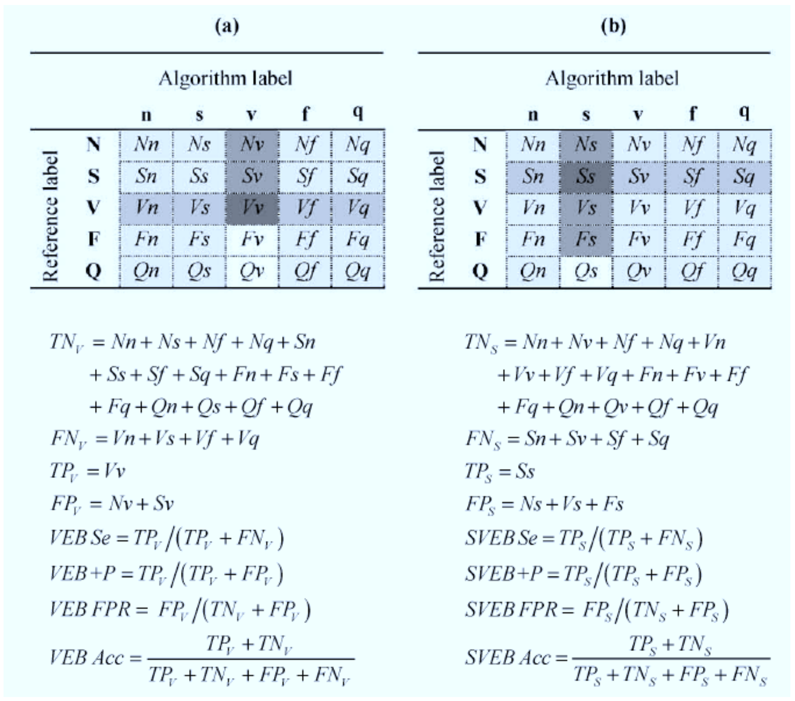
Method	VEB				SVEB			
	Acc	*Sen*	*Spe*	*+P*	*Acc*	*Sen*	*Spe*	*+P*
Hu *et. al.* [103]	94.8	78.9	96.8	75.8	N/A	N/A	N/A	N/A
Chazal *et. al.*[95]	96.4	77.5	98.9	90.6	92.4	76.4	93.2	38.7
Jiang *et. al.* [121]	98.8	94.3	99.4	95.8	97.5	74.9	98.8	78.8
Proposed	97.7	85.8	98.7	86.6	96.0	49.6	98.2	58.1

5.5 Discussion

The proposed technique is similar to the technique in [121]. However, in [121] the evolution algorithm with back-propagation learning is used for BBNN learning. Our purposed technique only uses the proposed CCGA for BBNN optimization. By using only CCGA, our proposed method is suitable for hardware implementation and by estimation, can provide speed up of over 100 times to software version in [121]. Our proposed method with four CCGA nodes has some disadvantage in comparison with work in [121]. This is due to the precision of hardware implementation and the limited number of CCGA nodes. With our proposed solution, the complete hardware solution of ECG signal classification can be implemented in hardware if the wavelet transform is used for feature selection.

5.6 Summary

This chapter presents the integration of CCGA and the feedforward implementation of BBNNs with its application to personalized ECG heartbeat classification. Network structure and connection weights are optimized using CCGA with four nodes that utilize evolutionary and gradient-based search operators. An adaptive rate adjustment scheme that reflects the effectiveness of an operator in generating fitter individuals produces higher fitness compared to predetermined fixed rates. The GDS operator enhances the performance as well as the optimization speed. The BBNN demonstrates a potential to classify ECG heartbeat patterns with a high accuracy for personalized ECG monitoring. The performance evaluation using the MIT-BIH arrhythmia database shows a sensitivity of 85.8% and overall accuracy of 97.7% for VEB detection. For SVEB detection, the sensitivity was 49.6% and the overall accuracy was 96.5%. These results shows improvement over other major techniques compared for ECG signal classification [112, 118]. The CCGA and BBNN approach can also be used where the dynamic nature of the problem needs an evolvable solution that can tackle changes in operating environments. The key contribution for our purposed technique is that the capability to be implemented in hardware for personal portable ECG classification because the CCGA and BBNN are designed to be implemented in hardware. The evolvable hardware based-on CCGA and BBNN can perform online ECG signal classification.



(a)

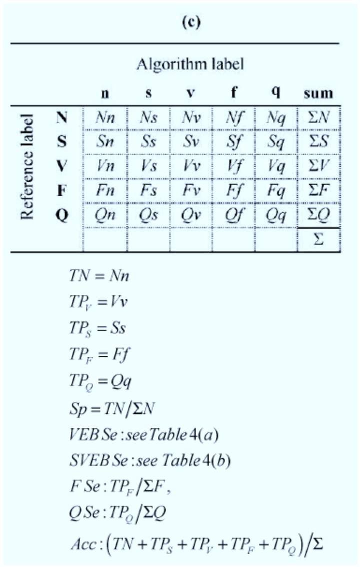
Reference label	Algorithm label: n	s	v	f	q
N	*Nn*	*Ns*	*Nv*	*Nf*	*Nq*
S	*Sn*	*Ss*	*Sv*	*Sf*	*Sq*
V	*Vn*	*Vs*	*Vv*	*Vf*	*Vq*
F	*Fn*	*Fs*	*Fv*	*Ff*	*Fq*
Q	*Qn*	*Qs*	*Qv*	*Qf*	*Qq*

$$\begin{aligned} TN_V &= Nn + Ns + Nf + Nq + Sn \\ &\quad + Ss + Sf + Sq + Fn + Fs + Ff \\ &\quad + Fq + Qn + Qs + Qf + Qq \\ FN_V &= Vn + Vs + Vf + Vq \\ TP_V &= Vv \\ FP_V &= Nv + Sv \\ VEB\ Se &= TP_V / (TP_V + FN_V) \\ VEB{+}P &= TP_V / (TP_V + FP_V) \\ VEB\ FPR &= FP_V / (TN_V + FP_V) \\ VEB\ Acc &= \frac{TP_V + TN_V}{TP_V + TN_V + FP_V + FN_V} \end{aligned}$$

(b)

Reference label	Algorithm label: n	s	v	f	q
N	*Nn*	*Ns*	*Nv*	*Nf*	*Nq*
S	*Sn*	*Ss*	*Sv*	*Sf*	*Sq*
V	*Vn*	*Vs*	*Vv*	*Vf*	*Vq*
F	*Fn*	*Fs*	*Fv*	*Ff*	*Fq*
Q	*Qn*	*Qs*	*Qv*	*Qf*	*Qq*

$$\begin{aligned} TN_S &= Nn + Nv + Nf + Nq + Vn \\ &\quad + Vv + Vf + Vq + Fn + Fv + Ff \\ &\quad + Fq + Qn + Qv + Qf + Qq \\ FN_S &= Sn + Sv + Sf + Sq \\ TP_S &= Ss \\ FP_S &= Ns + Vs + Fs \\ SVEB\ Se &= TP_S / (TP_S + FN_S) \\ SVEB{+}P &= TP_S / (TP_S + FP_S) \\ SVEB\ FPR &= FP_S / (TN_S + FP_S) \\ SVEB\ Acc &= \frac{TP_S + TN_S}{TP_S + TN_S + FP_S + FN_S} \end{aligned}$$



(c)

Reference label	Algorithm label: n	s	v	f	q	sum
N	*Nn*	*Ns*	*Nv*	*Nf*	*Nq*	ΣN
S	*Sn*	*Ss*	*Sv*	*Sf*	*Sq*	ΣS
V	*Vn*	*Vs*	*Vv*	*Vf*	*Vq*	ΣV
F	*Fn*	*Fs*	*Fv*	*Ff*	*Fq*	ΣF
Q	*Qn*	*Qs*	*Qv*	*Qf*	*Qq*	ΣQ
						Σ

$$\begin{aligned} &TN = Nn \\ &TP_V = Vv \\ &TP_S = Ss \\ &TP_F = Ff \\ &TP_Q = Qq \\ &Sp = TN / \Sigma N \\ &VEB\ Se : see\ Table\ 4(a) \\ &SVEB\ Se : see\ Table\ 4(b) \\ &F\ Se : TP_F / \Sigma F, \\ &Q\ Se : TP_Q / \Sigma Q \\ &Acc : (TN + TP_S + TP_V + TP_F + TP_Q) / \Sigma \end{aligned}$$

Figure 5.6: Fitness performance measures used for (a) distinguishing vebs from non-vebs, (b) distinguishing svebs from non-svebs, and (c) distinguishing the five AAMI heartbeat classes.

CHAPTER VI
CONCLUSIONS

The purpose of this chapter is to provide the summary and main conclusions of this dissertation. First, the contributions of the dissertation are summarized. Next, the main conclusions of the dissertation are provided.

6.1 What Has Been Done

A summary of the major results of this thesis follows:

Cellular Genetic Algorithm (CCGA) The thesis proposed the cellular genetic algorithm (CCGA), which is a parallel probabilistic model-building GA for evolvable hardware applications. CCGA replaces traditional migration of individual with the probabilistic migration using the concept of *confident counter*. Each CCGA improved the performance of the traditional compact GAs with elitism concepts. CCGA employs adaptive combination of probability vectors from its neighbors. CCGA can solve hard problems of bounded difficulty in subquadratic or quadratic time with respect to the number of candidate solutions that must be evaluated until the algorithm converges to the optimum. With parallel approach, CCGA can support scalability which is a limit of traditional GAs to solve the larger problems. CCGA is designed for hardware implementation. This allows hardware architecture of CCGA to be less complicated. The scalable hardware architecture for CCGA was proposed. For each node of CCGA, the scalable hardware architecture supports expandable number of variables to be optimized and flexible precision with expandable chromosome length. The proposed hardware architecture supports the concept of package switching on the chip with introducing *switch box,* designed with hardware FIFOs.

Evolvable hardware based-on Cellular Genetic Algorithm (CCGA) and Block-based neural network (BBNN) The thesis proposed an evolvable hardware based-on CCGA and BBNN. The layer-based architecture was proposed for integrating CCGA with BBNN in hardware. The new hardware design of BBNN neurons was proposed. The link-multiplexed concept is used for hardware design of BBNN neurons. BBNN is suitable for hardware implementation since it has array-like structure similar to

CCGA. With this new evolvable hardware, the evolvable hardware classifier can be realized in FPGA or even ASIC chip for real-world applications.

Real-world application of the proposed evolvable hardware to the online ECG signal classification. The proposed evolvable hardware based-on CCGA and BBNN was applied to the problem of online ECG signal classification. This demonstrates that CCGA can be scaled to solve the bigger problem. In addition, the proposed evolvable hardware can be applied to real-world problem. The proposed evolvable hardware can be implemented in FPGA or ASIC for a portable personalized ECG signal classifications for long term patient monitoring.

6.2 Main Conclusions

The evolvable hardware field has been the research subject for many years with gradual progress on the theory and application sides. The NASA and ESA initiative on adaptive and evolvable hardware for space application has stimulated this research community. The transition from traditional genetic algorithms to probabilistic model building GAs opens the way to design parallel genetic algorithms that communicating the probability model instead of individuals in the population. More advanced in FPGA technology allows designers to design larger digital circuits. This opens more opportunities for evolvable hardware. The proposed cellular genetic algorithm (CCGA) was created to take these opened opportunities. For computational intelligence, especially evolvable hardware field, the proposed evolvable hardware based-on CCGA and BBNN confirmed the effectiveness of hybrid between these two fields of evolutionary computation and neural networks. Furthermore, the solving real-world problem of ECG signal classification by the proposed evolvable hardware demonstrates that evolvable hardware can be an alternative to traditional solutions and capable to deal with real problems.

The CCGA and the proposed evolvable hardware contributed to the research in evolvable hardware community and hardware hybrid evolutionary computation and neural network in general. First of all, CCGA provides a solution to scalable problem of genetic algorithm. Second, the proposed evolvable hardware based-on CCGA and BBNN extends the probabilistic model building GAs to evolvable hardware. Finally, this evolvable hardware approach can be applied to the real-world problem of ECG signal classification.

6.3 Future works

There are opened problems in applying evolving connectionist model and bio-inspired approach to evolvable hardware. We conclude the key issues that can be subjects of future research as follows:

- From dualism concept presented in chapter II, there is trade-off and some insight on how to design the configurable fabric and the suitable learning algorithm to that fabric. We can generalize this idea by coming up with flexible fabric that can support more kinds of neural network and more bio-inspired technique to the fabric. Since we can see from the thesis that using only BBNN that implemented in hardware provides a fixed solution fabric. On the other side of dualism, the learning algorithm in our thesis we propose the CCGA and CoCGA for evolvable hardware. The more generalize hardware architecture can be proposed to allow more kind and more powerful genetic algorithms to be used with configurable fabric.

- More powerful hardware architecture of estimation of distribution algorithm (EDA) can be proposed since CCGA is the parallel version of univariate EDAs. Hardware BOA or ECGA can proposed as well.

- The bio-inspired approach can be further investigated to design configurable fabric architecture that supports self-replicating and cellular differentiation in the context of evolving connectionist model.

- The real-world application of evolvable hardware can be applied to the emerging bionic research since the evolvable hardware can support interaction between environment and can provide the optimum solution to the unknown best operating condition and configuration of human-and-machine interface.

References

[1] Higuchi, T., Iwata, M., Liu, Y., and Yao, X., (2006). Introduction to Evolvable Hardware. In Higuchi, T., Liu, Y., and Yao, X. (eds.), Evolvable Hardware, pp. 1–18. New York: Springer.

[2] Yao, X., and Higuchi, T. (1999). Promises and challenges of evolvable hardware. IEEE Transactions on Systems, Man, and Cybernetics 29: 87-97.

[3] Higuchi, T., et al. (1999). Real-world applications of analog and digital evolvable hardware. IEEE Transactions on Evolutionary Computation 3: 220–335.

[4] Mange, D. (1993). Wetware as a bridge between computer engineering and biology. In Proceeding of 2nd European Conference on Artificial Life, pp. 658–667.

[5] Mange, D., Stauffer, A., and Tempesti, G. (1998). Embryonic: a macroscopic view of the cellular architecture. In Proceeding of International conference on Evolvable Systems: From Biology to Hardware, pp. 174-184.

[6] Higuchi, T., et al. (1993). Evolvable hardware with genetic learning. In Proceeding of Simulated Adaptive behavior, pp. 417–424.

[7] Restrepo, H. F., and Mange, D. (2001). Embryonic implementation of a self-replicating universal turing machine. In Evolvable Systems: From Biology to Hardware, LNCS 2210, pp.74–87. Heidelberg: Springer.

[8] Tyrrell, A.M., and Barker., W. (2005). The POEtic hardware device: assistance for evolution, development and learning. In Higuchi, T., Liu, Y., and Yao, X. (eds.), Evolvable Hardware, pp.99–120. New York: Springer.

[9] Zhang, Y., Smith, S. L., and Tyrrell, A.M. (2004). Digital circuit design using intrinsic evolvable hardware. In Proceeding of NASA/DoD Conference on Evolvable Hardware, pp. 55–62.

[10] Liu, H., Miller, J.F., and Tyrrell, A. M. (2005). Intrinsic Evolvable Hardware Implementation of a robust biological development model for digital systems. In Proceeding of NASA/DoD Conference on Evolvable Hardware, pp. 87–92.

[11] Haddow, P., and Tufte, G. (2000). An evolvable hardware FPGA for adaptive hardware. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 553–560.

[12] Hollingworth, G., Smith, S., and Tyrrell, A.M. (2000). Safe intrinsic evolution of virtex devices. In Proceeding of NASA/DoD Conference on Evolvable Hardware, pp. 195-202.

[13] Bondalapati, K. and Prasanna, V. K. (2002). Reconfigurable computing systems. Proceeding of IEEE 90: 1201–1217.

[14] Hauck, S. (1998). The roles of FPGAs in reprogrammable systems. Proceeding of IEEE 86: 615–638.

[15] Sekanina, L. (2003). Virtual reconfigurable circuits for real-world applications of evolvable hardware. In Proceeding of International Conference on Evolvable system (ICES2003), pp. 332–343.

[16] Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning. New York: Addison Wesley.

[17] Mitchell, M. (1998). An Introduction to Genetic Algorithms. Cambridge, MA: MIT Press.

[18] Muhlenbein, H. and PaaB, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Proceeding of International Conference on Parallel Problem Solving from Nature (PPSN IV), pp. 100-111.

[19] Pelikan, M., Sastry, K., and Cantu-Paz, E. (2006). Scalable Optimization via Probabilistic Modeling. Heidelburg: Springer.

[20] Larranaga, P. and Lozano, J. A. (2001). Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation. Boston, MA: Kluwer Academic Publishers.

[21] Cantu-Paz, E. (2000). Efficient and accurate parallel genetic algorithms. Boston, MA: Kluwer Academic Publisher.

[22] Alba, E. and Tomassini, M. (2002). Parallelism and Evolutionary Algorithms. IEEE Transaction on Evolutionary Computation 6: 443–462.

[23] Kasabov, N. (2007). Evolving connectionist systems: the knowledge engineering approach. London: Springer-Verlag.

[24] Yao, X. (1999). Evolving artificial neural networks. Proceeding of IEEE 87: 1423–1447.

[25] Leung, H. F., Lam, H. K., Ling, S. H., and Tam, K. S. (2003). Tuning of the structure and parameters of a neural network using an improved genetic algorithm. IEEE Transaction on Neural Network 14: 79–88.

[26] Ludermir, T. B., and Zanchettin, C. (2006). An Optimization Methodology for Neural Network Weights and Architectures. IEEE Transaction on Neural Network 17: 1452–1459

[27] Yao, X. and Liu, Y. (1997). A new evolutionary system for evolving artificial neural networks. IEEE Transaction on Neural Networks 8: 694–713.

[28] Du, K. L. and Swamy, M.N.S (2006). Evolutionary Algorithms and evolving neural networks. Neural network in Softcomputing framework. London: Springer-Verlag.

[29] Tsai, J. T., Liu, T. K., and Chou, J. H. (2004). Hybrid Taguchi-genetic algorithm for global numerical optimization. IEEE Transaction on Evolutionary Computation 8: 365–377.

[30] Tsai, J., Chou, J., and Liu, TK. (2006). Tuning the Structure and Parameters of a Neural Network by Using Hybrid Taguchi-Genetic Algorithm. IEEE Transaction on Neural Network 17: 69–79.

[31] Moon, S. W and Kong, S. G. (2001). Block-based neural networks. IEEE Transaction on Neural Networks 12:307-317.

[32] Kajitani, I., Iwata, M., and Higuchi, T. (2006). A GA hardware engine and its application. In Higuchi, T., Liu, Y., and Yao, X. (eds.), Evolvable Hardware, pp. 41-64. New York: Springer.

[33] Sakanashi, H., Iwata, M., and Higuchi, T. (2006). EHW applied to image data compression. In Higuchi, T., Liu, Y., and Yao, X. (eds.), Evolvable Hardware, pp. 19-40. New York: Springer.

[34] Korenek, J. and Sekanina, L. (2005). Intrinsic evolution of sorting networks: a novel complete hardware implementation. In Proceeding of International Conference on Evolvable systems: from biology to Hardware (ICES2005), pp. 46–55.

[35] Martinek, T. and Sekanina, L. (2005). An evolvable image filter: experimental evaluation of a complete hardware implementation in FPGA. In Proceeding of the International Conference on Evolvable systems: from biology to Hardware (ICES2005), pp. 76–85.

[36] Smith, S. L., Crouch, D.P., and Tyrrel, A. M. (2003). Evolving image processing operations for an evolvable hardware environment. In Proceeding of International Conference on. Evolvable systems: from biology to Hardware (ICES2003), pp. 332–343.

[37] Kim, J. G., Noh, K. G. and Park, K. (2001). Human-like dynamic walking for a biped robot using genetic algorithm. In Proceeding of International Conference on Evolvable systems: from biology to Hardware (ICES2001), pp. 159–170.

[38] Stefatos, E. F. and Arslan, T., (2006) An incremental evolutionary strategy for the design of FIR filters targeting real-time applications. In Proceeding of IEEE congress on Evolutionary Computation, pp. 2787-2792

[39] Stefatos, E. F., Arslan, T., and Hamilton, A. (2008) Evolutionary techniques for precise and real-time implementation of low-power FIR filters. In Proceeding of IEEE congress on Evolutionary Computation, pp. 2706-2713.

[40] Pelikan, M., Goldberg, D.E., Lobo, F. (1999). A survey of optimization by building and using probabilistic models, IlliGAL Report No. 99018, Urbana, IL: University of Illinois at Urbana-Champaign,

[41] Larranaga, P. and Lozano, J. A. (2001). Estimation of Distribution Algorithms: A New Tool for Evolutionary Computation, Boston, MA: Kluwer Academic Publishers.

[42] Muhlenbein, H. and PaaB, G. (1996). From recombination of genes to the estimation of distributions I. Binary parameters. In Proceeding of International Conference on Parallel Problem Solving from Nature (PPSN IV), pp. 100-111.

[43] Harik, G., Lobo, F., and Goldberg, D. (1999). The compact genetic algorithm. IEEE Transactions on Evolutionary Computation 3: 287–309.

[44] Pelikan, M., Sastry, K., and Cantu-Paz, E. (2006). Scalable Optimization via Probabilistic Modeling. Heidelburg: Springer.

[45] Wook, C. and Ramakrishna, R.S. (2003). Elitism-based compact genetic algorithm. IEEE Transactions on Evolutionary Computation 7: 367–385.

[46] Cantu-Paz, E. (2000). Efficient and accurate parallel genetic algorithms. Boston, MA: Kluwer Academic Publisher.

[47] Alba, E. and Tomassini, M. (2002). Parallelism and Evolutionary Algorithms. IEEE Transaction on Evolutionary Computation 6: 443–462.

[48] Ocenasek, J. (2002). Parallel Estimation of Distribution Algorithms. Doctoral Dissertation. Faculty of Information Technology, Brno University of Technology.

[49] Mendiburu, A., Lozano, J. A., and Miguel-Alonso, J. (2005) Parallel implementation of EDAs based-on probability graphical model. IEEE Transaction on Evolutionary Computation, 9: 406-423.

[50] DelaOssa, L., Gmez, J.A., Puerta, J.M. (2004). Migration of probability models instead of individuals: an alternative when applying the island model to edas. In Proceeding of International conference on Parallel Problem Solving in Nature (PPSN 2004), pp. 242–252.

[51] Ahn, C. W., Goldberg, D. E., and Ramakrishna, R.S. (2003) Multiple-deme parallel estimation of distribution algorithm. In International conference on parallel processing and applied mathematics, pp. 544-551.

[52] Lobo, F. G., Lima, C. F., and Martires, H. (2004). An architecture for massive parallelization of the compact genetic algorithm. In Proceeding of Genetic and Evolutionary Computation Conference (GECCO 2004), pp. 412–413.

[53] DelaOssa, L. et al. (2006). Improving model combination through local search in parallel univariate EDAs. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 624–629.

[54] Schwarz, J. and Jaros, J. (2008). Parallel Bivariate Marginal Distribution Algorithm with Probability Model Migration. In Linkage in Evolutionary Computation, pp. 3–23, London: Springer-Verlag

[55] Sastry, K., Goldberg, D.E., and Liora, X. (2007). Towards billion-bit optimization via a parallel estimation of distribution algorithm. In

Proceeding of Genetic and Evolutionary Conference (GECCO 2007), pp. 412-413.

[56] Yoshida, N., and Yasuoka, T. (1999). Multi-gap: parallel and distributed genetic algorithm in VLSI. In Proceeding of International Conference on System, Man, and Cybernatics, pp. 571–576

[57] Bland, I. M., and Megson, G. M. (1998). The systolic array genetic algorithm: an example of systolic arrays as a reconfigurable design methodology. In Proceeding of IEEE Symposium on FPGA for Custom Computing Machine, pp. 260–261.

[58] Gallagher, J. C., Vigraham, S., and Kramer, G. (2004). A family of compact genetic algorithms for instrinsic Evolvable Hardware. IEEE Transactions on Evolutionary Computation, 8:111–126.

[59] Goldberg, D. E. (1989). Genetic Algorithms in Search Optimization and Machine Learning, New York: Addison Wesley.

[60] Yoshida, N., Moriki T., and Yasuoka, T. (1997). GAP: genetic VLSI processor for genetic algorithm. In Proceeding of 2nd International Symposium on Soft Computing, pp. 341–345.

[61] Shackleford, B. et al. (2000). An FPGA-based genetic algorithm machine. In Proceeding of ACM Symposium on Programmable Gate Array, pp. 218.

[62] Shackleford, B. et al. (1997). A high-performance hardware implementation of a survival-based genetic algorithm. In Proceeding of International Conference on Neural Information Processing (ICONIP97), pp. 686–689.

[63] Kajitai, T. et al. (1998). A gate level EHW chip: implementating GA operations and reconfigurable hardware on a single LSI. In Proceeding of International Conference on Evolvable System, pp. 1–12.

[64] Tufte, G. and Haddow, P. C. (1999). Prototyping a GA pipeline for complete hardware evolution. In Proceeding of First NASA/DoD Workshop on Evolvable Hardware, pp. 18–25.

[65] Wakabayashi, S. et al. (1998). GAA: a VLSI genetic algorithm accelerator with on-the-fly adaptation of crossover operators. In Proceeding of IEEE International Symposium on Circuits and Systems (ISCAS 98), pp. 268–271.

[66] Graham, P., and Nelson, B. (1996). Genetic algorithm in software and in hardware. In Proceeding of IEEE Symposium on FPGA for Custom Computing Machine, pp. 216–225.

[67] Turton, B H. and Arslan, T. (1994). A hardware architecture for a parallel genetic algorithm for image registration. In Proceeding of IEE/IEEE Conf. on Genetic Algorithms in Engineering Systems, pp. 88-93.

[68] Cho, M., Chung, S., and Chung, D. (2002). The implementation of GA processor with multiple operators, based on subpopulation architecture. In Proceeding of 9th International Conference on Neural Information Processing (ICONIP'02), pp. 1680-1683.

[69] Kim, J., Choi, Y., Lee, C. and Chung, D. (2002). Implementation of a high-performance genetic algorithm processor for hardware optimization. IEICE Transaction on Electronic. Computer E85C: 195–203.

[70] Nedjah, N., Alba, E., and Mourelle, L. M. (2006). Parallel Evolutionary Computation. London: Springer.

[71] Jelodar, M., Kamal, M, Fakhraie, S.M., and Ahmadabadi, M. (2006). SOPC-based parallel genetic algorithm. In Proceeding of IEEE Congress on Evolutionary. Computation, pp. 2800–2806.

[72] Aporntewan, C., and Chongstitvatana, P. (2001). A hardware implementation of the compact genetic algorithm. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 624–629.

[73] Kramer, G. R., and Gallagher, J. C. (2005). An analysis of the search performance of a mini-population evolutionary algorithm for a robot locomotion control problem. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 2768–2775.

[74] Vigraham, S. A., and Gallagher, J. C. (2004). On the relative efficacies of space saving *CGAs for Evolvable Hardware Applications. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 2187–2193.

[75] Zhu, Z., Mulvaney, D., and Chouliaras, V. (2006). Investigation of a new genetic algorithm designed for system-on-chip realization. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 2981–2987.

[76] Moon, S. W., and Kong, S. G. (2001). Block-based neural networks. IEEE Transaction on Neural Networks 12:307-317.

[77] Himavathi, S. et. al. (2007). Feedforward neural network implementation in FPGA using layer multiplexing for effective resource utilization. IEEE Transaction on Neural Networks 18:880-888.

[78] Savich, A. W. et al. (2007). The impact of arithmetic representation on implementing MLP-BP on FPGAs: a study. IEEE Transaction on Neural Networks 18: 240-252.

[79] Ienne, P., Cornu, T., and Kuhn, G. (1996) Special-purpose digital hardware for neural network: an architectural survey. Journal of VLSI Signal Processing Systems 13: 5-25.

[80] Dias, F. M., Antunes, A. and Mota, A. M. (2004). Artificial neural network: a review of commercial hardware. Journal of Engineering applications of artificial intelligence 17: 945-952.

[81] Zhu, J., and Sutton, P. (2003). FPGA implementation of neural networks- a survey of a decade of progress. In Proceeding of International symposium on Field Programmable Logic, pp. 1062-1066.

[82] Zang, D., and Pal, S. K (2002). Neural networks and systolic array design. New Jersey: World Scientific.

[83] Omondi, A. R., and Rajapakse, J.C. (2006). FPGA implementation of neural network. Heidelberg: Springer.

[84] Maeda, Y., and Tada, T. (2003). FPGA implementation of a pulse density neural network with learning ability using simultaneous perturbation. IEEE Transaction on Neural Networks 14: 688–695.

[85] Hikawa, H. (2003). A Digital Hardware Pulse-Mode Neuron With Piecewise Linear Activation Function. IEEE Transaction on Neural Networks 10: 545–553.

[86] Turner, R. H., and Woods, R. F. (2004). Highly efficient limited range multipliers for LUT-based FPGA architectures. IEEE Transaction on Very Large Scale Integration (VLSI) Systems 15: 1113–1117.

[87] Govindu, G., Zhuo, L., Choi S., and Prasanna, V. (2004). Analysis of high performance floating-point arithmetic on FPGAs. In Proceeding of IEEE

International Parallel and Distributed Processing Symposium, pp.149–156.

[88] Ewe, C. T. (2005). Dual fixed-point: an efficient alternative to floating-point computation for DSP applications. In Proceeding of Field Programmable Logic Application (FPL2005), pp. 715–716.

[89] Nedjah, N., Mourelle, L.M. (2007). Reconfigurable Hardware for Neural Networks: Binary radix vs. Stochastic. Journal of Neural Computing and Applications 16: 155–249.

[90] Nedjah, N. et al. (2008). Reconfigurable MAC-Based Architecture for Parallel Hardware Implementation on FPGAs of Artificial Neural Networks, In Proceeding of International Conference on Neural Network (ICANN 2008), pp. 169–178.

[91] Pasero, E., and Perri, M. (2004). Hw-Sw Co design of a flexible neural controller through a FPGA-based neural network programmed in VHDL. In Proceeding of IEEE International Joint Conference on Neural Networks, pp. 4639–4644.

[92] Kim, S. S., and Jung, S. (2004). Hardware implementation of real time neural network controller with a DSP and an FPGA. In Proceeding of IEEE International Conference on Robotics and Automation pp. 3161–3165.

[93] Kim, J., and Jung, S. (2008) Implementation of the RBF neural chip with the on-line learning back-propagation algorithm. In Proceeding of IEEE International Joint Conference on Neural Networks, pp. 378-384.

[94] Girones, R. G., Palero, R.C., and Boluda, J. C. (2005). FPGA implementation of a pipelined on-line backpropagation. Journal of VLSI Signal Processing 40: 189-213.

[95] Chazal, P. de., O'Dwyer, M., and Reilly, R. B. (2004). Automatic classification of heartbeats using ECG morphology and heartbeat interval features. IEEE Transaction on Biomedical Engineering, 51: 1196–1206

[96] Linh, T. H., Osowski, S., and Stodolski, M. (2003). On-line heart beat recognition using Hermite polynomials and neuro-fuzzy network. IEEE Transaction on Instrument and Measurement, 52: 1224–1231.

[97] Hamilton, P.S., and Tompkins, W. J. (1985) A real-time QRS detection algorithm. IEEE transaction Biomedical Engineering 32: 230-236.

[98] Hu, Y. H., Tompkins, W. J., Urrusti, J. L., and Afonso V. X. (1994). Applications of artificial neural networks for ECG signal detection and classification. Journal of Electro cardiology 11: 66–73

[99] Laguna, P., Jan, R., Caminal, P., Rix H., and Thakor, N. V. (1995) Adaptive estimation of the QRS complex wave in the electrocardiographic signal (ECG) by the Hermite model: Classification and ectopic beat detection. Medical & Biological Engineering & Computing 34: 58–68.

[100] Osowski, S., and Linh, T. H. (2001). ECG beat recognition using fuzzy hybrid neural network. IEEE Transaction on Biomedical Engineering 48: 1265–1271.

[101] Moon, S. W., and Kong, S. G. (2007). Block-Based Neural Networks for Personalized ECG Signal Classification. IEEE Transaction on Neural Networks 18: 1750-1761.

[102] Karimifard, S., and Ahmadian, A. (2007). Morphological heart arrhythmia classification using Hermitian model of higher-order statistics. In Proceeding of International Conference Engineering in Medicine and Biology Society (EMBS), pp. 3132-3135.

[103] Hu, Y., Palreddy, S., and Tompkins, W. J. (1997). A patient-adaptable ECG beat classifier using a mixture of experts approach. IEEE Transaction on Biomedical Engineering, 44: 891–900.

[104] Osowski, S., Hoai, L. T., and Markiewicz, T. (2004). Support vector machine based expert system for reliable heartbeat recognition. IEEE Transaction on Biomedical Engineering 51: 582–589.

[105] Shyu, L. Y., Wu, Y.H., and Hu, W. (2004). Using wavelet transform and fuzzy neural network for VPC detection from Holter ECG. IEEE Transaction on Biomedical Engineering 51: 1269-1273

[106] Khadra, L., Al-Fahoum, and Binajjai. (2005). Quantitative analysis approach for cardiac arrhythmia classification using higher order spectral techniques. IEEE Transaction on Biomedical Engineering 52: 1840-1845.

[107] Andreao, R.V., Dorizzi, B., and Boudy, J. (2006). ECG signal analysis through hidden Markov models. IEEE Transaction on Biomedical Engineering 53: 1541-154.

[108] Mitra, S., Mitra, M., and Chaudhuri, B. B. (2006). A rough set-based inference engine for ECG classification. IEEE Transaction on Instrument and Measurement 55: 2198-2206.

[109] Chazal, F. De, and Reilly, R. B. (2006). A patient adapting heart beat classifier using ECG morphology and heartbeat interval features. IEEE Transaction on Biomedical Engineering 53: 2535-2543.

[110] Inan, T., Giovangrandi, L., and Kovacs, J. T. A. (2006). Robust neural network based classification of premature ventricular contraction using wavelet transform and timing interval features. IEEE Transactions on Biomedical Engineering 53: 2507-2515.

[111] Melgani, F., and Bazi, Y. (2008) Classification of electrocardiogram signals with support vector machines and particle swarm optimization. IEEE Transactions on Information Technology in Biomedicine 12: 667-677.

[112] Michael, S., Schlosser, M., Schnitzer, A., and Ying, H. (2008). Online cardiac arrhythmia classification by mean of circle maps analysis implemented on an intelligent miniaturized sensor. In Proceeding of International Conference Engineering in Medicine and Biology Society (EMBS), pp. 1627-1630.

[113] Recommended practice for testing and reporting performance results of ventricular arrhythmia detection algorithms. (1987). Association for the Advancement of Medical Instrumentation, Arlington, VA.

[114] Mark R. and Moody G. MIT-BIH Arrhythmia Database Directory [Online]. Available from: www.physionet.org

[115] Jewajinda, Y., and Chongstitvatana, P. (2006). A cooperative approach to compact genetic algorithm for evolvable hardware. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 624–629.

[116] Potter, M. A., and De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. In Proceeding of the Third Parallel Problem Solving From Nature, pp. 249–257.

[117] Sipper, M. (1997). Evolution of parallel cellular machines: the cellular programming approach. Berlin: Springer-Verlag.

[118] Betz, V., Rose J., and Marquardt, A. (1999). Architecture and CAD for Deep- Submicron FPGAs. Boston: Springer.

[119] De Jong K. A. (1975). An analysis of the behavior of a class of genetic adaptive systems. Doctoral Dissertation, University of Michigan, Ann Arbor, Michigan

[120] Merchant, S. et al. (2006). FPGA implementation of evolvable block-based neural network. In Proceeding of IEEE Congress on Evolutionary Computation, pp. 3129–3136.

[121] Jiang, W., Kong, S. G., and Peterson, G. D. (2006). Continuous heartbeat monitoring using evolvable block-based neural networks. In Proceeding of International Joint Conference on Neural Network (IJCNN), pp.1950–1957.

[122] Jiang, W., Kong, S. G., and Peterson, G. D. (2005). ECG signal classification with evolvable block-based neural networks. In Proceeding of International Joint Conference on Neural Networks (IJCNN), pp. 326–331.

[123] Gopalakrishnan, R., Acharya, S., and Mugler, D. H. (2004). Real time monitoring of ischemic changes in electrocardiograms using discrete Hermite functions. In Proceeding of International Conference Engineering in Medicine and Biology Society (EMBS), pp. 438-441.

Biography

Yutana Jewajinda received B.Eng degree in Computer Engineering from King Mongkut's University of Technology Thonburi in 1992 and MS. Degree in Computer Engineering from University of South Carolina, Columbia, USA. He was in Doctoral program in Computer Engineering at University of Southern California, Los Angeles, USA before returning to Thailand. Since then He has been working for National Electronics and Computer Technology Center in area of VLSI design and Embedded Systems. He has been in Doctoral program in the department of computer engineering at Chulalongkorn University in 2004.