Embedded evolvable fuzzy hardware systems for cell flow scheduling

Li, Ju Hui

2006

Li, J. H. (2006). Embedded evolvable fuzzy hardware systems for cell flow scheduling.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4644>

<https://doi.org/10.32657/10356/4644>

Nanyang Technological University

Downloaded on 26 Apr 2023 18:51:52 SGT

Embedded Evolvable Fuzzy Hardware Systems

For

Cell Flow Scheduling

Li Ju Hui

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University

in fulfillment of the requirement for the degree of

Doctor of Philosophy

2006

ACKNOWLEDGMENTS

Here I express my deep appreciations to my supervisor Assoc Prof. Lim Meng Hiot for suggesting this promising research area, his valuable guidance, constructive suggestions, helpful criticism during the whole period of my research. His concerns on my life, on my research progress and on my course work kept pushing me closer and closer to my objective. His supervision built me up in almost all aspects including research attitude, research philosophy, English, Copyright, living attitude and so on. All these will help me very much in my onward whole life.

I would like to thank Mr. Sia Liang Poo, Mr. Goh Mia Yong, Mr. Seow Yong Hing William and Miss Lim Tan Gek Eng for their kindly help on software, hardware, management of the laboratory utilities and other technical supports.

I also want to say thanks to my juniors Ng Wil Lie, Cao Qi, Xu Yi Liang, Tang Jing and Agarwal Amit for their concerns and help during my study.

Finally, I wish to express my appreciations to my wife, my parents, my parents-in-law and my younger brother for their supports, understanding and encouragements.

Summary

Evolvable hardware (EHW) has become a promising branch of research in the area of hardware architecture. However, there remain many issues to be tackled before EHW can be applied more extensively in real-time problems. The three major open issues are *on-line adaptation*, *scalability* and *evolution termination*.

In this thesis, we propose the concept of an intrinsic evolvable and on-line adaptive *evolvable fuzzy hardware* (EFH). It can be viewed as a new type of EHW. Adopting our proposed EFH, the three major open problems can be addressed. In particular, EFH has great potential for solving real-time control problems efficiently. In order to demonstrate our idea, we apply the EFH to tackle the problem of ATM cell-scheduling in two different contexts: two-input cell scheduling and multiple-input cell scheduling.

For ATM cell-scheduling, when the scenarios of cell flow in the network change dramatically, traditional scheduling algorithms such as *first-in-first-out*(FIFO), *static priority weighted round robin* (WRR), *weighted fair queuing* (WFQ), *worst-case weighted fair queuing* (WF2Q) which employ static switching scheme may see their efficiency deteriorate.

A common trait of these algorithms is that their mechanisms are fixed, and they cannot adapt to changes in traffic flow efficiently. Benefiting from the evolutionary scheme, the fuzzy switching system with EFH scheme can track the changes in the pattern of traffic flow and maintain the desired level of system performance. A

very significant advantage of the proposed EFH is that the system performance can be intuitively tuned by a human operator through parametric adjustment of the fitness function. This advantage gives the EFH system a very special property that conventional scheduling methods cannot fulfill easily.

To support the development of EFH, a special purpose hardware known as *reconfigurable fuzzy inference chip* (RFIC) has been developed in a separate research initiative. The RFIC has the capability of online context switching with zero overhead. With the RFIC as the core to support context independent fuzzy inferencing, the whole EFH system including the genetic algorithms operations and the evaluation blocks is implemented as a System-on-Chip (SoC).

Table of Contents

Summary	i
List of Tables	ix
List of Figures	xiii
1 Introduction	1
1.1 Evolvable Hardware	1
1.2 Motivations	9
1.3 Objective	12
1.4 Major Contributions	13
1.5 Report Organization	14
2 ATM Traffic Scheduling	16
2.1 Cell Flows	17

2.2	Quality of Service	18
2.3	Existing Cell Scheduling Schemes	18
3	Genetic-Fuzzy Learning	23
3.1	Genetic Algorithm	26
3.1.1	Coding	27
3.1.2	Genetic Operations	27
3.2	Fuzzy System Overview	33
3.3	Genetic Coding of a Fuzzy System	34
4	Intrinsic Evolvable Fuzzy Hardware System	37
4.1	Solutions to open issues of EHW	39
4.2	Evolution scheme	43
4.2.1	Genetic coding	44
4.2.2	Inference scheme	46
4.2.3	Fitness function	47
4.2.4	Reconfigurable Fuzzy Inference Chip	48
5	Simulation Scheme	53
5.1	Simulation Architecture	53

5.2	Cell Flow Generation	56
5.3	Simulation Cases	59
5.3.1	Two-Channel Simulation	59
5.3.2	Multi-Channel Simulation	60
6	Two-Channel Simulation	62
6.1	Comparison	63
6.2	Tunability of EFH	72
6.3	Simulation on Realistic Traffic Scenario	75
7	Multi-Channel Scheduling	80
7.1	Evolution Scheme	82
7.2	Fuzzy System	84
7.2.1	Fuzzy Terms	87
7.2.2	GA Encoding	88
7.3	Fitness Function	88
7.4	Simulations	91
7.4.1	Computation Complexity Analysis	92
7.4.2	Normal QoS Performance	95

7.4.3	QoS Tunability	100
8	Hardware Implementation of EFH	102
8.1	System Description	105
8.2	Hardware Architecture	107
8.2.1	Hardware Architecture of Evolution System	107
8.2.2	Sort Cell	109
8.2.3	Crossover Cell	112
8.2.4	Mutation Cell	115
8.2.5	System Controller	116
8.2.6	Evaluation Cell	119
8.2.7	Simulation and Synthesis	126
9	Conclusions and future directions	131
10	Author's Publications	136
10.1	Journals and Book Chapters	136
10.2	Conferences	137
10.3	Statements	138
10.4	Others	139

11 Symbols and Acronyms	140
Bibliography	142

List of Tables

3.1	Simple crossover operation	28
3.2	Mutation operation	29
3.3	Linguistic matrix of the maximum rule set for a system with 2 input variables	35
3.4	Example of fuzzy rule set in a control system	35
4.1	A 25-rule fuzzy system for ATM cell scheduling	46
4.2	FIM content in $PB_{<1,1>}$	51
7.1	A startup rule set	88
7.2	Definition of variables	90
7.3	Variables description	93
7.4	Computation complexity of algorithm1 & algorithm 2	93
7.5	Average delay for scenario ₁ , scenario ₂ and scenario ₃	97

7.6	Bandwidth utilization ratio in scenario ₁ , scenario ₂ and scenario ₃ . . .	98
7.7	The tuned average delay for scenario ₁	101
8.1	Pseudo roulette wheel	113
8.2	Descriptions of each state	117
8.3	Signal description	121
8.4	Contents of PB ₁₁	125

List of Figures

2.1	Multiplexer scheme	16
2.2	Description of CBR	17
2.3	Description of VBR	18
3.1	Roulette wheel	30
4.1	Adaptation framework for EFH	40
4.2	Membership functions for c_1 and c_2	45
4.3	Membership functions for T and F	45
4.4	Block architecture of RFIC	52
5.1	Simulation architecture	55
5.2	Description of CBR	56
5.3	Description of VBR	56

5.4	Cell Flow Characteristic	56
5.5	Data format	57
5.6	Simulation flow diagram	58
6.1	Cell delay for $class_1$ in $scenario_1$	64
6.2	Cell delay for $class_2$ in $scenario_1$	65
6.3	Cell loss for $class_1$ in $scenario_1$	66
6.4	Cell loss for $class_2$ in $scenario_1$	67
6.5	Cell delay for $class_1$ in $scenario_2$	68
6.6	Cell delay for $class_2$ in $scenario_2$	69
6.7	Cell loss for $class_1$ in $scenario_2$	70
6.8	Cell loss for $class_2$ in $scenario_2$	71
6.9	Class ₁ cell delay for Scenario ₁	73
6.10	Cell loss for Scenario ₁	74
6.11	Class ₁ cell delay for Scenario ₂	74
6.12	Cell loss for Scenario ₂	75
6.13	Cell flow for scenario ₃	76
6.14	Class ₁ delay for Scenario ₃	78

6.15	Class ₂ delay for Scenario ₃	78
6.16	Class ₁ cell loss for Scenario ₃	79
6.17	Class ₂ cell loss for Scenario ₃	79
7.1	The system architecture for multi-channel scheduling	83
7.2	The architecture of the <i>scheduling model</i>	84
7.3	The membership functions for $c_{\#}$	87
8.1	System architecture	106
8.2	Membership functions	107
8.3	GA system architecture	110
8.4	Architecture of RNG _#	111
8.5	The architecture of sort cell	111
8.6	Crossover Cell	114
8.7	Mutation cell	115
8.8	Interface of finite state machine	117
8.9	Finite state machine	118
8.10	Evaluation cell implementation	120
8.11	Architecture of fitness calculation	122

8.12 The architecture of RFIC	123
8.13 The configuration map of the FIM block	125
8.14 The architecture of OAM	126
8.15 Simulation results of GA block for $\alpha=0.3$	128
8.16 Snapshot of evolution process for $\alpha=0.3$	129
8.17 Simulation results of GA block for $\alpha=0.5$	129
8.18 Snapshot of evolution process for $\alpha=0.5$	130

This page is intentionally left blank.

Chapter 1

Introduction

1.1 Evolvable Hardware

In recent years, hardware design and implementation by means of biologically inspired methodologies have attracted much attention. The idea behind bio-inspired hardware is either neural network, embryonics, or evolutionary algorithms.

Neural network (NN) system is a different computing form than the traditional Neumann machine. It simulates the parallel architecture of human brains. It is composed of many simple process unit which simulates the human neurons. Each process unit is connected with its surrounding units through an adaptive connection and works parallelly with the other units. Some parameters on the connections of the units can be adapted during the training process to make the system performance converge to the required standard. Neural network system can be implemented in

software system and hardware system. The major objective of the adaption in NN is the parameter evolution other than the evolution of the hardware architecture as in the following two schemes.

Embryonics is focused on hardware implementation to simulate the cell architecture of living beings [1–4]. Most of the research works on hardware implementation using embryonics try to solve problems through self-repairing and self-replication. For example in [2], the authors proposed a new *field programmable gate array* (FPGA) structure. This new architecture consists of at least of two levels: *molecular* and *cellular* level. The cell in the FPGA consists of 6 molecules. The architecture of the molecules is the same. At any specific time, there are always some spare molecules. If some working molecules do not work, the spare ones will be used to replace the failed ones. This can be viewed as a form of fault tolerance at the molecular level. At the cellular level of self-repairing, there are also some spare cells that can be used to substitute the faulty ones. The naming of the functional components in embryonics suggest similarity in architecture to that of living beings, except that it is in a much simplified form. This scheme is effective to do self-repairing to some degree. Besides self-repairing, there has been research which focus on self-replication mechanisms, for example in [3].

There are many algorithms inspired by the evolution of natural life. One of them is *genetic algorithm* (GA). GA is inspired by Darwin’s theory of evolution. The representation of solution structure in GA is referred to as a chromosome. The main idea of GA is that by combining two good-performing chromosomes, the probability of

deriving a better performing chromosome is higher. In traditional ways of designing systems and circuits, we usually depend on our knowledge and experience. All the solutions we have can be called the known space. In reality, there is also a very large space that is still unknown. Typically, there may exist better solutions in the unknown space which may be very time-consuming and difficult or even impossible to search using traditional ways. With GA, we can carry out search in a big unknown space. This is the basis of the application of GA. When applied for hardware design, it extends to a class of bio-inspired hardware implementation referred to as *evolvable hardware*(EHW) [5,6] .

EHW can be classified into *extrinsic* or *intrinsic* EHW, based on the type and nature of evolution. Extrinsic EHW relies on a simulated evolutionary process. It is independent of the hardware. Extrinsic EHW may rely on hardware description languages (HDL) , C, C++ or other languages to represent circuits and then rely on an evolutionary algorithm to evolve the hardware architecture. Only the elite design will be implemented or downloaded into the reconfigurable device. Intrinsic evolvability means that evolution will be done at the hardware level and evaluation will also be carried out in the EHW. In principle, intrinsic EHW can modify its own hardware architecture and behavior autonomously. If the environment changes, the behavior or architecture will also change, and then the system performance will be maintained at an acceptable level. So far, extrinsic EHW and intrinsic EHW have achieved much progress [7–13], but considerable research still needs to be carried out in this area.

Some researchers do not recognize extrinsic EHW as real EHW in the structural sense. They only regard it as a method to design circuits or systems [14,15]. There is also research that focuses on intrinsic EHW. In some papers, the researchers partitioned the evolutionary model in software and the fitness evaluation component in hardware. This can be called off-line adaptation intrinsic EHW. In this type of EHW, adaptation takes place during a learning phase that is decoupled from the real physical operating environment. To date, most published work concerning intrinsic EHW is of a form similar to those in [16–18]. For example in [16], the authors used intrinsic evolution to carry out robot navigation. The whole process can be divided into two phases, the learning phase and the working phase. The system evolves a circuit first and then carries out evaluation. The best performing chromosome will be finally downloaded into the system. This procedure is referred to as the learning phase. After that, the system carries out the control work. This is referred to as the working phase. The circuit architecture cannot be modified when it is carrying out the control task. This type of EHW is generally better than extrinsic EHW. However, it is still very difficult to adopt it in real-time problems, a promising area for the application of EHW. To address this issue, we propose a form of on-line adaptative and intrinsic evolvable EHW. Our proposed EHW is referred to as *evolvable fuzzy hardware* (EFH), which carries out evolution intrinsically and carries out adaptation on-line. The significance of the EFH will be clear when we consider it from the evolution granularity perspective.

From the perspective of evolution granularity, EHW can be classified into at least

four types; transistor level, gate level, function level and fuzzy rule level. At the transistor level, the smallest component in the genotype is a transistor. This will give the evolution the most flexibility because transistor is the smallest component of a circuit. There are at least two modes of evolution at this level. One mode of evolution adapt by adjusting the characteristic parameters of the transistors to develop new functional circuits [17, 19]. For example, in [17], the authors succeeded in using their proposed CMOS FPTA (*field programmable transistor array*) chip to evolve NAND, NOR, AND, OR and XOR gates. All the evolved gates are designed using special transistors where the parameters, particularly the *width* (W) and *length* (L) can be tuned within a certain range. These tunable transistors were referred to simply as programmable transistors. The values of W and L can be 1, 2, ..., 15 μm and 0.6, 1, 2, 4, 8 μm respectively. The connections of these transistors can also be tuned within a tight range. Another adaption mode is to adjust the connectivity of the standard transistors [20, 21]. In [21], the authors evolved an amplifier, a computing circuit to exhibit a Gaussian voltage-input current-output characteristic, a filter and so on. In these evolution processes, GA were used to tune just the connectivity of the transistors. Each transistor can be connected to any other transistors. This characteristic is significantly different from the previous evolution scheme.

Granularity at the gate level means that gates are the smallest components of the EHW [22–28]. In [24], the authors designed a gate-level EHW to control an artificial hand. They implemented the EHW on a single chip. The chip includes a GA unit,

a PLA (*programmable logic array*) unit, registers and a control unit. The chip was interfaced to an external dual-port RAM and a host PC (*personal computer*) . The memory was used to store training data, chromosome and fitness value. In the application, the host PC downloads the training data and carries out the fitness computation. The hardware performs some GA operations. The time it took to evolve using the proposed EHW chip for one evolution was $94.8 \mu\text{s}$, much faster than using software running on a PC or workstation. The paper [24] can be looked upon as the latest development of gate level EHW. It is the first EHW whereby parts of the GA and parts of the evaluation component were implemented in one chip [24].

Granularity at the functional level means evolution of macro units (adder, multiplier, sine, cosine, etc.) components [8,22,29,30]. In [8], the authors proposed a functional EHW to perform ATM cell scheduling. There are 100 *functional processing units* (FPUs) in the FPGA chip. Each FPU can be configured to perform one of the high-level functions such as addition, subtraction, multiplication, division, sine, and cosine [8]. The functions and connections of the FPU were configured based on the elite chromosome. The authors adopted this type of device to evolve a circuit to calculate proper service rate in order to achieve good *quality of service* (QoS) performance. After 2500 iterations, the authors succeeded in getting a final circuit to calculate a satisfactory service rate.

Our proposal involves evolution granularity at the fuzzy rule level. The evolution is mainly done on the fuzzy rule level. When the fuzzy rule set changes, the archi-

texture of the inference hardware should be able to incorporate the changes in the domain rule set. The reconfigurable hardware in this system is called *reconfigurable fuzzy inference chip* (RFIC) which can accommodate various kinds of fuzzy rule sets without significant reconfiguration overhead [31–33]. Here, it should be clearly spelled out that the proposed *evolvable fuzzy hardware* (EFH) is different from traditional evolvable fuzzy system. In traditional evolvable fuzzy system, the system evolves toward an exact fuzzy rule set or membership function or some predefined control parameters. Ultimately, the system will be implemented in hardware or an embedded sub-system to carry out the control function. In any case, the fuzzy rule set and the real hardware in the traditional fuzzy system do not change. This is essentially the main difference between EFH and traditional EFS (evolvable fuzzy system) .

There are currently three types of devices that can be used to implement EHW. Transistor level EHW uses FPTA to do circuit evolution [17]. FPGAs can be used in function level EHW. FPGA can also be used in some EHW that has no definite components. For example in [34], the authors used Xilinx 6200 FPGA to evolve digital circuits. In these kinds of EHW, each chromosome represents the configuration of the device. It does not represent the connection relationship of some definite components. The advantages of Xilinx 6200 FPGA are as follows. Firstly, it has microprocessor interface which is useful for application development. The second is that it can be partially reconfigured. This aspect allows the configuration of only a small part of the FPGA instead of the whole device. This feature makes it more

convenient to implement EHW. The third advantage is that it offers safe configuration. This characteristic means that any bit strings can be downloaded into the device without the risk of damaging the device. It is inevitable that the evolved chromosomes may often have incorrect programming bits which may cause short circuit between power and ground. The safe configuration of Xilinx 6200 FPGA is thus very useful to the implementation of EHW. But unfortunately, Xilinx stopped manufacturing this kind of device in May 1998. So researchers have to find other ways to implement this kind of EHW. In [35], the authors designed a new circuit suitable for implementing EHW. The proposed circuit satisfies the requirements of realizing EHW.

The reasons that EHW relies heavily on a general computing machine can be ascribed to the complexity of the systems. Without general computing platforms, only small and simple systems can be evolved efficiently. In order to address such issues, a method referred to as *increased complexity evolution* has been proposed [36–42]. The main idea of this method is to divide a complex system into many simple sub-systems and evolve the small sub-systems. The whole network consisting of the sub-systems can also be evolved if necessary. This is called second-level evolution. Some experimental results and comparisons on evolving character recognition systems were given. The results show that this method can dramatically decrease the time spent on evolution and evaluation. From the viewpoint of granularity, evolution at the sub-system level can be looked upon as a form of first-level evolution granularity. From transistor level [17], gate level [22, 24, 28], functional level [8, 22]

to sub-system level [36], the granularity level of EHW is increasing. According to this tendency of change in granularity, our proposed EFH also follows the concept of *increased complexity evolution*.

In order to test and demonstrate our proposed EFH framework, we choose ATM (asynchronous transmission mode) cell scheduling as our domain problem. This problem was also adopted by *Liu et al.* [8] in their function level EHW. By adopting EFH on ATM cell scheduling, the scheduler is enhanced by not only keeping consistent QoS for the different cell flows but also exhibiting good QoS tunability.

1.2 Motivations

Fuzzy logic is a powerful methodology in the area of embedded control. It has been widely used in many real time applications since its inception. Cell flow scheduling is also a real time control problem, hence ideal for fuzzy control. But it can be tedious and complicated to manually formulate the fuzzy rule set for efficient cell scheduling for all scenarios. In this respect, our proposed EFH is very appropriate. With EFH, a good fuzzy rule set can be consistently maintained. In many fuzzy systems, rule set is manually derived based on intuition. This may limit the performance of the fuzzy system since the manually derived rule set may only be efficient for specific scenarios. Based on this premise, the principle behind EFH is therefore to evolve the rule set accordingly, which in this case is the changing patterns of application scenarios. The adaptation of the rule set can be achieved through symbolic hybridization of

an evolutionary algorithm embedded with fuzzy inference mechanisms [7, 43].

For ATM cell-scheduling, if the pattern of cell flow changes, the fuzzy rule set should be changed accordingly to maintain consistent performance. To achieve this, on-line evolution to adapt to the cell flow should be implemented. Ultimately, it is desirable to implement the system as a form of intrinsic evolvable fuzzy hardware [43–47].

There is much published research on the application of GA techniques for designing fuzzy systems. For example, in [7, 48–50], fuzzy systems or fuzzy logic controllers had been developed using GA or other evolutionary algorithms. In [7, 48, 49], the goal is to use genetic computing to design the fuzzy rule set, membership functions or both. After design, the fuzzy system will not change to adapt to its changing environment. In [50], the authors used a specified learning algorithm to select suitable parameter settings (for example, fuzziness degree, shape of the membership function, rule set index, as well as rule weight) among a large data set stored in the memory.

EHW has been attracting more attention in recent years. Significant research have been carried out on EHW to solve practical problems, e.g. micro air vehicles [51], robot navigation [16], ATM cell-scheduling [8], pattern recognition [11] and so on. Although EHW has seen much progress since its inception, there are some problems still remain to be addressed. Some of the open issues are *on-line adaptation*, *scalability*, as well as *termination of evolution* [14]. We briefly discuss these three issues.

On-line adaptation means that the system hardware adapts during normal opera-

tion. Significant research work has been done to realize interactive on-line adaptation or off-line adaptation. In [52] and [53], the system has two working modes, training mode and operation mode. In the training mode, GA is fired and evolution takes place. After the evolution, the best solution found will be adopted by the operation mode to carry out the normal control. In these systems, the working mode switches from training mode and operation mode and vice versa. In this kind of on-line adaption, the normal system operation has to be interrupted for evolution. In [16], the best chromosome after specific generations of evolution was downloaded into the FPGA after the off-line evolution. In [8], only the best architecture can be downloaded into the FPGA chip. Off-line adaptation has a significant shortcoming. After the elite (the best chromosome found) has been downloaded into the hardware, the system cannot change in order to adapt to the environment. Otherwise, the system would have to stop its normal operation to wait for the evolution. On-line adaptation is very hard to realize because the system has to reconfigure every chromosome to evaluate it. Some chromosomes may have very poor performance. If these chromosomes are evaluated by reconfiguring the hardware, there may potentially result in some damage or disasters [14].

Regarding the open issues in EHW domains, *Scalability* basically refers to the length of the chromosome. In [17], the researchers try to evolve some analog circuits on the FPTA chip. The cell matrix is 16×16 ; hence the chromosome has to be $16 \times 16 \times 24$. This implies a very large search space. In [8], the system implementation relies on function level EHW to realize the ATM cell scheduling. The chromosome length is

100. However, some of the genes are decimal numbers, hence the search space is also very large.

Termination of evolution pertains to stoppage of evolution, for example, the number of runs [14]. Although only three issues are mentioned, there are still many other open issues in EHW research. A detailed discussion on the issues pertaining to EHW can be found in [14]. Our proposed EFH in this dissertation holds great promise for addressing some of these issues. We demonstrate its applicability by using the framework to solve the ATM cell-scheduling problem. In this dissertation, we show that the issues of *adaptation*, *scalability* and *termination* are well addressed by our EFH framework. The details will be presented as discussion in Chapter 4.1.

1.3 Objective

For our research, we specify four goals that we seek to achieve. First, we will propose some conceptual ideas on how to solve some of the open issues of EHW. Second, we will demonstrate the viability of our EFH framework for solving real-time practical problems. The system performance with EFH is comparable with traditional schemes. Third, we will explore techniques for achieving on-line adaptation and intrinsic evolvable EFH which works independently. On-line adaptation means that our EFH system can adapt to the changing environment without stop its normal function. Intrinsic evolution refers to the evolution carried out by the system itself without relying on external powerful computation platform. Finally, we will explore

the feasibility of implementing our proposed EFH in hardware using SoC (system on chip) technology.

It is emphasized that a central theme of our work is to develop a framework for the implementation of an on-line adaptation and intrinsic evolvable hardware system for real-time applications. Cell-scheduling problem is a problem domain we adopt to demonstrate the viability of the on-line adaptative and intrinsic evolvable EFH.

1.4 Major Contributions

The outcome of our work has been positive thus far, resulting in several significant contributions. The contributions of this thesis can be concluded as follows.

- The framework of our proposed EHW is based on using fuzzy rules as the evolution granularity. This effectively addresses the current outstanding issues on *scalability* and *termination* pertaining to EHW (discussed later in Chapter 4).
- With our proposed EFH framework, it is feasible to realize on-line adaptation and intrinsic evolvable EFH completely within a single chip. The EFH system is able to carry out evolution and evaluation without relying on any external computation platform. This makes it possible that EHW can be implemented as a very compact system and embedded into other systems.
- The proposed EFH can be directly used for real-time applications. ATM cell-

scheduling is adopted as a problem domain to demonstrate this viability. The application of EFH can be extended to other real-time areas such as audio signal processing, video signal processing, image processing, etc.

- We developed and simulated the whole EFH using Verilog Hardware Description Language. The whole EFH system is synthesized by Synopsys Design Compiler and the chip size is smaller than a state of the art of microprocessor. This can be deemed as the first EHW system that can realize on-line adaptation and intrinsic evolution.

1.5 Report Organization

This chapter has given an overview of the project by presenting the essential background on EHW. The motivations, objectives and contributions of this project are also described. The remaining chapters of this thesis will be organized as follows.

In chapter 2, some literature survey on cell scheduling schemes are presented. GA and fuzzy system are two important component of the EFH. In chapter 3, some basic introduction on genetic algorithm (GA), fuzzy systems and genetic-fuzzy learning are presented. In Chapter 4, our proposed EFH in the context of two-channel ATM cell-switching problem will be described in detail. The methods of solving the open issues mentioned above will also be discussed. Chapter 5 describes the simulation scheme for the performance simulation of EFH. Chapter 6 will give results of simulating our EFH applied to two-channel ATM cell scheduling. Some comparisons

with traditional scheduling schemes are also given in this chapter. Chapter 7 will present the application of EFH in the context of multi-channel ATM cell scheduling problem which is the extension of two-channel scheduling problem. Some simulation results and comparisons with traditional schemes are presented. Chapter 8 will describe the hardware implementation of the whole EFH system to solve the ATM cell scheduling problem. Finally, Chapter 9 presents conclusion of the research conducted thus far and outline some aspects of future work.

Chapter 2

ATM Traffic Scheduling

Multiplexer is an important component in network system for the purpose of bandwidth utilization. In order to save bandwidth resource, some input channels can be multiplexed to an output channel through time-division scheme. The multiplexer with two input flows can be described by Fig. 2.1.

Figure 2.1: Multiplexer scheme

$class_{\#}$ represents the input cell flows. $BUF_{\#}$ is for buffering the arriving cell units.

MP is the multiplex unit to schedule the cell units to the OUT channel. Switch

Control block represents the scheduling schemes.

2.1 Cell Flows

$class_1$ and $class_2$ in Fig. 2.1 can be different network traffics. The major part of the network traffic can be CBR (*Constant Bit Rate*) traffic and VBR (*Variable Bit Rate*) [54] .

CBR always reserves a static bandwidth, and it can transmit cells at the *peak cell rate* (PCR) for any duration of time. CBR is normally used to support telephone, video conference and entertainment video. VBR is for the services that require variable bandwidth resource. It is mainly characterized by the PCR and *maximum burst size* (MBS) . The cell flow characteristic of CBR can be modelled by Fig.2.2. VBR can be modelled by a two-state *Markov chain model* in which the cell flow has ON (*burst*) period and OFF (*silence*) period [8, 55–59]. The source transmits cells equidistantly during the ON period while keeping silent during the OFF period. VBR's cell flow characteristic is depicted by Fig.2.3. The cell bit rates for different ON periods can vary randomly.

Figure 2.2: Description of CBR

CBR represents the flows which requires constant bit rate and VBR represent the

Figure 2.3: Description of VBR

burst traffic flow.

2.2 Quality of Service

The major factors of Quality of Service (QoS) for ATM traffic flow can be described by *cell loss* and *cell delay* [60]. *Cell loss* is the number of the cells being dropped by the multiplexer due to the congestion. *Cell delay* refers to the average waiting time of the cells staying in the buffer before being transmitted. Different class of service has different QoS requirement. For example, real-time traffic of voice and video is sensitive to delay and data service such as email system is delay insensitive and loss sensitive.

2.3 Existing Cell Scheduling Schemes

Many algorithms have been designed to solve the problem of cell scheduling. The common switching schemes are *first-in-first-out* (FIFO) , *static priority* (SPR) , *dynamically weighted priority scheduling*(DWPS) [57], *round robin* (RR) , *weighted round robin* (WRR) , *generalized processor sharing* (GPS) [61] , *minimized cycle round robin* (MCRR) [62], *weighted fair queuing* (WFQ) [63], *worst-case fair*

weighted fair queuing (WF2Q) [64] and so on. FIFO is a very simple approach for ATM cell scheduling. It is easy to implement in hardware, but it is not very good in terms of QoS performance. It is also called *first-come-first-serve*. FIFO schedules cell units based on their arrival time. The oldest in the buffer will be serviced first. SPR is also a simple method to control cell scheduling. Each input channel has its own fixed priority assigned by the system. The cell flow with the highest priority is always serviced if there are cells in its buffer. The channels with lower priorities can only be serviced when the buffers for the channels with higher priorities are empty. This scheme tends to be unfairly biased towards one cell flow [43, 57]. DWPS is a significant improvement over the static SPR scheme. It adjusts the priority according to the cell flow scenarios [57]. Each input channel is pre-assigned a priority at the beginning of the process. During the scheduling process, the priority of each flow is adjusted according to the waiting time of the oldest cell in the buffer. The longer the head cell stays in the buffer, the higher the priority the channel may get according to the adjustment. According to simulations, the adaptation scheme is too simple and may not be very efficient if the cell flows change dramatically. DWPS involves high computation cost during the enqueueing process.

GPS is an idealized scheduling scheme using a *fluid model* [61]. It is assumed that every input channel is serviced by an independent server simultaneously with a reserved link capacity. GPS has two significant properties, appropriate delay bound and fair bandwidth allocation. Although GPS has significant properties, it is not realizable due to the ideal fluid model it adopts. Nevertheless, GPS is used by

many algorithms as a benchmark to generate time stamps. WFQ, also called PGPS (*packet-by-packet generalized processor sharing*) is a scheme which selects cells based on the simulation of the GPS system [63]. The cell that finishes its service first in the GPS system will be selected for transmission. This system can guarantee a service of no bigger than one packet for each cell than the GPS system does. It also guarantees that the total bits serviced for a session in a period is not one packet size more than GPS does. The most significant disadvantage of WFQ is that it may service the cell much earlier than the GPS system. In order to overcome this disadvantage, *worst-case fair weighted fair queuing* (WF2Q) has been proposed [64]. WF2Q can approximate to GPS system very effectively. It chooses packets which have already started services in GPS system and which will depart the server first. By adopting this selection scheme, WF2Q cannot be far ahead or far behind of GPS. WFQ and WF2Q needs to simulate GPS system during the scheduling process, thus it involves very high computation cost during the en-queuing process which may affect the process speed. Other schemes such as *virtual clock* [56], *start-time fair queuing* (SFQ) [65] and *self-clocked fair queuing* (SFCQ) [66] use time stamps to control the sequence of cells transmission.

In RR approach, each input channel is serviced according to a pre-determined quota. One channel is serviced for the quota, and then the second channel is serviced, and so forth. WRR is a modified RR scheme [67]. It is a reconfigurable scheduling scheme. Each input channel is assigned a weighted value based on its reserved bandwidth resource. The weighted factor is the service quota each channel can acquire during

a service round. After its quota runs out, the channel will not be serviced any more until the next service round. WRR can strictly guarantee the bandwidth allocation ratio regardless of other misbehaving input flows. But its major drawback is its queuing delay and delay jitter. To overcome this, the MCRR (minimized cycle round robin) is proposed. It is an improvement of WRR in two aspects. First, the quota of each cell flow is determined in such a way that there is no common factors among them. The second improvement is that one cell from each cell flow is serviced during one visit (this modification is similar to WRR1 proposed in [68]) [62]. In order to maintain good isolation of ill effects from misbehaving cell flows and to achieve good delay guarantees without causing high computational complexity during the scheduling process, multi-class WRR has been proposed in [68]. In multi-class WRR, the cell flows are categorized as $group_1$, $group_2$, $group_3$ and so on, in the order of decreasing bandwidth ratio. For each round robin cycle, all the sessions in $group_1$ are serviced, followed by the first few sessions of $group_2$. In the second round robin cycle, the sessions from $group_1$ are serviced again, and then the left sessions of $group_2$, and then the sessions of $group_3$, and so on so forth. This is called *minicycle*. The maximum length of the *minicycle* is the smallest value of D_i , the maximum length of the round robin slot in which all the sessions of $group_i$ must be visited. The bandwidth ratio denoted by the parameter $\phi_i = \frac{1}{D_i}$, is allocated to every session of $group_i$. One major drawback of multiclass WRR is that ϕ_i cannot be an arbitrary value. For example, suppose ϕ_1 is 0.7, then D_1 is $\frac{1}{0.7} \approx 1.4$. But D_1 should be an integer, so D_1 can only be 1 or 2. If it is 1, $group_1$ will monopolize the complete bandwidth. If it is 2, $group_1$ will only utilize half of the bandwidth resource.

Fuzzy systems are also adopted to control cell scheduling in some papers. In [69], a fuzzy logic based cell scheduling system was proposed and compared with FIFO scheme. The simulation results showed that the proposed system significantly reduced wait time for high priority traffic and improved its QoS performance. In [70], the authors proposed a fuzzy system to provide good balance between drop and delay of non-real-time and real-time flows of differentiated services networks. Real-time and non-real-time flows are queued into queues with different sizes. The proposed fuzzy control system is compared with a crisp scheme. The simulation results in the paper indicate that better balance is achieved by the fuzzy system than the crisp scheme.

The FIFO, SPR, GPS based system, RR based system or the fuzzy logic based system mentioned above are nonflexible systems. Most of them are fixed schemes. Although some of them are adjustable, such as DWPS and WRR, they are not QoS oriented tunable algorithms. By tuning DWPS and WRR, only the tendency of increasing or decreasing of cell delay is determined, but how much the rough delay is cannot be known. If the evolutionary system is adopted, as presented in the following chapters, the system not only provides very good adaptation with respect to the changing flows, but also provides a very special advantage that the system can be tuned in an efficient way according to the QoS requirement.

Chapter 3

Genetic-Fuzzy Learning

This chapter will introduce some fundamental concepts of GA, fuzzy logic and genetic-fuzzy learning. GA is inspired by Darwin's theory of evolution. A major principle behind its functionality is that through some operations similar to that in population genetics, manipulations on two good performing solutions yield a better performing solution with high probability. GA work very well for combinatorial problems and exhibit a lower tendency of being trapped in a local optimum than some other search algorithms. The good performance of GA is derived from its intrinsically parallel computational search paradigm.

Fuzzy logic uses non-crisp sets to describe real-world situation. An object can belong to several fuzzy sets and the relationship between the object and the fuzzy sets can be described by the corresponding membership value. The bigger the membership value, the high degree the object belongs to a fuzzy set. A fuzzy rule is

an “if-then” conditional statement for making fuzzy inference. Fuzzification, fuzzy inference and defuzzification are three procedures in a fuzzy system. Fuzzification refers to conversion of crisp variables into fuzzy sets by using fuzzy membership function. Fuzzy inference is a process to fire several fuzzy rules to achieve a final fuzzy conclusion. Defuzzification converts the final fuzzy conclusion into a crisp variable, which is the final output or action for the plant that is being controlled.

There are many approaches for handling large-scale search problems. Some of them are *constraint based search*, *neural network* (NN), *Ant Colony Optimization* (ACO), *Genetic Algorithm* (GA) and so on. Among the above mentioned algorithms, *constraint-based search* is mainly used to optimize a linear function over a convex polyhedron based on some constraints. This scheme is mainly applicable for the operation researches such as factory profit maximization and bus company daily operations. NN system is a different computing form than the traditional Neumann machine. It has many simple process unit which simulates the human neurons. Each process unit is connected with its surrounding units through adaptive connections and works parallelly with the other units. Some parameters on the connections of the units can be adapted during the training process to make the system performance to converge to the required standard. NN can search locally other than globally like GA do. ACO is an optimization scheme inspired by routing philosophy of ant colony. In this scheme, a lot of search engine cooperate together to find a good solution. It is a parallel searching system as GA. But some of its intrinsic characteristics prevent it from being implemented efficiently in hardware. Each

individual needs to communicate with all the other entities to understand their histories and performances. Each entity needs to keep big storage for the record of its own history and performance which is called *artificial pheromone trail*. ACO is a non-jumping algorithm, each individual has to explore its adjacent terrain. The bigger the solution space, the longer the optimization time. ACO sometimes need to integrate local optimization schemes to improve overall system efficiency. GA has a parallel and jumping search scheme. Each chromosome can inherit its parents' architecture and performance without much communication. Compared with ACO, GA' individual needs much less memory storage.

Genetic-fuzzy learning adopts GA or genetic programming (GP) or other evolutionary algorithms to derive or search for appropriate membership functions, fuzzy rule sets or other elements. Genetic-fuzzy learning can be done in many ways. In [48], the authors adopted GA to evolve membership function shapes, membership function types, fuzzy rule set and the number of rules inside the fuzzy rule set. In [49], GA is used to generate an optimal or near optimal set of fuzzy rules and membership functions. In [50], the authors attempt to evolve contribution weights attached to the fuzzy rules. The weights determine how much the rule affects the final outcome of the fuzzy inference. There are also some other methods to do genetic-fuzzy learning. For example, it is also possible to evolve methods of defuzzification [50] for fuzzy systems. All these approaches are complicated and difficult to realize in hardware. However, from previous work on genetic-fuzzy learning [7,48,71], evolution involving only the fuzzy rule set can be effective in bringing about performance improvement

into the fuzzy system. To make our EFH easy to implement in hardware, and also to make sure EFH can be applied in real-time control area with good performance, we intend to choose evolution of fuzzy rule set as the basis of our genetic-fuzzy learning approach. This means that our EFH system will focus on exploring the potential power of achieving optimal performance through good fuzzy rule sets.

In this chapter, we begin with some concepts of GA. Then we will describe a typical fuzzy system in a fairly simplistic manner. The last part of this chapter provides details on how to incorporate GA into fuzzy systems and achieve the desired learning characteristics.

3.1 Genetic Algorithm

Genetic algorithm (GA) were first introduced in 1970 by John Holland. GA exhibits very strong performance compared to many human derived schemes in solving NP-hard problems. Problem coding or genetic representation, objective function and genetic operations are the three major issues pertaining to the application of GA in solving real-world problems. The following parts will briefly introduce these three major components.

3.1.1 Coding

Genetic representation of the problem being solved is generally referred to as the coding scheme. It basically refers to the representation of the solution structure as chromosome. A chromosome is a representation of a feasible solution of the problem and a collection of chromosomes during a GA run forms the population. Since each chromosome represents a solution, the chromosome should in some way contain information about the solution that it represents. One of the most commonly used methods of encoding is by means of binary string. An example of a chromosome could be “101100100110110”. The constructs of a chromosome are genes identified by its position within the string. The value expressed by a gene is referred to as the allele. Each bit in the string can represent certain characteristic of the solution.

There are of course, many ways of encoding. This depends mainly on the problem to be solved. For example, one can encode the solution structure directly using integers or real numbers. At times, it may be useful to encode the problem as a permutation string structure [72, 73].

3.1.2 Genetic Operations

In GA, the basic search and exploratory mechanisms basically rely on three operators; crossover, mutation and selection.

Crossover Crossover selects genes from parent chromosomes and creates a new offspring. The simplest and most common way to do this is to randomly choose the crossover point. To create a new offspring, the sub-string before this point is copied from the first parent while the sub-string after the crossover point is copied from the second parent. A simple illustration of crossover is clearly shown in Table 3.1 where “|” represents the crossover point.

Table 3.1: Simple crossover operation

Chromosome 1	11001 00100110110
Chromosome ₂	11011 11000011110
Offspring ₁	11001 11000011110
Offspring ₂	11011 00100110110

There are other ways to achieve crossover. For example, it is also common to choose multiple crossover points. Crossover can be rather complicated and depends very much on the encoding of the chromosome. The suitability of a specific crossover operation can only be judged based on the problem that the GA is trying to solve. In general, an effective crossover operation can significantly improve the overall performance of GA.

Mutation After a crossover is performed, mutation can take place. This is to prevent the solutions in the population from converging towards a local optimum. Mutation usually occurs with low probability and creates new offspring with a great degree of randomness. For binary encoding we can switch a few randomly chosen bits from 1 to 0 or from 0 to 1. An illustration of the mutation is as shown in Table 3.2.

Table 3.2: Mutation operation

Original chromosome ₁	110 1 111000011110
Mutated chromosome ₁	110 0 111000011110
Original chromosome ₂	1101100 1 001101 1 0
Mutated chromosome ₂	11011011001101 0 0

The specific manner in which mutation is carried out depends greatly on the encoding as well as the crossover. For example when we are encoding as permutation string structure, mutation can be achieved by randomly exchanging two genes.

Selection Chromosomes are selected from the population to be parents for crossover.

There must however be some mechanisms for deciding on which chromosomes to select for crossover. According to Darwin's theory of natural evolution, fitter individuals tend to have a higher chance of survival and create new offspring. It is therefore necessary to formulate mechanisms to evaluate the fitness of a chromosome. For GA, the fitness function forms a useful link between the real-life domain problem and the algorithm itself. It is a mechanism to measure goodness of the chromosome. The formulation of the fitness function depends greatly on the problem domain. With the fitness function as a basis for evaluation of chromosomes, what remain are the methods of selecting the chromosomes for crossover. Some examples of the methods for selecting parents include roulette wheel selection, Boltzman selection, tournament selection, rank selection and steady state selection. In general, the selection should favor chromosomes with higher fitness.

We provide an illustration for one particular mechanism of selection, namely the

roulette wheel selection. It is one of the most popular methods used. Parents are selected according to their fitness. The better the chromosomes are, the higher the chances of them being selected. Imagine a roulette wheel with slots sized according to the fitness of the chromosomes in the population. Each chromosome is given a slot according to its fitness value. A marble is thrown and the slot where the marble lands corresponds to the chromosome being selected. This way, chromosome with higher fitness will be selected more frequently. A simple illustration involving a population of 5 chromosomes is given in Fig 3.1.

Figure 3.1: Roulette wheel

Parameters of GA The dynamics of GA is very much determined by the parameters set by the user. There are three important parameters in GA, *crossover probability*, *mutation probability* and *elitism number*.

Crossover probability dictates how often crossover will be performed. If there is no crossover, an offspring is an exact copy of a parent. If crossover is to occur, the offspring will consist of parts from the parents. If the crossover probability is 100%, then all the offspring will be a result of crossing over of selected parents. If it is 0%, a whole new generation is made up of exact copies of chromosomes from the

old population. Crossover is made in the hope that new chromosomes will inherit good features of the old chromosomes and perhaps perform better. However it is also desirable to allow some parts of the population to survive into next generation.

Mutation probability determines how often parts of the chromosome will be mutated. If there is no mutation, offspring is taken after crossover (or copy) without any change. If mutation is performed, part of the chromosome is changed. If mutation probability is 100%, the whole population undergoes mutation. Mutation is made to prevent GA from falling into a local maximum in the search space, but it should not occur very often. Too high a mutation rate will render the GA to behave like a random search algorithm.

In order to prevent discarding good chromosomes in the next generation, it is common to adopt an elitism policy in GA. This means that a few best solutions will be copied directly from the old generation into the new generation without any changes. This number can be referred to as the elitism number.

Applications of GA Genetic Algorithm has been used to solve complex NP-hard search problems in machine learning and also for evolving simple programs. An obvious advantage of GA is their parallelism. Since GA works based on a population of individuals representing parallel search path through the solution space, it is less likely to be stuck in a local extreme like some other methods.

Another advantage of GA is that they are relatively easy to implement. Once we have configured a GA for a particular problem, it can be easily reconfigured to

accommodate a new problem. If the encoding is the same, all that is required is to change the fitness function. However, the choice of a suitable encoding and fitness function can be difficult and may require some degree of experimentation.

The main disadvantage of GA is its computational time requirement. It can be slower than some other methods. Nowadays, this problem is alleviated by the availability of faster computers and possibility of enhancing the speed of computation through parallel computing platforms.

There are many problems that have been solved successfully using GA. The following is a short list of some successful application areas of GA.

- Nonlinear dynamical systems - prediction, data analysis
- Designing neural networks, both architecture and weights
- Circuit Design
- Evolving LISP programs (genetic programming)
- TSP and sequence scheduling

GA can also be used in the design of fuzzy systems. One of the applications of GA in fuzzy system is searching for optimal fuzzy rule sets [7]. Designing fuzzy rule sets can be time consuming and tedious. By using GA, we need only to know how to evaluate the performance of the fuzzy rule set and how to get the training data. This is the basic requirement for achieving a good performing fuzzy system. The application of GA in fuzzy system will be outlined in the following section.

3.2 Fuzzy System Overview

A fuzzy system can be formally defined as an application or system, which employs a fuzzy control algorithm. In general, the fuzzy control algorithm refers to a set of *if-then* rules with linguistic values and variables. The values are specified as fuzzy concepts which are defined by membership functions. Herein, we will use the term fuzzy system to implicitly mean a set of rules and membership functions.

Consider a fuzzy system having m input variables $x_1, x_2, \dots, x_m$ and single output control variable y . A typical rule set will be “*if* $\langle x_1 \text{ is } A_1 \rangle$ and $\langle x_2 \text{ is } A_2 \rangle \dots$ and $\langle x_m \text{ is } A_m \rangle$ then $\langle y \text{ is } D \rangle$ ”. $A_1, A_2, \dots, A_m$ and D are fuzzy concepts or linguistic values. Usually, the development of a fuzzy system involves specifying a finite set of labels to represent the linguistic values to describe the variables. If the number of labels for the input variables $x_1, x_2, \dots, x_m$ are $\xi_1, \xi_2, \dots, \xi_m$ respectively, the number of rules declared will be $\xi_1 \times \xi_2 \times \dots \times \xi_m$. This can be referred to as the maximum or exhaustive rule set. An q -rule fuzzy system would therefore indicate a system with q being less than or equal to $\xi_1 \times \xi_2 \times \dots \times \xi_m$. This we refer to as an q -rule constrained fuzzy system, or simply an q -rule fuzzy system [7].

Alternatively, the rules can also be expressed as a linguistic matrix as described by Kosko [74]. As a general illustration, Table 3.3 shows the linguistic matrix of the maximum rule set for a system of 2 input variables x_1 and x_2 with the linguistic term sets $\{A_1, \dots, A_m\}$ and $\{B_1, B_2, \dots, B_n\}$ respectively. Each entry $\alpha(A_i, B_j)$

in the cells of the matrix represent the linguistic values of the conclusion part of a rule. For example, the cell $\alpha(A_2, B_4)$ can be rewritten as “if $< x_1$ is $A_2 >$ and $< x_2$ is $B_4 >$ ” then $< y$ is $\alpha(A_2, B_4) >$ ” where $\alpha(A_2, B_4)$ is a linguistic value of the variable y .

3.3 Genetic Coding of a Fuzzy System

Let p denote the size of the maximum rule set and $S_{m/p}$ the genetic structure describing an m -rule fuzzy system. $S_{m/p}$ is a string that is usually referred to as a chromosome, which can be written as

$$S_{m/p} = (g_1, g_2, \dots, g_p) \quad (3.3.1)$$

The symbol g_i is an identifier corresponding to the conclusion of the rules. For example in Table 3.3, $g_1, g_2, \dots, g_p$ correspond to $\alpha(A_1, B_1), \alpha(A_1, B_2), \dots, \alpha(A_w, B_v)$ respectively. In genetic terms, each g_i is known as a gene and its value, an allele. For an output variable y with linguistic term set $\{D_1, D_2, \dots, D_w\}$, a typical coding would have $g_i \in \{0, 1, 2, \dots, w\}$, where 0 is used to represent a NULL entry while $1, 2, \dots, w$ represent $D_1, D_2, \dots, D_w$ respectively. A string $S_{m/p}$ of an m -rule fuzzy system will therefore consist of $(p - m)$ NULL entries. The search space for the GA is then ${}^pC_m \times n^m$. If $p=25$, $m=11$ and $w=5$, the search space is approximately 2.18×10^{14} .

Table 3.3: Linguistic matrix of the maximum rule set for a system with 2 input variables

		x_2			
		B_1	B_2	...	B_v
x_1	A_1	$\alpha(A_1, B_1)$	$\alpha(A_1, B_2)$	...	$\alpha(A_1, B_v)$
	A_2	$\alpha(A_2, B_1)$	$\alpha(A_2, B_2)$	...	$\alpha(A_2, B_v)$
	$\cdot$	$\cdot$	$\cdot$	$\cdot$	$\cdot$
	A_w	$\alpha(A_w, B_1)$	$\alpha(A_w, B_2)$	...	$\alpha(A_w, B_v)$

An example of a fuzzy rule set for a control system in linguistic matrix format [74] is presented in Table 3.4. As an illustration of the coding mechanism, the corresponding genetic code of the 11-rule system is “0010000230123450340000500”. From this illustration, we can see that 0 is representative of a NULL entry. Similarly, 1,2,3,4 and 5 are representative of the labels NM, NS, Z, PS and PM respectively.

Table 3.4: Example of fuzzy rule set in a control system

		θ				
		NM	NS	Z	PS	PM
ω	NM			NM	Z	
	NS			NS		
	Z	NM	NS	Z	PS	PM
	PS	$\cdot$	Z	PS		
	PM	$\cdot$	$\cdot$	PM		

Alternatively, we can write $S_{m/p}$ as a string of (N, V) pairs, where N uniquely identifies the rule in the maximum rule set space and V denotes the conclusion of the rule. For example, the rule set in Table 3.4 may be coded as (2,1) (7,2) (8,3) (10,1) (11,2) (12,3) (13,4) (14,5) (16,3) (17,4) (22,5).

From the viewpoint of implementing the GA crossover and mutation operators, the first representation will be much easier than the second one. Assume that we choose

the second representation, i.e. the (N, V) pairs, to carry out mutation, we should first select randomly the gene to be mutated. Secondly, we should randomly mutate N in the gene and then V respectively. These steps are not required if we adopt the first representation.

Nevertheless, the advantage of using the second representation is that we are able to carry out crossover and mutation while at the same time, ensuring that we satisfy the constraints of the solution structure. Subsequently, we can convert the format of the chromosome into the first format. From a hardware point of view, it is still a little complex. The indirect representation approach will incur additional hardware or firmware complexity. In order to achieve higher throughput in system evolution, we prefer the first representation to the second one. In the next chapter, we offer some descriptions on how to make a fuzzy system evolve intrinsically and achieve online adaptation based on the above considerations and some novel approaches.

Chapter 4

Intrinsic Evolvable Fuzzy Hardware System

EHW can be classified into *extrinsic* and *intrinsic* EHW based on the scheme of evolution used. Extrinsic EHW relies on a simulated evolutionary process independent of the hardware. Intrinsic evolvability means that the evaluation of solutions is carried out at the hardware level of the EHW system. Currently, there has been great progress made for extrinsic type of EHW [8–12, 75]. There are also research works that focused on intrinsic EHW. Most of the works on intrinsic EHW up to now can be found in [16–18]. This type of EHW offers some advantages over extrinsic EHW. Since the evaluation is carried out in hardware, the evaluation process is very fast, and the performance of the elite is not affected by errors in the simulation model. Intrinsic EHW is useful for applications that require on-line and real-time system reconfiguration. However, the implementation of intrinsic EHW still poses

significant challenges for such promising areas.

For the implementation of intrinsic evolvable and on-line adaptive EHW, there are three main open issues that need to be addressed [14]. They are *on-line adaptation*, *scalability*, as well as *termination of evolution*. In order to perform on-line adaptive and intrinsic evolvable hardware, we propose a new form of EHW that is referred to as *evolvable fuzzy hardware* (EFH) which can address the three open issues very elegantly. EFH can be viewed as a form of *evolvable fuzzy system* (EFS) whereby the fuzzy inference system is implemented in hardware to deliver real-time inference throughput. Furthermore, the domain knowledge of the fuzzy system should be able to support online real-time reconfiguration. EFH can overcome the disadvantages of the other three EHWs described earlier and is amenable to intrinsic evolution and online adaptation. ATM cell scheduling will be an application problem to demonstrate the applicability of EFH. EFH searches for an appropriate fuzzy rule set to carry out the scheduling task on dynamically changing cell flows. The evolutionary search process carries out parallelly with the system operation and does not cause any interruption in the system operation. After a good fuzzy rule set is found, the old one is replaced immediately. The reconfiguration capability of EFH can be supported by a *reconfigurable fuzzy inference chip* (RFIC), forming the core processing unit of the EFH system. Its context can be changed or reconfigured on-line [31] without significant overhead. The reconfiguration can be achieved by a simple load operation of the context register.

In Section 4.1, we describe specifically how the implementation challenges of the

intrinsic EFH are addressed. In Section 4.2, we describe the evolution schemes and the formulation of the fitness function adopted in our EFH in details.

4.1 Solutions to open issues of EHW

In order to carry out the packet scheduling task mentioned in Chapter 2, we design the system architecture, incorporating evolutionary mechanisms as in Figure 4.1. In this system, the training buffers TB_1 and TB_2 are used to store $class_1$ and $class_2$ cells respectively. When either TB_1 or TB_2 is full, the evolutionary process is triggered. Fitness evaluation is carried out by subjecting each chromosome to the *scheduling model* according to the cell flow stored in TB_1 and TB_2 . The purpose of the *scheduling model* is to emulate the function of the multiplexer as in Figure 2.1.

After a specified number of cycles and generations, if a chromosome that corresponds to a system rule set is better than the working chromosome, the working chromosome is replaced immediately. In order to prevent the search procedure from being trapped in a local region, after a pre-specified number of generations, the whole evolutionary process is restarted, from the point where the initial population is generated. This is essentially the start of a new *evolution cycle*. Functionally, the *scheduling model* emulates the packet switching to derive the cell delay and cell loss parameters. This is achieved through a multiplexer emulation within the *scheduling model* block. The derived parameters enable the fitness value to be calculated using the fitness function. Basically, the *evolution module* evolves the appropriate rule

set by interacting with the *scheduling model* to evaluate the fitness of each evolved fuzzy rule set. When evolution is triggered, it works in the background while the MP unit is in operation. With EFH, the fuzzy inference circuit is a very important component and it directly affects the speed of the system's response to the changes in cell flow. Two high-speed fuzzy inference components are required. One is in the *scheduling model* and another is the RFIC block performing cell scheduling control. During evolution, it is inevitable that poor quality chromosomes i.e., chromosomes that result in poor switching performance, are also evaluated. To avoid the possibility of detrimental effects on the system performance by these chromosomes, the *scheduling model* is incorporated in Figure 4.1 to emulate the cell scheduling process.

Figure 4.1: Adaptation framework for EFH

This allows for evaluation of the evolved chromosomes in the background. After the evolution process, only the final fuzzy rule set will be configured in the RFIC block. In this way, we address the first major open issue of the intrinsic EHW.

In order to achieve online adaptation and intrinsic evolution for real-time control,

another issue that can be regarded as a sub-problem of online adaptation and intrinsic evolution, must also be addressed. During evolution, training data are required. In [8], the EHW system uses the same data for training and testing. This scheme can work well in applications where the real time data do not change dramatically. But if the application scenario is significantly different from the training situation, the system may not perform very well. This indicates that extensive data samples are necessary for such an evolution scheme. If the real-time data change dramatically, it is not practical to incorporate diversely representative real data samples to train the system. For many real-time control areas, we believe that there is no need to do so. In fact, we can apply the principle of “locality” to substantiate this belief. For example, in computer operating system, the design of the cache memory system is based on this principle. Accordingly in computer operating system, if a program is accessing a certain part of the memory, then there is a great likelihood that the program will also access the part of the memory within the same locality in the next time period. In our EFH, we contend that there is a very high probability that the data model within a small time window is the same as the model of data samples in the previous time window. The locality proposition is valid if we assume that the time window is small enough. For the CBR flow as described in Fig. 2.2, since the cell rate is constant [54, 76], the cell rate at any particular time period is the same as that of the preceding time period. For VBR flow, which can be described by a *two-phase burst/silence* model as described in Fig. 2.3, cells can be sent equidistantly during the burst period and no cells are transmitted during the silence period. The cell rate during the burst period can be approximated based

on the principle of “locality”. But at the edge of the burst period and the silence period or vice versa, significant error may occur. This kind of prediction error can be tolerated if the time window is sufficiently small. Based on this justification, we can train the system using the previous data flow to approximate the expected data model of the subsequent time period. The smaller the time window, the more flexible the EFH adapts to the cell flow. The best chromosome after an evolution process will be used to do scheduling in the next time period.

To address the scalability issue, we adopt an evolutionary granularity at the fuzzy rule level. In the EFH for packet switching, a chromosome can be represented as a string of 25 integers. Each gene of a chromosome represents a fuzzy rule. For this scheme, the search space is not too big compared to the search space in [8,17], in which each chromosome is represented by a string comprising of hundreds of integers or thousands of bits. The evolution time in the EFH is thus reasonable. The third issue to address is the termination of evolution. In many EHW systems, the evolution system may require thousands of generations to get close to an optimal chromosome. The extent of evolution time may limit the applicability of the system for real-time application. In [8], in order to get a good functional EHW to do ATM cell scheduling, the system evolved for 2500 generations with a population size of 400. In [17], in order to derive a circuit with Gaussian output voltage characteristic, the evolvable hardware system has to evolve for 10000 generations. The time scale for evolution in these reported works is not appropriate if used in real-time intrinsic EHW control system. For comparison, in the proposed EFH, a very small population

size and small number of the generations are important features of the evolutionary process. In order to prevent the system from adopting a very poor performing fuzzy rule set, we defined a core rule set in the system derived based on the analysis of the problem through human intuition. The core rule set is also used as the startup rule set. If the EFH system is not able to find a chromosome that is better than the core rule set within a fixed number of generations, the core rule set is adopted. The appropriate number of generations for each evolutionary cycle is determined through experimentation. The objective of the evolution is to get a fuzzy rule set better than the working chromosome for the cell flow of the following time period. Even if the derived fuzzy rule set is not optimal, it is deemed to be sufficient. By adopting this idea, the criterion for the termination of evolution can be satisfactorily managed.

4.2 Evolution scheme

To carry out evolution, GA manipulates a population of chromosomes. These chromosomes are solution representations denoting the application domain fuzzy rule sets, when decoded. In this section, we will first introduce the fuzzy system and its coding scheme. Then we will describe the inference scheme and the fitness function of this system.

4.2.1 Genetic coding

A fuzzy system can be formally defined as an application or system, which employs a fuzzy control algorithm. In general, the fuzzy control algorithm refers to a set of *if-then* rules with linguistic values and fuzzy variables. The values are specified as fuzzy concepts defined by membership functions. Fuzzy system implicitly means a set of rules and membership functions.

To begin with, we define two symbols for the inputs, c_1 and c_2 . The symbol c_1 refers to the status of $class_1$ cell flow, which is a function of V_1 and V_{max} . V_1 is the current cell rate of $class_1$ cell flow while V_{max} is the line capacity. The symbol c_2 refers to the buffer status of BUF_2 . It is a function of L_2 and L_{max} . L_2 is the number of empty units in BUF_2 while L_{max} is the length of BUF_2 . For c_1 and c_2 , the memberships are characterized by the terms set $\{VS, S, M, L, VL\}$ as depicted in Figure 4.2. These are standard triangular membership functions. The output SEL of the *fuzzy switching control* block (see Figure 4.1) is characterized by the term set $\{T, F\}$. Both T and F are singletons, or fuzzy sets with impulse membership functions as shown in Figure 4.3. Functionally, a T or *true* means that the MP unit allocates time slots to cater for the $class_1$ cell flow in BUF_1 . An output F or *false* implies that switching is reverted to cells in BUF_2 .

Based on the above characterization of the switching network, it is possible to define the n -rule heuristics to control the switching behavior. With the fuzzy memberships defined, one can rely on intuitive logic to define the necessary input-output mappings

Figure 4.2: Membership functions for c_1 and c_2

Figure 4.3: Membership functions for T and F

as shown in Table 4.1. The 25-rule system serves as the default cell scheduling algorithm on system startup. We refer to this rule set as the core rule set.

A fuzzy rule set can be represented as a string of integers. For example, the genetic code for the 25-rule system in Table 4.1 can be described by the string "222112211122111211111111". The allelic code 1 and 2 correspond to the labels *true* (T) and *false* (F) respectively. The position of the gene in the string identifies a specific rule in Table 4.1 when interpreted accordingly in a row wise manner. If the value of a gene is 0, it means that there is no specific fuzzy rule defined for the corresponding input condition. The core rule set not only serves as the startup rule set, but also provides a means to benchmark the performance during the evolution of chromosomes. This scheme guarantees that the performance of the system is better than or at least comparable to that of the core rule set.

Table 4.1: A 25-rule fuzzy system for ATM cell scheduling

Fuzzy Variables		c_1				
		VS	S	M	L	VL
c_2	VS	F	F	F	T	T
	S	F	F	T	T	T
	M	F	F	T	T	T
	L	F	T	T	T	T
	VL	T	T	T	T	T

4.2.2 Inference scheme

Each entry in Table 4.1 can be interpreted as a statement of the form "if $antecedent_1$ and $antecedent_2$ then $conclusion$ ". The $antecedent_{\#}$ represents the fuzzy conditions for c_1 or c_2 , characterized over the term set $\{VS, S, M, L, VL\}$. The $conclusion$ can be T or F . The degree of firing for each fuzzy rule is taken as the minimum of the degrees of matching between the inputs c_1 and c_2 and the antecedents. The aggregation is carried out by averaging the fuzzy conclusions derived from all the rules.

Although we have shown a 25-rule system for this evolvable system, the number of the fuzzy rules in our system can vary between 0 and 25. If it is needed to further improve the evolution time and reduce the search space, the size of the rule set can be fixed to be less than 25 as in [71], so that the evolution time can be managed. This is because the search space for a reduced rule set is more manageable and hence the evolution efficiency can be significantly improved. Suppose the rule set size is fixed to be 11, the search space can be $^{11}C_{25}$. The bigger the search space, the more difficult to find a rule set which is close to the optimum.

4.2.3 Fitness function

According to the specifications of the problem, the capacity of the output channel is fixed. This implies that no further adjustment on the output capacity can be made to cater for fluctuations in demand. If the bandwidth is not big enough to meet the demand of the two cell flows, servicing *class*₁ cell would mean filling up the *class*₂ buffer and eventually resulting in cell loss for *class*₂. Hence for a specified requirement on the level of cell delay for *class*₁, a certain expected level of *class*₂ cell loss is inevitable. In other words, the *class*₂ cell loss is constrained by the desired level of *class*₁ cell delay that the system is trying to achieve.

There is one main consideration in formulating the fitness function for the EFH. This pertains to the *class*₁ average cell delay. From the above discussion, it is apparent that the level of *class*₂ cell loss is negatively correlated to the average *class*₁ cell delay. Adjusting *class*₁ cell delay will adversely affect the *class*₂ cell loss. Based on these justifications, the fitness function can be described explicitly as in Eq.4.2.1.

$$F = \kappa - |AveDelay - (\lambda \times DelayFactor)| \quad (4.2.1)$$

In Eq.4.2.1, κ is a very large numerical constant. It is used to adjust the range of fitness values such that F is proportional to the fitness measure of the chromosome. The larger the fitness value, the fitter the chromosome. *AveDelay* is the average delay of *class*₁ cell units after all the cells in TB₁ have been processed. *DelayFactor* is a constant used as a reference for scaling the value of λ based on

the desired $class_1$ cell delay. λ is an adjustable coefficient to denote the desired level of average cell delay for $class_1$ cell units stored in TB_1 . In general, the system tries to search for a chromosome with minimum $|AveDelay - \lambda \times DelayFactor|$. Both the $AveDelay$ and $DelayFactor$ in Eq.4.2.1 can be determined from Eq.4.2.2 and Eq.4.2.3 respectively.

$$AveDelay = \frac{1}{\tau} \times \sum_{i=1}^{\tau} m(i) \quad (4.2.2)$$

$$DelayFactor = \rho \times v \quad (4.2.3)$$

In Eq.4.2.2, $m(i)$ is the waiting time of the i th cell in TB_1 before being sent out. τ is a variable denoting the number of $class_1$ cell units in TB_1 sent during evaluation. $\sum_{i=1}^{\tau} m(i)$ is the sum of the cell delay of the cell units in TB_1 . In Eq.4.2.3, ρ is a constant corresponding to the time required to send a cell through the output channel. The value of ρ depends on the bandwidth capacity of the output channel. The symbol v denotes the size of $TB_{\#}$. With Eq.4.2.3, a reference value for the possible delay of $class_1$ cell units can be determined.

4.2.4 Reconfigurable Fuzzy Inference Chip

According to the evolution scheme described by Figure 4.1 in Section 4.2, the chromosomes need to be evaluated within a very short time period for each evolutionary

cycle. If the whole evolution process can be completed within the time it takes to send one cell packet through the OUT channel, and a good fuzzy rule set can be found during this time period, the system will enjoy the greatest flexibility in adapting to the changing environment. On the whole, the performance of the system is very much dictated by the quality of the rule set being applied. Each rule set instance is referred to as a context, and is applicable to the current scenario of the operating environment. As context changes, the fuzzy inference circuit is required to accommodate the new context without incurring significant overhead for setup. This implies that a reconfigurable high-speed fuzzy inference circuit is very critical in EFH. In order to achieve fast fuzzy inference and at the same time accommodate real-time online context updating, a hardware scheme for fuzzy inference called *reconfigurable fuzzy inference chip* (RFIC) has been proposed. The detailed information of RFIC can refer to [31].

The novelty of the RFIC lies in its ability to accommodate an online context change without interrupting the system operation. The block architecture of RFIC is as shown in Figure 4.4. The main component is the FIM (*fuzzy inference map*) block. It adopts an implicit inference approach to deliver high inference speed for applications with dynamically changing contexts. The current applicable context is managed by the CMU (*context management unit*). It stores the working fuzzy context and generates control signals such as $\text{Ena}_{\langle x,y \rangle}$ and $\text{Sel}_{\langle x,y \rangle}$ for the FIM. AEM (*address encoding mechanism*) is the module that generates the address to access the FIM partition blocks activated by the $\text{Ena}_{\langle x,y \rangle}$ signals (see Fig. 4.4). The OAM (*output*

aggregation mechanism) is the dedicated circuit for fuzzy inference aggregation.

The proposed EFH system for cell scheduling is able to accommodate fuzzy rule sets of up to 25 fuzzy rules. Hence, the FIM block incorporates 25 PBs (*partition blocks*); $PB_{<1,1>}$, $PB_{<1,2>}$ ... $PB_{<5,5>}$. Each PB is a mapping that accommodates all the input situations with specific outputs. The mapping for each PB is created based on a software fuzzy inference model. To illustrate the basic structure and format of each PB, we can assume that the inputs and the membership functions are digitized to 5 bits. A sample of the mapping data for $PB_{<1,1>}$ is presented in Table 4.2 for illustration. The left column of the table lists the addresses. The whole address string is composed of three parts, i.e., the digitized values of $Input_1$, $Input_2$ and $Sel_{<1,1>}$. The data are made up of two parts. The most significant bit is the fuzzy conclusion bit indicating T or F . The other bits represent the degree of firing for the corresponding fuzzy rule. For example, referring to the first memory unit in Table 4.2, where both $Input_1$ and $Input_2$ equal to “00000”, the degree of matching to the membership function VS is “11111”. So the corresponding datum in the location is “0,11111”. Its first bit “0” represents the fuzzy conclusion T and the other bits “11111” is the firing strength.

CMU stores the current application context and generates $Ena_{<x,y>}$ and $Sel_{<x,y>}$ signals. For the application described, the size of the context register required is 50 bits. Each two-bit datum in the register represents a fuzzy rule. The position of each two-bit datum in the 50-bit string identifies the specific rule of the context. A “01” means the fuzzy conclusion is T and “10” indicates the fuzzy conclusion is

F . A “00” means that there is no fuzzy rule for the corresponding input situation. Each $\text{Ena}_{\langle x,y \rangle}$ signal can be generated by applying the logical *OR* operation to the corresponding two bits. A value of “1” for $\text{Ena}_{\langle x,y \rangle}$ indicates that $\text{PB}_{\langle x,y \rangle}$ is enabled, which otherwise is disabled. $\text{Sel}_{\langle x,y \rangle}$ also depends on the specific two bits and is connected to $\text{PB}_{\langle x,y \rangle}$ separately. A “01” generates a “0” for $\text{Sel}_{\langle x,y \rangle}$ and “10” produces a “1”.

OAM is an aggregation circuit. In [31], Ave_2 and Ave_3 blocks are adopted. For the case of the application in this thesis and for the purpose of simplification of circuit implementation, a Sum/Sub block is adopted. It can refer to Fig.8.14 in page 126.

Table 4.2: FIM content in $\text{PB}_{\langle 1,1 \rangle}$

Address	Data
00000,00000,0	0,11111
00000,00000,1	1,11111
00000,00001,0	0,11011
00000,00001,1	1,11011
00000,00010,0	0,10111
00000,00010,1	1,10111
.	.
.	.
.	.
00001,00000,0	0,11011
00001,00000,1	1,11011
.	.
.	.
.	.
00111,00111,0	0,00111
00111,00111,1	1,00111

Figure 4.4: Block architecture of RFIC

Chapter 5

Simulation Scheme

5.1 Simulation Architecture

There are many academic network simulation tools available, a famous one is NS-2 [77]. NS-2 is a very powerful tool, but it is not very suitable for this research. This simulator is designed for the purpose of simulation of a whole network. In this research, only one type of network component – Multiplexer is considered. This component is not very complicated from the point of view of simulation. The key point of this research is to implement and simulate the evolution block which is far more complicated and time consuming than the multiplexer. NS-2 has a Tcl programming interface. Tcl is a Parsing and Execution based programming language. Execution speed will be a disadvantage if Tcl is used to simulate GA functions. From the consideration of the above two reasons, the EFH simulation is implemented in

C++ which includes GA operations, fuzzy system's computation and the function of the multiplexer. Some traditional scheduling schemes such as FIFO and DWPS schemes are also simulated using C++ program for performance comparison. For the validation of the simulator, short data flow is used as input and manual calculation is performed for comparison. For the validation of EFH system, two steps are involved. First, different channels are given the same setup and same input data flows, the calculated cell delays by the program for the input channels should be similar. The difference should be within several microseconds. Second, different flows are given to different channels. The inputs for channel one, channel two, and so on can be shifted in counter-clockwise. The delay calculated for the same data flows should be very close even if it is input to different channels. Again, the difference should be within several microseconds. Since the manual calculation of the traditional scheduling scheme is not very complex, for the validation of the traditional scheme, short data flow is used as input and manual calculation is performed for comparison. The two results should be the same.

The simulation architecture is as Fig. 5.1. Timer is the simulation timing unit. FlowReader is to read in the flow data from several data flow files generated by another program. It reads the flow file and outputs the cell unit to $class_j$ if the cell's arrival time is equal or bigger than the SIMULATION_TIME of the Timer unit. BUF_#, Mux and ControlUnit make up the components necessary for normal scheduling. BUF_# buffers the cell units as the congestion occurs. Mux can schedule out a cell in the head unit of BUF_# under the control of the ControlUnit. The

Mux's output CELL is the cell unit scheduled out. DelayCalculator calculates the corresponding cell delay for each flow according to the simulation time. LossCounter is counting block for the number of cell loss for each flow. ControlUnit implements different scheduling schemes such as FIFO, MCRR, fuzzy logic and so on so forth. Evolution block is an optional block. It only exists for EFH. It performs the GA operations and chromosome's evaluation. EFH block outputs the searched final chromosome as ELITE to the ControlUnit. After each evolution, if a better fuzzy rule set is found, the fuzzy rule set in the ControlUnit will be replaced by the searched elite to control the cell scheduling of the following cell flows.

Figure 5.1: Simulation architecture

5.2 Cell Flow Generation

In this thesis, CBR and VBR flows are mainly considered. These two cell flows can be described by Fig.5.2 and Fig.5.3. CBR always has cell units with constant internal gap. VBR also has cell units with constant internal gap during the ON period. It can be simulated by the Markov chain model as described in [78]. The burst length is modelled as a geometrically distributed random variable. The characteristic of the cell units in CBR and VBR can be described by the Fig.5.4. c_i in the figure represents the cell or packet. The variable l is the packet length which is fixed in the ATM network. The packet gap n can be a constant value. In this thesis, n is set to be 1 to 10 bits.

Figure 5.2: Description of CBR

Figure 5.3: Description of VBR

Figure 5.4: Cell Flow Characteristic

The normal cell format for a network includes *UNI header*, *NNI header* and data

frame. In this thesis, the packet doesn't go through any other network components. For the purpose of simulation simplification, a different cell format which is convenient for scheduling simulation and delay calculation is adopted to simulate the cell flow. It is presented in Fig. 5.5.

Cell#	Arrival_time	Bit_rate

Figure 5.5: Data format

Cell# refers to the cell number. *Arrival_time* indicates the generation time of the cell unit. *Bit_rate* refers to the current bit rate. This cell format is mainly used for simplifying simulation process. In the true ATM application, the current bit rate can be detected by a specialized circuit. The overall flow diagram of the system simulator for EFH is as shown in Fig.5.6. The simulation flows for the normal schemes are also very similar to Fig.5.6 except that the parallel running evolution block doesn't exist.

T_{a1} , T_{a2} and T_{ai} in Fig.5.6 refer to the cell arrival time for *class*₁, *class*₂ and *class*_{*i*} respectively while t is the current running time of the simulation. $TB_{\#}$ is a buffer collecting the information of the coming cell units. When one of the TB buffers is full, the Evolution block in the dash frame is triggered. It runs parallelly with the main scheduling flow. If no better rule set found after an evolution cycle, the working fuzzy rule set continues controlling the scheduling operation without any changes. If a better one is found, the fuzzy rule set which is in the scheduling block will be replaced by a register operation.

Figure 5.6: Simulation flow diagram

5.3 Simulation Cases

For the demonstration of EFH's capability of controlling the cell scheduling, the multiplexer with two input flows is considered first. And then the application is expanded to multiple input channels where three input channels are mainly considered. In these two different simulation cases, different fitness functions are adopted for different system performance requirements.

5.3.1 Two-Channel Simulation

For the purpose of demonstrating the viability of the EFH scheme, in these simulation cases, we carried out simulations of EFH in cell scheduling on two different scenarios, scenario_1 and scenario_2 . In both scenarios, we assume the capacity of the output channel (OUT) and the input channels to be 155.52M bits per second. These two scenarios are quite different cell flow scenarios. They will be described in Chapter 6.

The simulation results of EFH are compared with that of *first-in first-out* (FIFO) and *dynamically weighted priority scheduling* (DWPS) [57] in terms of the class_1 cell delay and the balance of cell loss for class_1 and class_2 . After this, the special characteristic of EFH's tunability is demonstrated. FIFO is a very traditional scheduling method. It schedules the cell flow based on the arrival time of the packets. FIFO can achieve very good balance between class_1 cell loss and class_2 loss, but it is very bad in terms of class_1 cell delay performance. DWPS is a very good algorithm for

cell scheduling. It adjusts the priority according to the cell flow scenarios. But the adaptation scheme of DWPS is not very efficient if the cell flow changes dramatically. DWPS can be described by Eq.5.3.1. In Eq.5.3.1, v_i is the fixed priority for different cell flow inputs, a lower value indicates a higher priority. $T_i(t)$ is the waiting time of the oldest packet in the buffer of the i th channel. Q_i is the priority index associated with each cell. The lower the value, the higher the priority. γ is an emphasis parameter and the recommended value is 0.9.

$$Q_i = \frac{v_i}{[T_i(t)]^\gamma} \quad (5.3.1)$$

5.3.2 Multi-Channel Simulation

After the simulation on two input channels, EFH is further expanded to control multiple input channels. In this case, three input channels are considered. In this case, the ability of suppressing contract breaking flows and protecting the contract following flows from being affected are tested. From this point of view, *generalized processor sharing* (GPS) is the ideal algorithm. Due to its principle, it cannot be implemented. But GPS can be used as a benchmark on how good a scheduling system is. EFH is simulated and compared with GPS, FIFO and *minimized cycle round robin* (MCRR). The tunability of EFH is also demonstrated in this case.

In both of the two-channel simulation and multiple-channel simulation, the evolution time of $20\mu\text{S}$ is considered. This value is from the computation complexity analysis

presented in Section 7.4.1.

After the two simulation cases, for the purpose of completeness, the hardware implementation of the EFH for the application of two input channels is described and the circuit is simulated and compared with the results of C++ simulation. A short part of the data flow lasting for 100 mS is given to the C++ program and the hardware representation. The purpose of presenting hardware implementation and simulation is to demonstrate that the whole EFH can be implemented and can work, comparable to the C++ simulator.

Chapter 6

Two-Channel Simulation

Two scenarios are considered in this simulation. For *scenario*₁, *class*₁ is the CBR cell flow with cell bit rate of 155.52M bits per second. *class*₂ is VBR cell flow, also with a cell bit rate of 155.52M bits per second. The difference is that the VBR specified has a 2ms ON time period and a 2ms OFF time period. This scenario is a very extreme case used to test the system's controllability. In order to simulate the system performance on a more realistic cell flow, we can adopt *scenario*₂. For *scenario*₂, *class*₁ refers to CBR cell flow with a cell bit rate of 100M bits per second. *class*₂ is VBR cell flow with unknown random cell bit rate. The minimum cell bit rate for VBR is 55.52M bits per second while the maximum is 155.52M bits per second. In these two scenarios, since the sum of the CBR and VBR cell rate is larger than the OUT channel's capacity, cell loss is unavoidable. From a practical point of view, the second scenario is more likely compared to the first scenario.

6.1 Comparison

For the simulation, the size of BUF_1 and BUF_2 is 100 cells, and the size of TB_1 and TB_2 is 300 cells. In the fitness function, λ is 0.35. All the simulations are carried out by using a C++ program. The setting for the parameters of the evolutionary algorithm is derived based on experimentation. The simulation results presented later show that the parameters setting below can achieve good system performance.

- population size=10;
- elite pool size=2;
- crossover probability=0.6;
- mutation probability=0.05;
- number of generation=9;
- number of evolutionary cycle=2.

We simulated each scheduling scheme for cell flows lasting 2 seconds. Fig. 6.1, 6.2 and 6.3, 6.4 are the simulation results of FIFO, DWPS and EFH schemes on *scenario*₁. Fig. 6.5, 6.6 and 6.7, 6.8 are simulation results for FIFO, DWPS and EFH schemes on *scenario*₂. The simulation results demonstrate the viability of the evolution scheme and that EFH can fulfill the cell scheduling task. For *scenario*₁, EFH can achieve lower *class*₁ cell delay than FIFO and DWPS. The balance of *class*₁ and *class*₂ cell loss by using these three methods is acceptable. None of the schemes show significant bias towards any of the two cell flows. For *scenario*₂, the situation is quite different. EFH can still achieve lower *class*₁ cell delay with

an acceptable balance between $class_1$ cell loss and $class_2$ cell loss. The $class_1$ cell delay by using DWPS is higher than that of EFH and the balance between the $class_1$ cell loss and the $class_2$ cell loss is not good. So according to the quality factors as discussed above, EFH can control the cell scheduling better than FIFO and DWPS when the cell flow changes dramatically.

Figure 6.1: Cell delay for $class_1$ in $scenario_1$

Figure 6.2: Cell delay for *class₂* in *scenario₁*

Figure 6.3: Cell loss for $class_1$ in $scenario_1$

Figure 6.4: Cell loss for *class₂* in *scenario₁*

Figure 6.5: Cell delay for *class₁* in *scenario₂*

Figure 6.6: Cell delay for *class₂* in *scenario₂*

Figure 6.7: Cell loss for *class₁* in *scenario₂*

Figure 6.8: Cell loss for *class₂* in *scenario₂*

6.2 Tunability of EFH

One advantageous property of EFH is that the system performance can be adjusted very intuitively by decreasing or increasing the value of λ in Eq.4.2.1 of Chapter 4.

The smaller the value, the smaller the $class_1$ cell delay. This property cannot be conveniently achieved using traditional scheduling methods. As in the above, the tunability of EFH is demonstrated by simulation results on $scenario_1$ and $scenario_2$.

The results of simulation when λ is progressively increased are as shown in Fig. 6.9, 6.10, 6.11 and 6.12. In Fig. 6.9 and 6.10, when λ is smaller than 0.3, the $class_1$ cell delay and cell loss of $class_1$ and $class_2$ do not change. We note that the average cell delay is not zero. This can be logically explained by taking into account the initial starting rule set for the evolutionary system. The starting rule set is essentially a manually defined core rule set. Characteristically, the performance of the core rule set is neutral to $class_1$ and $class_2$ in terms of average cell delay. When the evolution process is triggered initially, the inherent cell delay is characteristic of the performance for the initial core rule set adopted. From Fig. 6.9 and 6.10, it is observed that the level of $class_1$ cell delay, $class_1$ cell loss and $class_2$ cell loss can be tuned by adjusting the parameter λ . For smaller values of λ , a greater emphasis is placed on $class_1$ delay. For larger values of λ , $class_2$ cell loss is accorded greater importance. As λ increases, $class_2$ cell loss performance becomes more important than $class_1$ cell delay. In Fig. 6.11 and 6.12, the same conclusion can also be drawn. The major difference between Fig. 6.9 and 6.10 and Fig. 6.11 and 6.12 is that when λ is zero, the average $class_1$ cell delay in Fig. 6.11 is close to zero. The reason

for this difference is that in *scenario₂*, the cell bit rate of *class₂* is not large at the beginning. Before the start of the first evolution, only a small number of *class₂* cells arrived and some of them are serviced accordingly. A major portion of the time slots is given to *class₁* cell flow. This explains why the *class₁* cell delay before the start of the first evolution is not large.

In Fig. 6.9 and 6.11, when λ is 0, *class₁* cell delay is very small. It is the minimum *class₁* cell delay that the EFH can achieve. According to earlier justifications, *class₂* cell loss in Fig. 6.10 and 6.12 when λ is 0 is very close to the maximum value of the sum of cell loss for specific scenario. Based on Fig. 6.9 and 6.11, it can be seen that the *class₁* cell delay has a very wide range of tunability. It can also be seen in Fig. 6.10 and 6.12 that *class₁* cell loss and *class₂* cell loss can be tuned to a wide range. These aspects of performance tuning cannot be easily realized by static and non-adaptive scheduling mechanisms.

Figure 6.9: Class₁ cell delay for Scenario₁

Figure 6.10: Cell loss for Scenario₁

Figure 6.11: Class₁ cell delay for Scenario₂

Figure 6.12: Cell loss for Scenario₂

6.3 Simulation on Realistic Traffic Scenario

In the previous section, we presented the simulation results and the tunability of the EFH on two traffic scenarios. To further show that the EFH has considerable controllability and tunability, we consider another traffic scenario. Realistic traffic flow may not necessarily be the same as *scenario₁* and *scenario₂* according to the *connection admission control* mechanism of ATM. This means that a connection can only be accepted if there are sufficient bandwidth resources [79]. *scenario₁* and *scenario₂* adopted in Section 6.1 can be used to test the system's controllability and flexibility. Whether or not the EFH is efficient for practical applications can be further verified by simulations on some more realistic traffic patterns. One such traffic pattern is as shown in Fig. 6.13.

In Fig. 6.13, *class₁* refers to constant bit rate flow with cell bit rate of 80M bits

Figure 6.13: Cell flow for scenario₃

per second that is sensitive to delay, while $class_2$ is a bursty traffic flow that is sensitive to cell loss. Assuming that the connection for $class_2$ has a prescribed *peak bit rate* of (155.52-80)M bits per second or 75.52M bits per second, but due to the malfunction of some nodes in the network, the peak bit rate of $class_2$ is about 100M bits per second. For ATM system specifications, there are requirements to depress the influence of this malfunction on other normal cell flow [60]. There are many algorithms designed to do so, some of which have been mentioned in Chapter 1. These algorithms have a common shortcoming of having a fixed cell scheduling mechanism. They cannot adapt efficiently to the changes in traffic environment to deliver the desired QoS. They cannot be adjusted flexibly at the level of isolation, which means how much to keep the QoS of $class_1$ cell flow from the influence of $class_2$ cell flow. In order to realize the aspects mentioned above, the EFH can be a good choice. With its evolutionary mechanisms, the EFH can track the traffic pattern and do self-adaptation according to the change in operating environment. It can also isolate the influence of non-conforming $class_2$ cell flow on the QoS of $class_1$ traffic at any level by tuning the parameter λ . Some simulation results are presented below for $\lambda = 0.2, 0.5$ and 0.8 to illustrate this point.

From Fig. 6.14, it can be seen that setting 0.2 for λ puts more emphasis on *class*₁ cell delay than the value of 0.5. This means that setting of λ to 0.2 can isolate the influence to *class*₁ cell flow caused by the malfunction of *class*₂ cell traffic more strictly than the value of 0.5. To further depress the influence, we can assign even smaller values for λ . From Fig. 6.16 and 6.17, we can see that the balance of *class*₁ cell loss and *class*₂ cell loss for the value of 0.5 is much better than that achieved by $\lambda = 0.2$. In this case, more emphasis is placed on establishing a balance between the cell losses. If the main concern is the QoS of *class*₂, λ can be assigned a larger value, for example, 0.8. By setting λ to 0.8, the plots in Fig. 6.14, 6.15, 6.16 and 6.17 show that the QoS for *class*₂ is emphasized much more, while the QoS for *class*₁ is further de-emphasized. The tunability of the system via the parameter λ in the fitness model is consistent with the observations in the previous section. Although suffering from dynamic cell bit rates and quite different scenarios, EFH can still keep consistent QoS performance. These observations are indications that the EFH with the fitness modelling in Eq.4.2.1 of Chapter 4 can be applied in controlling realistic cell flow traffic.

Figure 6.14: Class₁ delay for Scenario₃

Figure 6.15: Class₂ delay for Scenario₃

Figure 6.16: Class₁ cell loss for Scenario₃

Figure 6.17: Class₂ cell loss for Scenario₃

Chapter 7

Multi-Channel Scheduling

For modern networks, effective management of bandwidth resource is critical because of the diversity in the services required. Bandwidth sharing is thus inevitable for efficient bandwidth utilization. Multiplexer is a network component to administer the sharing of high speed link for different input channels through *time division*. In the previous chapter, we presented EFH system to solve two-channel ATM cell scheduling problem. In this chapter, the application of EFH system on multiple input channels will be carried out. In the two-channel application, only the $class_1$ cell delay and the balance of $class_1$ cell loss and $class_2$ cell loss are mainly considered. For the multiple input scenarios, the effectiveness of the control scheme can be gauged by another means of controlling fairness. For such a situation, only the flows which override their reserved bit rates should suffer significant QoS degradation.

Each input flow should negotiate with the network system on its bit rate require-

ment before the admission of its connection. Regarding the required bit rate, the connection would be set up only if there is enough link capacity to accommodate the input flow. After the success of the admission process, the bit rate of the traffic flow cannot exceed its reserved bit rate. Based on a QoS consideration, significant QoS degradation should only be suffered by the mis-behaving flow channels. This can be called QoS isolation. This approach can effectively restrain the aggressive source domination, which could be a symptom of systematic vicious attacks on a network. The delay suffered by a cell unit in a network system includes propagation delay, queuing delay and service time at every queuing point. In this chapter, we focus on a single node switching system, thus only queuing delay is of major concern.

Various algorithms or systems have been designed to perform cell scheduling control for multiple input scenario fulfilling the QoS isolation. Examples include the *generalized processor sharing* (GPS), *weighted fair queuing* (WFQ), *worst-case fair weighted fair queuing* (WF2Q), *weighted round robin* (WRR), *minimized cycle round robin* (MCRR) as mentioned in Section 2.3 in Chapter 1.

Both the *fluid model* (see page 19) approach and the *round robin* (see page 18) approach have their own advantages and disadvantages. GPS time stamp based approach incurs high computational overhead during the queuing or de-queuing process. This can significantly affect the efficiency of the cell scheduling. Round robin approach has the lowest computational complexity but poor delay guarantee. In order to solve the cell scheduling problem, *evolvable hardware* (EHW) system can be adopted. In the previous chapters, the application of EFH in cell scheduling has

been described. In order to promote the application of EHW, we will show how the EHW model can be extended to handle multi-channel applications. The idea and the evolution scheme behind the application in this chapter are the same as that presented in the previous chapters.

The remainder of this chapter is organized as follows. In Section 7.1, the evolution scheme of the EFH for three input channels is described. In Section 7.2, the encoding scheme of the fuzzy system for *genetic algorithm* (GA) in the multi-channel application is introduced. This is followed by the introduction of the fitness function adopted in the EFH for multi-channel applications in Section 7.3. In Section 7.4, the simulation results of EFH and their comparisons with GPS, MCRR, and *first-in first-out*(FIFO) are presented.

7.1 Evolution Scheme

The block architecture of the EFH can be described by Fig.7.1. $class_{\#}$ represents the different classes of input cell flow. $BUF_{\#}$ is the queuing buffer for each input channel. It is used to temporarily store data packets when congestion occurs. The size of $BUF_{\#}$ corresponds to the size of the time window, consistent with the principle of “locality”. $TB_{\#}$ is the buffer for the storage of training patterns. It stores mainly the information of each cell, in particular the arrival time and the arriving bit rate. When one of the $TB_{\#}$ is full, the evolution process is triggered. MP is a multiplexer to send the cells in the buffers through the OUT channel under the control of an

RFIC. The value of SEL indicates the buffer from which the cells are to be sent. The *evolution module* is a component that realizes the GA search. Every chromosome generated by the *evolution module* is evaluated by the *scheduling model* block. The best chromosome after a fixed number of generations will be downloaded into the RFIC. At the beginning of the evolution, the content in $TB_{\#}$ is modified based on the arrival time of the last cell unit in $BUF_{\#}$ to prepare the training data. The architecture of the *scheduling model* is as shown in Fig.7.2. It includes RFIC', MP', $BUF_{\#}'$ and *fitness computation* blocks. RFIC' is also a *reconfigurable fuzzy inference chip*. MP' is an emulation of the MP. It simulates the functionality of the MP block by calculating the queuing delay for every cell stored in $BUF_{\#}'$. $class_{\#}'$ is the cell flow from $TB_{\#}$. The *fitness computation* block in Fig.7.2 computes the fitness value for each evaluated chromosome.

Figure 7.1: The system architecture for multi-channel scheduling

The evolution process of the EFH can be described by the procedural code listing in

Figure 7.2: The architecture of the *scheduling model*

Algorithm 1. The function of *TB_# Modification* routine is to prepare the training data for the evaluation process. This process generates packet information such as arrival time and bit rate based on the last few cells in *BUF_#*. The *Initialization of BUF_#'* routine copies the cell information from *BUF_#* in Fig. 7.1 to *BUF_#'* in Fig. 7.2. *Evolution and Evaluation* routine carries out the evolution and evaluation. The routine for the *Evaluation* module in Algorithm 1 is described further by Algorithm 2. The *Enqueuing* process adds the information of the arriving cells to *BUF_#'*. *Evaluation process* implements the fuzzy inference and cell scheduling. The *Fitness Computation Process* carries out fitness computation.

7.2 Fuzzy System

In this section, the fuzzy terms involved in the multi-channel cell scheduling application of EFH are first introduced. For the purpose of GA search in the solution space of the fuzzy rule set, genetic coding of the fuzzy rule set is necessary. In the second

---

Algorithm 1 Evolutionary process

Notations:

m : The size of $TB_{\#}$
 v : The size of $BUF_{\#}'$ and $BUF_{\#}$
 g : The number of the generation
 p : Population size for GA evolution
 $q_{\#}$: The number of cells in $BUF_{\#}'$
 c : The number of evolution cycle
 l : Chromosome length in bits
 j : Channel number

 $TB_{\#}$ Modification:

/*Construct $TB_{\#}$ based on the bit rate
when evolution is triggered*/

for $i < m$ do
    modify $TB_1(i)$;
    modify $TB_2(i)$;
    :
    modify $TB_j(i)$;
end for;

Initialization of $BUF_{\#}'$:

/*Copy cell information from $BUF_{\#}$ to $BUF_{\#}'$ */

for $i < j$ do
    copy $BUF_1(i)$ to $BUF_1(i)'$;
    copy $BUF_2(i)$ to $BUF_2(i)'$;
    :
    copy $BUF_j(i)$ to $BUF_j(i)'$;
end for;

Evolution and Evaluation:

for $i < c$ do
    for $i_1 < p$ do
        Initial population generation;
        Call Evaluation();
    end for;
    for $i_2 < g$ do
        for $i_3 < p$ do
            Toss for crossover;
            Select chromosomes for crossover;
            Do crossover operation;
            Toss for mutation;
            Do mutation operation;
            Call Evaluation();
        end for;
    end for;
end for;

Algorithm 2 Evaluation()

Enqueuing: /*Upon arrival of cells*/

    modify $\text{BUF}_1(i)'$;

$q_1 = q_1 + 1$;

    or / and

    modify $\text{BUF}_2(i)'$;

$q_2 = q_2 + 1$;

$\vdots$

    or / and

    modify $\text{BUF}_j(i)'$;

$q_j = q_j + 1$;

Evaluation: /*Simulate cell transmission*/

    while $((\exists i) q_i > 0)$ and (no cell is sending) do

        fuzzy inference;

        /*according to the fuzzy inference result*/

        Calculate queuing delay of the oldest unit in BUF_i ;

        Delete the oldest unit from BUF_i ;

    end while;

Fitness Computation:

/*Compute the fitness value*/

    Fitness Computation;

---

part of this section, the encoding scheme of the fuzzy rule set is introduced. In the following parts, three input channels ($j=3$) are assumed for simplicity of discussion.

7.2.1 Fuzzy Terms

For the fuzzy system, we define three fuzzy variables c_1 , c_2 and c_3 . $c_{\#}$ can be defined as $\frac{L_{\#}}{L_{max}}$. $L_{\#}$ represents the number of empty units in $BUF_{\#}$. L_{max} is the length of the buffers. $c_{\#}$ can be characterized by the fuzzy terms $\{S, M, L\}$. The membership functions for these fuzzy terms are chosen to be triangular as in Fig.7.3.

Figure 7.3: The membership functions for $c_{\#}$

The output of the fuzzy system can be described by the terms set $\{0, 1, 2, 3\}$. “0” indicates the absence of fuzzy rule for the corresponding input condition. “1”, “2” and “3” mean that the cell in the corresponding buffer is to be transmitted. For the fuzzy system, the fuzzy rules are in the form “if $\langle antecedent_1 \rangle$ and $\langle antecedent_2 \rangle$ and $\langle antecedent_3 \rangle$ then $\langle conclusion \rangle$ ”. Two examples of rules are “if $\langle c_1 \text{ is } S \rangle$ and $\langle c_2 \text{ is } S \rangle$ and $\langle c_3 \text{ is } S \rangle$ then 1” and “if $\langle c_1 \text{ is } M \rangle$ and $\langle c_2 \text{ is } M \rangle$ and $\langle c_3 \text{ is } M \rangle$ then 2”. The first rule means that the cell in BUF_1 is to be transmitted while the second rule indicates that the cell in BUF_2 is to be sent. When these two rules fire, the final conclusion is determined

by aggregating the conclusions derived from both rules. An example of rule set for the startup of the system is shown in Table 7.1. All these rules can work together to fulfill the control scheme. The inference results related to “1”, “2” or “3” can be aggregated respectively and the final value of the aggregated results determines whether $class_1$, $class_2$ or $class_3$ is to be serviced.

Table 7.1: A startup rule set

$c_3 = S$		c_1		
		S	M	L
c_2	S	1	2	2
	M	1	3	3
	L	1	3	3

$c_3 = M$		c_1		
		S	M	L
c_2	S	1	2	2
	M	1	2	3
	L	1	2	3

$c_3 = L$		c_1		
		S	M	L
c_2	S	1	2	3
	M	1	2	3
	L	1	2	3

7.2.2 GA Encoding

According to the rule set presented in Table 7.1, the core rule set can be coded as “122133133,122123123,123123123”. The first 9 genes correspond to the case when $c_3 = S$. The second 9 genes correspond to $c_3 = M$, and the last 9 genes represent the rules for $c_3 = L$. Each sub-table grouping in Table 7.1 can be interpreted in a row-wise manner.

7.3 Fitness Function

Fitness function is an important component which directly affects the behavior of the evolvable system. In Chapter 6, where only two cell flows were considered, the QoS of $class_1$ is positively correlated to that of $class_2$. Hence, only the cell

delay of $class_1$ need to be considered in the fitness function. For a multi-channel application, it is far too complicated to distinguish between cell flows that are delay sensitive and cell flows which are loss sensitive. Since the main objective of the EFH is to approximate the performance of GPS, the average cell delay and bandwidth allocation of each cell flow are included in the fitness function.

The fitness function can be described by Eqs. 7.3.1-7.3.8 as follows.

$$\beta_i = 10 \times \left(\frac{n_i}{BitRate_i} - 1 \right) \quad (7.3.1)$$

$$\epsilon'_i = \begin{cases} \lambda \times 10^{\beta_i} & \text{if } \beta_i < 0 \\ \lambda & \text{else} \end{cases} \quad (7.3.2)$$

$$\epsilon_i = \begin{cases} \epsilon'_i & \text{if } \epsilon'_i < \epsilon_i \\ \epsilon_i & \text{else} \end{cases} \quad (7.3.3)$$

$$Ratio_i = \gamma \times \left| \frac{n_i}{n_1 + n_2 + n_3} - \frac{m_i}{m_1 + m_2 + m_3} \right| \quad (7.3.4)$$

$$ReqDelay_i = \alpha_i \times \rho \times v \quad (7.3.5)$$

$$RealDelay_i = \frac{\sum_{h=1}^{m_i} Delay(h)}{m_i} \quad (7.3.6)$$

$$Delay_i = |RealDelay_i - ReqDelay_i| \quad (7.3.7)$$

$$F = \kappa - \sum_{i=1}^3 \epsilon_i \times (Delay_i + Ratio_i) \quad (7.3.8)$$

The meaning of the variables in the above equations are outlined in Table 7.2.

Table 7.2: Definition of variables

Variable	Definition
β_i	Status of contract following of $class_i$
$BitRate_i$	Bit rate of the flow patterns stored in TB_i
n_i	Subscribed bandwidth of $class_i$
λ	A constant for the fitness weight factor
ϵ_i	Fitness weight factor for $class_i$
m_i	Number of cells transmitted from $class_i$
γ	Weight factor for bandwidth error
$Ratio_i$	Error of bandwidth ratio for $class_i$
α_i	Tuning factor of the average cell delay
ρ	Transmission time of one cell unit
v	Size of $BUF_{\#}$
$ReqDelay_i$	Required average delay for $class_i$
$RealDelay_i$	Average delay suffered by cells from TB_i
$Delay_i$	Error of average delay for $class_i$
$Delay(h)$	Delay suffered by every cells
κ	A constant
F	Fitness value

If $BitRate_i$ is larger than n_i , indicating that $class_i$ is misbehaving, β_i will be negative and hence ϵ_i in Eq.7.3.3 will be much smaller than λ . This will put less emphasis on the significance of the QoS of $class_i$ in Eq.7.3.8. In Eq.7.3.2, λ is a constant value. If β_i is larger than 0, ϵ'_i in Eq. 7.3.2 is assigned the value of λ , otherwise it is left unchanged. Eq.7.3.3 captures the smallest value of ϵ_i during the control process. This

equation indicates that if one cell flow overrides its contract, its QoS is subsequently deemphasized even if it recovers. This approach eliminates the bad effects on future flow patterns by past misbehaviour. m_i in Eq.7.3.4 is the number of cells from $class_i$ transmitted during the evaluation process. $\frac{m_i}{m_1+m_2+m_3}$ indicates the real bandwidth allocation during the evaluation process by the evaluated chromosome for $class_i$. $\frac{n_i}{n_1+n_2+n_3}$ represents the desired bandwidth ratio. $\left| \frac{n_i}{n_1+n_2+n_3} - \frac{m_i}{m_1+m_2+m_3} \right|$ thus is the error of the bandwidth ratio allocation for $class_i$. $ReqDelay_i$ indicates the desired average delay for cells from $class_i$. $RealDelay_i$ is the average delay suffered by cells from $class_i$ during evaluation. Eq.7.3.8 describes the calculation of the fitness value of each evaluated chromosome. This fitness model expresses the objective of small error in bandwidth allocation ratio and small difference in average delay between EFH and the desired average delay.

7.4 Simulations

In this section, the computation complexity of the evolution and evaluation process is first analyzed. Based on the analysis of the computation complexity, a parameter will be set in the following simulations to consider the time taken by the evolution and evaluation processes. EFH has a special property whereby the QoS performance can be tuned through parametric adjustment of the fitness function. In this section, the good performance achieved in terms of bandwidth allocation and average delay is demonstrated by comparisons with GPS, MCRR and FIFO. Then the property

of tunability is demonstrated by tuning α_i . For the simulations, it is supposed that the capacity of the output channel is 155.52M bits per second. For the CBR flows, the bit rate is constant and the gap between each packet is one bit time period. For VBR flow, the bit rate in every ON period is constant which is a normally distributed random variable. The gap between two adjacent ON periods is also a normally distributed random variable. The time gap between two adjacent packets is fixed as in the CBR case. All the simulations are carried out for 2 seconds of cell flows. The program for simulation is coded in C++. For the simplicity purpose, three input channels are consider in this simulation. Each scheduling scheme is implemented in a separate program.

7.4.1 Computation Complexity Analysis

In order to assess the implementability of the whole EFH on hardware circuit, it is useful to determine the computation cost of the evolution process. The faster the evolution, the better the system reacts to the changing patterns. It is assumed that multiplication and addition involved in Eqs. 7.3.1-7.3.8 are basic operations incurring the same computation cost of 1 time unit. The time unit can be defined by the most complicated operation such as addition or multiplication. Exponential operation can be realized by means of a lookup table. Thus the time taken by an exponential operation is only the memory access time. The number of the input channels considered is 3 ($j=3$). All the notations are the same as in Algorithms 1 and 2. The variables involved in the complexity analysis are described in Table 7.3.

Table 7.3: Variables description

Variable	Description
j	Number of input channels
m	Size of $TB_{\#}$
v	Size of $BUF_{\#}'$ and $BUF_{\#}$
g	Number of generations
p	Population size
c	Number of evolution cycle
l	Chromosome length in bits

The computation complexity of Algorithm 1 and Algorithm 2 are summarized in Table 7.4.

Table 7.4: Computation complexity of algorithm1 & algorithm 2

Algorithm	Operations	Computation Complexity
Algorithm 1	$TB_{\#}$ Modification	$m \times j$
	Initialization of $BUF_{\#}'$	$\rho \times j$
	Evolution & Evaluation	$g \times \rho \times E$
Algorithm 2	Enqueueing	$2 \times j \times m$
	Evaluation	$3 \times j \times (m + v)$
	Fitness Computation	22

For Algorithm 2, during the whole evaluation, there is maximally $m \times j$ cells arriving from $TB_{\#}$. Each arriving cell takes two operations such as modifying BUF_1' and incrementing the cell counter ($q_{\#}$). Thus the computation cost for the *Enqueueing* part in Algorithm 2 is at most $m \times j \times 2$ basic operations. The fuzzy inference can be realized by hardware implementation and it takes one basic operation time. Hence, the number of computations required by the *Evaluation* in Algorithm 2 is about $j \times (m + v) \times 3$. The value “3” corresponds to the number of operations within the *while* loop. Each operation is a fundamental operation. For the whole evaluation process, there are at most $j \times (m + v)$ cells to be processed, hence $j \times (m + v)$ times of execution of the *while* loop. The computation time taken by the *Fitness*

Computation in Algorithm 2 is about 22 time units, because each calculation in Eqs.7.3.1-7.3.8 in Section 7.3 takes about one operation cycle and there are about 22 operations in total. Thus the total number of operations for Algorithm 2 is $(m \times j \times 2) + [j \times (m + v) \times 3] + 22$ which can be represented by E .

For Algorithm 1, *TB_# modification*, filling out $TB_{\#}$ based on the corresponding bit rate when evolution is triggered, takes $m \times j$ operations. During *Initialization of BUF_#'*, the copying of cell information from $BUF_{\#}$ to $BUF_{\#}'$, requires a computation cost of $j \times v$. In the *Evolution and Evaluation*, the first internal *for* loop requires a total of $p \times (l + E)$ operations in which l is for the initialization cost of every chromosome. “ E ” is the computation cost of Algorithm 2. This is followed by two embedded *for* loops, which require approximately $g \times p \times (5 + E)$ operations in total. The value “5” is for the 5 instructions before the *Evaluation* operation. In Algorithm 1, the *Evaluation* subroutine is the most computationally intensive. In order to solve the bottleneck of the evolution speed, the p *Evaluation* processes can be realized through parallel hardware implementation. Hence, the *Evolution and Evaluation* routine in Algorithm 1 can be modified to Algorithm 3.

Adopting Algorithm 3, the number of operations required by Algorithm 1 is approximately $c \times [(p \times l + E) + g \times (5 + E)]$ operations. The parameters adopted in the following simulations are $m = 100$, $v = 100$, $g = 10$, $p = 10$, $j = 3$, $c = 2$ and $l = 9$.

Based on the above parameters, the total number of operations required by Algorithm 1 is approximately 50000 operations. Given a state-of-the-art microprocessor of 3000MIPS (*Million Instructions Per Second*), the evolution process requires less

---

Algorithm 3 Modified evolution

Evolution and Evaluation:

```

for  $i < c$  do
  for  $i_1 < p$  do
    Initial Population generation;
  end for;
  Call Evaluation();
  for  $i_2 < g$  do
    for  $i_3 < p$  do
      Toss for crossover;
      Select chromosomes for crossover;
      Do crossover operation;
      Toss for mutation;
      Do mutation operation;
    end for;
    Call Evaluation();
  end for;
end for;

```

---

than 20 microseconds of computation time. If hardware pipelining is adopted in the implementation, the computation cost can be further reduced. In the following simulations, a computation cost of 20 microseconds is considered for the evolution process. This means that the working chromosome is replaced by the searched elite only 20 microseconds after the trigger of the evolution process to simulate the delay of the evolution process.

7.4.2 Normal QoS Performance

In this section, three different cell flow scenarios are adopted to test the performance of EFH. In the first scenario, all the input channels are CBR with reserved bit rates. But caused by hardware problem or routing problem, $class_1$'s actual bit rate is much bigger than it's reserved bit. In the second scenario, $class_1$ is VBR while

$class_2$ and $class_3$ are CBR. Due to some unknown reasons, $class_1$'s actual bit rate varies among a range. In the third scenario, $class_1$ is CBR while $class_2$ is VBR and $class_3$ is CBR. In this scenario, $class_2$ and $class_3$ follow their contracts well but the actual bit rate of $class_1$ is bigger than its contract.

In the above scenarios, one channel breaks its contract and tries to transmit more data. The scheduling system should isolate the QoS of the good performing channels from being affected by the contract breaking flows. In these scenarios, the various combinations of CBR and VBR are input to the scheduling systems. Many other combinations were also simulated, however, these are not included in the thesis.

Scenario₁

In this scenario, $class_1$ is CBR with a reserved bit rate of 65.52M bits per second. $class_2$ and $class_3$ are also CBR with reserved bit rates of 50M bits per second and 40M bits per second respectively. Assume that for unknown reasons, $class_1$'s actual bit rate is 100M bits per second while that of $class_2$ and $class_3$ adhere to their contracts. For this case, the parameters in the fitness function can be set as follows, in which $\alpha_{\#}$ can be calculated by equation $\frac{t_i}{\rho \times v}$. Here, t_i refers to the possible transmission delay suffered by cells from $class_i$ in the GPS system based on its reserved bit rate. The parameters setting for the simulation are $\alpha_1 = 0.0237$, $\alpha_2 = 0.0312$, $\alpha_3 = 0.0387$, $\lambda = 1000$, $\gamma = 1000$, $\rho = 2.73$, $v = 100$ and $\kappa = 10^8$. λ should be a value which is much larger than 1. According to Eq.7.3.4 and 7.3.8, setting $\gamma=1000$ emphasizes the significance of the error of bandwidth allocation ratio to be

1000 times that of the delay error. The average delay for the four schemes based on simulation is presented in Table 7.5(a).

Table 7.5: Average delay for scenario₁, scenario₂ and scenario₃
(a) SCENARIO₁

	GPS(μ s)	EFH(μ s)	MCRR(μ s)	FIFO(μ s)
<i>class</i> ₁	650.2	647.9	620.5	647.4
<i>class</i> ₂	8.5	11.9	155.9	648.7
<i>class</i> ₃	10.6	15.5	86.5	650.3

(b) SCENARIO₂

	GPS(μ s)	EFH(μ s)	MCRR(μ s)	FIFO(μ s)
<i>class</i> ₁	558.8	555.2	557.2	589.2
<i>class</i> ₂	8.5	10.1	59.3	588.9
<i>class</i> ₃	10.6	13.0	39.0	589.5

(c) SCENARIO₃

	GPS(μ s)	EFH(μ s)	MCRR(μ s)	FIFO(μ s)
<i>class</i> ₁	1421.6	147.2	1767.8	63.0
<i>class</i> ₂	4.9	6.2	68.0	82.9
<i>class</i> ₃	10.6	12.2	859.9	63.6

From the table, it can be seen that FIFO dealt with all the flows equally, thus the three flows show similar average delay. MCRR achieved much bigger delays for the contract following flows than that achieved by GPS. EFH achieved good average delay for the input flows of *class*₂ and *class*₃, which observe their contracts. In this scenario, the delay of *class*₂ and *class*₃ achieved by EFH is very close to that of GPS.

The bandwidth utilization ratio for this scenario is as shown in Table 7.6(a). In this scenario, the bandwidth ratio is sampled every 50 microseconds. Table 7.6 shows the means and standard deviations for the three input flows over these samples. Based on this table, MCRR's performance is closer to that of GPS than EFH and FIFO in terms of mean and standard deviation. But the difference in performance

of these four schemes is very slight. By considering both the QoS of delay and the bandwidth allocation, EFH is much better than MCRR and FIFO.

Table 7.6: Bandwidth utilization ratio in scenario₁, scenario₂ and scenario₃
(a) SCENARIO₁

	GPS		EFH		MCRR		FIFO	
	mean	dev	mean	dev	mean	dev	mean	dev
<i>class</i> ₁	0.4214	$1.15 \times e^{-4}$	0.4205	0.0012	0.4208	$3.66 \times e^{-4}$	0.4218	0.0042
<i>class</i> ₂	0.3215	$8.56 \times e^{-5}$	0.3219	$7.41 \times e^{-4}$	0.3217	$2.17 \times e^{-4}$	0.3213	0.0023
<i>class</i> ₃	0.2572	$5.46 \times e^{-5}$	0.2575	$4.61 \times e^{-4}$	0.2575	$1.65 \times e^{-4}$	0.2570	0.0019

(b) SCENARIO₂

	GPS		EFH		MCRR		FIFO	
	mean	dev	mean	dev	mean	dev	mean	dev
<i>class</i> ₁	0.4208	0.0036	0.4200	0.0112	0.4266	0.0914	0.4219	0.0150
<i>class</i> ₂	0.3219	0.0021	0.3221	0.0053	0.3183	0.0501	0.3212	0.0084
<i>class</i> ₃	0.2574	0.0071	0.2579	0.0076	0.2551	0.0455	0.2569	0.0067

(c) SCENARIO₃

	GPS		EFH		MCRR		FIFO	
	mean	dev	mean	dev	mean	dev	mean	dev
<i>class</i> ₁	0.2160	0.0134	0.2890	0.0447	0.1209	0.1501	0.3150	0.0174
<i>class</i> ₂	0.4961	0.0312	0.4515	0.0490	0.6976	0.2497	0.4331	0.0311
<i>class</i> ₃	0.2879	0.0178	0.2595	0.0082	0.1816	0.1456	0.2520	0.0138

Scenario₂

In this scenario, *class*₁ is VBR with the reserved bit rate of 65.52M bits per second while *class*₂ and *class*₃ are CBR with the reserved bit rates of 50M bits per second and 40M bits per second respectively. Assume that for some unknown reasons, *class*₁'s actual bit rate varies between 55.52M bits per second and 105.52M bits per second randomly during its ON period and *class*₂ and *class*₃ adhere to their contracts well. For this scenario, the parameters setting can be the same as that of *scenario*₁. From simulation, the average delay for the four schemes is presented

in Table 7.5(b). From the table, it can be seen that FIFO achieved similar average delays for all the three cell flows. MCRR achieved better delay performance for the contract following flows, but the delay for $class_2$ and $class_3$ are still not good. On the other hand, EFH achieved good average delay for the input flows that adhere to their contracts. In the table, the delay of $class_2$ and $class_3$ achieved by EFH is very close to that of GPS.

The corresponding bandwidth utilization ratio for this scenario is as shown in Table 7.6(b). In this scenario, $class_1$ is a bursty flow. The congestion may only occur during the ON periods of $class_1$, the bandwidth utilization ratio is averaged over each bursty period of $class_1$. Based on the mean and deviation values in Table 7.6(b), EFH performs better than MCRR and FIFO. The mean of the bandwidth allocation ratio for the three flows achieved by EFH is closer to that of GPS than using MCRR or FIFO. Furthermore, the bandwidth allocation deviations by EFH are also much smaller than that of MCRR and FIFO.

Scenario₃

In this scenario, $class_1$ is CBR with reserved bit rate of 30M bits per second while $class_3$ is CBR with reserved bit rate of 40M bits per second. $class_2$ is VBR with reserved bit rate of 85.52M bits per second, but its actual bit rate varies between 55.52M bits per second and 85.52M bits per second randomly during its ON period. In this scenario, $class_2$ and $class_3$ follow their contracts well but the actual bit rate of $class_1$ is 50M bits per second. The related parameters in the fitness function can

be set as follows according to the GPS model: $\alpha_1 = 0.0519$, $\alpha_2 = 0.0183$ and $\alpha_3 = 0.0387$. The average delays for the four schemes based on simulations are presented in Table 7.5(c). From the table, it can be seen that FIFO achieved average delays within a tight range for the three flows as in *scenario*₁ and *scenario*₂. MCRR achieved poor delay performance for the three flows. EFH achieved good average delay in this scenario. The delay of *class*₂ and *class*₃ in this scenario achieved by EFH is very close to that of GPS.

The bandwidth utilization ratio for this scenario is as shown in Table 7.6(c). For this scenario, *class*₂ is a bursty flow. The congestion may only occur during the ON period of *class*₂, the bandwidth utilization ratio is averaged over each bursty period of *class*₂. MCRR shows a very good overall bandwidth utilization ratios of 0.1931, 0.5494 and 0.2575. They are close to the corresponding reserved bandwidth ratios of $\frac{30}{155.52} = 0.1929$, $\frac{85.52}{155.52} = 0.5499$ and $\frac{40}{155.52} = 0.2572$. However, its bandwidth utilization ratios during the bursty periods are poor. As shown in the table, its performance is far from that of GPS while EFH achieved good performance, very close to that of GPS.

7.4.3 QoS Tunability

A special property of EFH is its QoS tunability. Suppose the average delay of *class*₂ in *scenario*₁ needs to be 60 microseconds. The value of α_2 can be calculated by $\frac{60}{\rho \times v} = \frac{60}{2.73 \times 100} = 0.2198$. The values of α_1 and α_3 are the same as that in the previous simulation of *scenario*₁. The average delay of EFH for all the flows with

$\alpha_2 = 0.2198$ is listed in Table 7.7.

Table 7.7: The tuned average delay for scenario₁

	GPS(μ s)	EFH(μ s)	MCRR(μ s)	FIFO(μ s)
<i>class</i> ₁	650.2	645.2	620.5	647.4
<i>class</i> ₂	8.5	50.0	155.9	648.7
<i>class</i> ₃	10.6	14.6	86.5	650.3

Comparing Table 7.7 to Table 7.5, it can be deduced that, after tuning the value of α_2 , only the average delay of *class*₂ for EFH shows a significant change. The average delay of *class*₂ increases significantly. The average delay of *class*₃ is almost the same as that in Table 7.5. The achieved average delay of *class*₂ is less than the desired value of 60 microseconds. This is because the EFH is an open loop system rather than a closed-loop feedback system. Another reason for the error is that the GA parameters adopted in this system for real-time implementability. For example, if bigger population size and bigger generation number are used, the error will be much smaller. The property of *tunability* is very useful in the case where one or more good performing flows need bigger queuing delay. By tuning the parameters of $\alpha_{\#}$, these requirements can be satisfied without affecting the other good performing flows.

Chapter 8

Hardware Implementation of EFH

Evolvable hardware (EHW) is a combination of reconfigurable chips and evolutionary algorithms. A good EHW will be able to track the changing situations and adapt to the changes to maintain a good performance. The study of EHW is still relatively new. Previous research works are not so practical for real-time applications due to its heavy reliance on external computation platform to execute the evolutionary search. In order to overcome this problem and present an intrinsic evolvable and on-line adaptive EHW system, we proposed an evolvable fuzzy system (EFS) [44] and further, a conceptual architecture of evolvable fuzzy hardware (EFH) system [45], [46].

Currently, there are three types of EHW classified according to their evolution granularity; *transistor level*, *gate level*, *functional-level* EHW [14]. *Transistor level* EHW focuses on very small size analog circuit such as amplifier and digital-to-analog con-

verter. Earlier reported work on this can be found in [17,20,21]. *Gate level* EHW carries out evolution on primitive digital circuits such as AND gates, OR gates, XOR gates and so on. In order to evolve a 5×5 multiplier, even with technique such as partitioned training sets to improve the evolution process, a gate-level EHW has to evolve 2.2 million generations for 32 partitions and 13.7 million generations for 16 partitions. The generation number will increase by about six-fold if a circuit that is double in size is required [40]. All these are based on the need for a high performance computation platform to carry out the evolution computation. In order to improve on the evolution efficiency, *divide-and-conquer* approach can also be adopted [42]. This approach to evolution is based on the use of more complex building blocks, such that more available knowledge is integrated into the building blocks without needing further evolution. It can be an effective way to accelerate the evolution process of EHW. *Functional-level* EHW follows this thought. From the perspective of system level, *functional-level* EHW is much more applicable than *transistor-level* EHW and *gate level* EHW. In [8,75], *function-level* EHW was proposed for the control of ATM cell scheduling. This is an evolvable system based on a FPGA consisting of 100 *programmable floating processing units* (FPU). Each FPU can be programmed to perform different floating point operations. The evolvable system works with 400 chromosomes for 2500 generations. Each allele consists of 4 integer array representing the working status of each FPU. A total of 48 groups of training data sets were applied during the evolution process. Obviously, such an evolution process cannot be fulfilled in a very short time even with the latest, most advanced computer system. On the other hand, this system is also too complicated

to be implemented as an ASIC chip.

Our proposed EFH system also follows the *divide-and-conquer* strategy while further improves the knowledge integration in the evolving blocks. As reported in our previous work [46], the whole system we have developed evolves 25 fuzzy rules. Each rule can only have 3 situations: 0, 1 or 2. In such a case, with the changes of the evolved fuzzy rule set, some parts of the *reconfigurable fuzzy inference chip* (RFIC) are activated and other parts are deactivated. The connections between the FIM block and the aggregation circuit are also changed. In other words, all these changes are based on the evolutionary algorithm. In this system, the population size is 10, the total generation number is less than 20 and the training set is a small part of the cell flows. All these indicate that the system complexity is much less than other kinds of EHW and the evolutionary algorithm is much easier to be implemented than in other EHWs. This chapter will discuss the implementation of the whole EFH system for the ATM cell-scheduling application. The implementation includes chromosome sorting block, crossover operation block, mutation block, evolution controller, evaluation block and the modified RFIC block. These all can make up of the EFH system. All the circuit blocks in this thesis were described by using Verilog HDL language, simulated by ModelSim and synthesized by Design Compiler. The synthesis target technology library is TSMC 0.18 μ m

This chapter will be organized as follows. In section 8.1, the whole EFH will be briefly reviewed. In section 8.2, the hardware implementation of different blocks will be presented in detail. The synthesis and the simulation results will also be

presented in this section.

8.1 System Description

In [44, 46], two types of ATM cell flow, *constant bit rate* (CBR) and *variable bit rate* (VBR) are scheduled. The cell flow characteristics of CBR and VBR have been shown earlier in Fig. 5.2 and Fig. 5.3 respectively.

For CBR, the cell flow always maintain the same cell bit rate as PCR (peak cell rate). For VBR, it is significantly different. The cell flow is a bursty traffic. There are ON periods where the cells are sent at a prescribed bit rate and OFF period where no cell is sent. The cell bit rates for different ON periods can vary randomly.

With evolvable systems, usually a large training set is required to represent various kinds of working situations. The training set is fed into the system to guide the evolution. This inevitably increases the computation cost of the evolutionary algorithm. In contrast, EFH adopts the principle of “locality” to collect training data on-line and in small manageable time window. With a small time window, not only the training time is reduced to a practically real-time level, a better approximation of the data flow in the subsequent time window is achieved. The current flow can be adopted as a model to evaluate the chromosomes which will be used to perform cell flow control for the subsequent time period. This approach works well, judging from the simulation results presented in the previous sections. The chromosome consists of 25 integers. Each gene can be the value of “0”, “1” or “2”. “0” indicates that no

fuzzy rule is needed for the input situation. “1” means that CBR cell flow will be processed and “2” represents VBR cell flow being scheduled.

The system block architecture of EFH is as shown in Fig. 8.1. BUF₁, BUF₂ and MP make up the normal scheduling system. RFIC is the *reconfigurable fuzzy inference chip* to accommodate the appropriate evolved context and to carry out fuzzy inference for the scheduling system. The *evolution module* carries out the evolution operations. The final best chromosome searched by the *evolution module* is configured into the RFIC to carry out the control. In chapter 4 and in our papers [31,43], the whole system is based on 5×5 fuzzy rule sets. For simplicity, we demonstrate the implementation based on 3×3 fuzzy rule sets in this chapter. The input fuzzy variables are defined as $c_1 = \frac{V_1}{V_{max}}$ and $c_2 = \frac{L_2}{L_{max}}$. V_1 represents the current cell bit rate of CBR and V_{max} refers to the capacity of the input line. L_2 is the current available unit number of BUF₂ and L_{max} is the size of BUF₂. The membership functions of c_1 and c_2 are shown in Fig. 8.2. An example of a chromosome can be “122122122”.

Figure 8.1: System architecture

Figure 8.2: Membership functions

8.2 Hardware Architecture

The following parts will discuss the hardware implementation of the genetic operations and the evaluation process, the key components in the whole EFH system. We adopt a parallel scheme to process each chromosome. In this implementation, each gene is represented by two binary bits.

8.2.1 Hardware Architecture of Evolution System

The system architecture of the whole evolution system is as shown in Fig. 8.3. In the figure, the Random Number Array block is made up of random number generators (RNG) for initializing the population. There are 10 generators in this block, $RNG_1, RNG_2, \dots, RNG_{12}$. Each $RNG_{\#}$ corresponds to one chromosome. Implementation wise, $RNG_{\#}$ is an 18-bit *linear feedback shift registers* (LFSR), a very popular scheme for pseudo random number generation [80]. There are two other cells in this block, RNG_{11} and RNG_{12} . These two are not necessarily random number generators. RNG_{11} stores the core rule set which will also take part in the GA operations. RNG_{12} stores the working chromosomes only for evaluation purpose. The output of the Random Number Array block consists of 18×12 bits. Each 18-bit

data corresponds to a chromosome. The architecture of the RNG_# is as shown in Fig. 8.4. The outputs of stage 1, 5, 10 and 14 are fed back to the first register bit after an exclusive OR operation. Driven by a clock signal, the content of the LFSR changes continuously. Chromosome Buffer Array in Fig. 8.3 is a register array. Each cell block is configured as an 18-bit parallel-input-parallel-output data register configured using D flip-flops. All the chromosomes that make up the population are buffered in this block. The chromosomes in the Chromosome Buffer Array are then forwarded to the corresponding cells within the Evaluation Array where each chromosome is evaluated. Each Evaluation_# requires many clock cycles. To improve on the speed of evolution, each evaluation block can be parallelized. All the chromosomes and their fitness values after evaluation become the inputs for every cell in the Sort Array. In this block, each cell carries out a dedicated sorting. For example, Sort₁ sorts out the chromosome with the highest fitness value, while Sort₂ sorts out the chromosome with the second highest fitness value, and so forth. The sorted chromosomes go through Crossover Array and Mutation Array. We set elite pool size to be 3. Because of this, the first three chromosomes pass by the Crossover Array and Mutation Array without any operations. The remaining 8 chromosomes undergo *crossover* and *mutation* by the corresponding cells in the two blocks. The outputs of the Mutation Array are stored in Chromosome Buffer Array for further evaluation. When the whole evolution process is completed, the BestChrom which is the output of Sort₁ is compared with the one from Evaluation₁₂. The better one will finally be downloaded into the RFIC in Fig. 8.1. In the following parts, the implementation of the *sort cell*, *crossover cell*, *mutation cell* and *evaluation cell* will

be discussed in details.

8.2.2 Sort Cell

Fig. 8.5 shows the architecture of the cell of Sort_1 . The circuit consists of 11 comparators $\text{Comp}_1, \text{Comp}_2, \dots, \text{Comp}_{11}$, a LargerDecoder, a EqualDecoder and a De-Multiplexer. There are two parts in this circuit. The first part is the comparator array where the fitness value of Chromosome_1 is compared with that of every others. The 2-bit output of the comparator indicates the fitness value of Chromosome_1 is bigger than others if it is “11”. The output with value “10” indicates the two input fitness values are equal. Otherwise the output is “01”. The output of every comparator become the input to the LargerDecoder and EqualDecoder. LargerDecoder detects the number of chromosomes with fitness values greater than Chromosome_1 . EqualDecoder detects the number of chromosomes before Chromosome_1 with the same fitness value as Chromosome_1 , including itself. If no chromosome which is in front of Chromosome_1 has equal fitness value, EqualDecoder outputs 0. The sum of these two blocks’ outputs indicates the position of Chromosome_1 in the sorted sequence. If all the chromosomes have the same fitness value, the sorting sequence will remain the same as the original sequence of $\text{Chromosome}_1, \text{Chromosome}_2$ and so forth. The second part is the De-Multiplexer. The De-Multiplexer channels the Chromosome_1 to the appropriate output. Chromosome_1 with a i th biggest fitness value is output to S_i and the other outputs are set to hi-Z (high impedance) state. The S_i output from the Sort cells are connected together to form the input to the

Figure 8.3: GA system architecture

Figure 8.4: Architecture of $RNG_{\#}$

Crossover cells.

The architectures of the cells in the Sort Array are similar. The difference between $Sort_1$ and the other cells is very slight. For every comparator in Fig. 8.5, there is a common input $Fitness_1$. In $Sort_i$, the common input is replaced by $Fitness_i$. The input of the De-Multiplexer is also replaced by the $Chromosome_i$ in $Sort_i$.

Figure 8.5: The architecture of sort cell

8.2.3 Crossover Cell

Crossover cell is made up of two parts, pseudo random number generator and operation block. The pseudo random number generator in this cell is also a LFSR, which is very similar to the LFSR component in the Random Number Array. The difference is that in this part, the LFSR is 21 bits. The first 5 bits is for “Crossover?”. The *crossover rate* adopted is 0.6, which is approximately $\frac{20}{32}$. This means that if the decimal equivalent value of the first 5 bits is no larger than 19, *crossover* should be carried out. The second 6 bits is for choosing the first parent and the third 6 bits is for choosing the second parent. The last 4 bits is for the *crossover point*. The normal scheme for choosing parents is *roulette wheel*. The *roulette wheel* is a mechanism to choose chromosomes in a random way with a bias or preference towards better chromosomes. A direct *roulette wheel* scheme is complicated to implement. To avoid the complications of a direct *roulette wheel* scheme, we adopt random number range shown in Table 8.1 to emulate the *roulette wheel*.

According to the table, if a random number falls into a specific inclusive range, the corresponding chromosome is adopted as a parent. As shown in the table, the better the chromosome, the bigger the slot allocated and the bigger the chance for the chromosome to be selected. This means that the fitter the chromosome is, the bigger the chance for it to be selected as parent. One advantage of this scheme is that fitness scaling is not necessary because the chance of the chromosome to be a parent depends on the position it is in the sequence, not on the fitness value. In the case when one chromosome has much higher fitness value than the others, even

if the fitness scaling is performed before the roulette wheel operation, at times this chromosome may still dominate in terms of reproduction. Another advantage of our scheme is that it can avoid this and maintain the salient characteristics of a GA global search.

Table 8.1: Pseudo roulette wheel

Random Number Range	Parent
52-63	chrom1
41-51	chrom2
31-40	chrom3
23-30	chrom4
16-22	chrom5
10-15	chrom6
6-9	chrom7
3-5	chrom8
1-2	chrom9
0	chrom10

The operation block of the Crossover_{*i*} is shown in Fig. 8.6. S_# is the output of the Sort_#, shown earlier in Fig. 8.3. S₁ is from Sort₁, S₂ is from Sort₂, and so forth. If the value of “Crossover?” is “0”, the corresponding original chromosome determined by signal Seq is output to Ch₁ and Ch₂. Otherwise, output Ch₁ is the chromosome selected by the input Chr₁ and Ch₂ is the chromosome selected by Chr₂. En₁ and En₂ are two 9-bit signals. If *crossover* is to be performed, En₁ and En₂ are generated according to the *crossover point* to enable the first part of Ch₁ and the second part of Ch₂ to output. The second part of Ch₁ and the first part of Ch₂ output “hi-Z” state. These outputs make up the new chromosome for further genetic operations.

Figure 8.6: Crossover Cell

8.2.4 Mutation Cell

Similar to the *crossover cell*, the *mutation cell* also has two parts, pseudo random number generator implemented by LFSR and an operation block. In *mutation cell*, LFSR only has 11 bits. The first 5 bits is for “Mutate?”. The *mutation rate* adopted is 0.05, which is approximately $\frac{2}{32}$. This means that if the decimal equivalent value of the first 5 bits is less than 2, *mutation* should be performed. The second 4 bits is for the *mutation point*. The last 2 bits is for the *mutation value*. The operation part of *mutation cell* is shown in Fig. 8.7. Chrom is from the corresponding *crossover cell*. If the value of “Mutate?” indicates that no mutation is to be performed, Chrom goes directly to the output NChrom. Otherwise, *mutation* is executed accordingly. *Mutation point* denotes the gene that will be mutated. If *mutation* is to be done, *mutation value* replaces the corresponding bits in Chrom. NChrom is the result of the mutated chromosome.

Figure 8.7: Mutation cell

8.2.5 System Controller

The controller for the system can be described by the interface block in Fig. 8.8 and the *finite state machine* (FSM) in Fig. 8.9. The major tasks executed in each state is shown in Table 8.2. S0 is the start of the FSM. When either of the TB_# is full, FSM will be in S1. Otherwise, FSM stays at S0. In S1, initialization of the population is carried out. It is the start of the first evolutionary cycle. Due to the characteristics of the Random Number Array in Fig. 8.3, the population can be obtained in one clock cycle. S2 is the state to perform evaluation. Evaluation requires several cycles, depending on the status of TB_# and TFIFO_#' in Fig. 8.10. When the evaluation process is finished, FSM moves to S3 to sort out the chromosomes within one cycle. The sorted chromosomes will be presented at the output of the Sort Array in Fig. 8.3 in decreasing order of fitness values sequence. There is a counter named GenCounter in the FSM. If this counter reaches a pre-fixed generation number, the FSM moves to S6 from S3 to start the second evolutionary cycle. Otherwise the following two states of the FSM are S4 and S5, during which *crossover* and *mutation* are carried out respectively. After S5, FSM will move back to S2 to evaluate the new population. S6 to S10 perform similar functions as S1 to S5. The only difference is that if the generation number is reached, the FSM will move to S11 instead of S6. In S11, the final best chromosome is downloaded into RFIC in Fig. 8.1. After that, FSM moves to S0 to wait for the next evolution.

The interface signals of FSM are shown in Fig. 8.8. TB₁Full and TB₂Full are from TB_# representing their status. EvaluationFin is from Evaluation cell indicating the

Figure 8.8: Interface of finite state machine

Table 8.2: Descriptions of each state

State Name	Description
S0	Initial State
S1, S6	Initial Population Generation
S2, S7	Perform Evaluation
S3, S8	Sorting out Chromosomes
S4, S9	Crossover
S5, S10	Mutation
S11	Download the Final Chromosome into RFIC

completeness of the evaluation process. Rng₁En controls the generation of initial population. Rng₂En and Rng₃En control the generation of random numbers for crossover and mutation respectively. ChromOk indicates that the GA operation is completed and the chromosome output by the Mutation Array can be registered into the Buffer Array and evaluated. EvaluationStart is to trigger the start of the evaluation process. EvolutionFin tells the RFIC (see Fig. 8.1) that the final chromosome after evolution is ready. The chromosome will be registered by the RFIC for actual control of the multiplexing operations.

1. TB _# Full is ON.	10. Move with no condition.
2. None of TB _# Full is ON.	11. EvaluationFin is "0".
3. Move with no condition.	12. EvaluationFin is "1".
4. EvaluationFin is "0".	13. Move with no condition.
5. EvaluationFin is "1".	14. GenCounter is 10.
6. Move with no condition.	15. Move with no condition.
7. GenCounter is 10.	16. Move with no condition.
8. Move with no condition.	17. Move with no condition.
9. Move with no condition.	

S₀: ChromOk=1, Rng₁En=1, Rng₂En=1, Rng₃En=1, EvolutionFin=0, EvaluationStart=0
 S₁: ChromOk=1, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₂: ChromOk=0, Rng₁En=0, Rng₂En=1, Rng₃En=1, EvolutionFin=0, EvaluationStart=1
 S₃: ChromOk=0, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₄: ChromOk=0, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₅: ChromOk=1, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₆: ChromOk=1, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₇: ChromOk=0, Rng₁En=0, Rng₂En=1, Rng₃En=1, EvolutionFin=0, EvaluationStart=1
 S₈: ChromOk=0, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₉: ChromOk=0, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₁₀: ChromOk=1, Rng₁En=0, Rng₂En=0, Rng₃En=0, EvolutionFin=0, EvaluationStart=0
 S₁₁: ChromOk=0, Rng₁En=1, Rng₂En=1, Rng₃En=1, EvolutionFin=1, EvaluationStart=0

Figure 8.9: Finite state machine

8.2.6 Evaluation Cell

Evaluation cell is made up of $TB_{\#}$, $TFIFO_{\#}$, TRFIC, TMP and a Fitness block. $TB_{\#}$ is a 100 units *first-in-first-out* buffer. It processes the CBR or VBR cells and stores their information such as *cell number*, *arrival time* and *bit rate* to be used in the evaluation process. $TFIFO_{\#}$ is also a 100-unit *first-in-first-out* buffer similar to the $TB_{\#}$. The implementation of *evaluation cell* is as shown in Fig. 8.10. TMP is a pseudo multiplexer. It is used to simulate the function of the MP in Fig. 8.1 and calculate CellDelay and CellNumb. When the evaluation is finished, TMP sets EvaluationFin. The TRFIC is a RFIC block. All the signals involved in the *evaluation cell* are described in Table 8.3. CBR and VBR are from Fig. 8.1. $TB_{\#}Full$ denotes the status of $TB_{\#}$ having already stored 100 cell units. CBR' and VBR' are signals from $TB_{\#}$. CBR' and VBR' include *cell number* (20bits), *arrival time* (28 bits) and *bit rate* (16 bits). $Time_1$ and $Time_2$ are *arrival time* of CBR' and VBR' respectively. $Time_3$ is the time when the current transmission will finish. $Time_{\#}$ is processed by the Comparator and the smallest value is output at T_1 representing the current simulation time. Chromosome is from $BUF_{\#}$ in Fig. 8.3. c_1' and c_2' are the inputs of the TRFIC for fuzzy inference. EvaluationStart is from the FSM to trigger the evaluation process. CellDelay (20 bits) and CellNumber (12 bits) are the outputs of the MP. They will be used to calculate the fitness value.

The only difference between the *evaluation cell* and the true multiplexer is that in the *evaluation cell*, only the cell information is processed, while in the true multiplexer, the real cell unit is processed and transmitted out bit by bit. The architecture of

the RFIC and the Fitness block will be presented next.

Figure 8.10: Evaluation cell implementation

Fitness Value Calculation

The architecture of the Fitness block is shown in Fig. 8.11. The fitness function adopted in [45, 46] is as described by Eq.4.2.1 in page 47. In our work, we adopt K as “0xFFFFFFFF”, because in C++ simulation, κ is supposed to be a very large numerical constant as in Eq.4.2.1. DelayFactor is set at $100 \times 100 \times 2.73$. This is roughly the maximum delay that may be suffered by a cell unit. 100 is the length of the cell flow. 2.73 is the transmitting delay of every cell. Supposing the cells from one class is always assigned priority while cells from another class are always kept waiting, the maximum delay suffered by the last cell is about 100×273 .

In Fig. 8.11, CellDelay is from Fig. 8.10. It is accumulated by the Adder to derive the AccCellDelay. AccCellDelay is divided by CellNumber to obtain the AverageDe-

Table 8.3: Signal description

Signal	Description
CBR	CBR cell flows
VBR	VBR cell flows
TB_1Full	Indicating TB_1 training buffer is full
TB_2Full	Indicating TB_2 training buffer is full
CBR'	cell flows from TB_1
VBR'	cell flows from TB_2
$Time_1$	Arrival time of CBR cell
$Time_2$	Arrival time of VBR cell
$Time_3$	The finish time of the current transimion
T_1	The smallest value of $Time_{\#}$
$Empty_1$	Indicating $TFIFO_1$ is empty
$Empty_2$	Indicating $TFIFO_2$ is empty
TChromosome	The chromosome being evaluated
c_1'	Fuzzy variable described in section 8.1
c_2'	Fuzzy variable described in section 8.1
$Cell_1'$	The head cell in $TFIFO_1$
$Cell_2'$	The head cell in $TFIFO_2$
ChAva	Indicating transmission buffer in TMP is empty
CellDelay	Delay of the last cell transmitted
CellNumb	Transmitted cell number
EvaluationFin	The current evaluation is finished
EvaluationStart	From FSM to trigger the evaluation process

lay. The Divider block is implemented by multiplying AccCellDelay and $\frac{1}{\text{CellNumber}}$. The possible maximum value of CellNumber is 2 times the size of $\text{TFIFO}_{\#}$ plus 2 times the size of $\text{TB}_{\#}$. This indicates that the maximum of CellNumber is 400. Thus the calculation of $\frac{1}{\text{CellNumber}}$ is possible to be implemented by a *look up table* (LUT).

The Multiplier block is to calculate $\alpha \times \text{DelayFactor}$. It can also be implemented as a LUT. α in Eq. 4.2.1 can be 0.1, 0.2, $\dots$, 1.0. The input signal Alpha in Fig. 8.11 has 4 bits representing various values of α . The Abs block is to obtain the absolute value as per Eq. 4.2.1. The last component in Fig. 8.11 is a subtractor to calculate the final fitness value.

Figure 8.11: Architecture of fitness calculation

RFIC implementation

RFIC is a critical part in the proposed EFH. It can accommodate different fuzzy rule set with a very low reconfiguration overhead. RFIC is composed of CMU, FIM and OAM. The detailed information on RFIC can be found in [31]. The whole architecture of the RFIC is as shown in Fig. 8.12.

Figure 8.12: The architecture of RFIC

The CMU is an 18-bit simple register circuit. It stores the working chromosome and generates $Ena_{<xy>}$ and $Sel_{<xy>}$ signals. If there is a rule in a specific allele, the corresponding $Ena_{<xy>}$ is set to “1”. Otherwise, it is set to “0”. $Sel_{<xy>}$ depends on the fuzzy conclusion of the specific rule represented by the gene. The output is “1” if the fuzzy conclusion represented by the gene corresponds to CBR being allocated the time slot. It is “0” if VBR is to be allocated the time slot. FIM is a 3×3 partition blocks. Each block stores a fuzzy inference map. One example is in Table

8.4. c_1 and c_2 determine which unit in the partition block is activated. $Ena_{<xy>}$, $Sel_{<xy>}$, c_1 and c_2 determines the output signals of the FIM block, D_{11} , D_{12} , $\dots$, D_{33} . OAM block performs the fuzzy aggregation and derives the final conclusion.

The configuration map of the FIM block is as shown in Fig. 8.13. Each of the PB blocks is a LUT based component. It covers all the fuzzy inference scenarios for specific fuzzy inputs. For example, the content of PB_{11} is as shown in Table 8.4. The left most bit is the most significant bit. The numbers in the Address column represent the digitized values of c_1 (5 bits), c_2 (5 bits) and $Sel_{<xy>}$ (1 bit) (see Fig. 8.12). The values in the Content column represent the fuzzy conclusion (1 bit) and its fuzzy strength(5 bits). If the first two bits of the CMU register is “01” (representing VBR), $Sel_{<11>}$ is “0” and $Ena_{<11>}$ is “1”. This indicates that PB_{11} is active. Now if c_1 and c_2 are “00000” and “00010” respectively, the last 4 bits of c_1 and c_2 make up of the address to access PB_{11} . This will access the output “011011”, indicating the fuzzy conclusion is VBR and the fuzzy strength is “11011”. The outputs of $PB_{<11>}$, $PB_{<12>}$, $\dots$, $PB_{<33>}$ form the inputs to the OAM which derives the final conclusion.

The architecture of the OAM is shown in Fig. 8.14. This OAM is a modification of the one shown in [31, 46]. The blocks which make up the OAM is not an Ave₂ or Ave₃ block as in [31, 46] but a Sum/Sub block. If the most significant bits of the inputs are the same, the Sum/Sub block performs addition. If the most significant bits of the inputs are different, the Sum/Sub block performs subtraction and the most significant bit of the output depends on the first bit of the bigger input. T_F,

PB_{11} S*S	PB_{12} S*M	PB_{13} S*L
PB_{21} M*S	PB_{22} M*M	PB_{23} M*L
PB_{31} L*S	PB_{32} L*M	PB_{33} L*L

Figure 8.13: The configuration map of the FIM block

Table 8.4: Contents of PB_{11}

Address	Content
000000000	011111
000000001	111111
000000010	011101
000000011	111101
000000100	011011
000000101	111011
000000110	011001
000000111	111001
000001000	010111
000001001	110111
000001010	010101
.	.
.	.
.	.
111111100	000001
111111101	100001
111111110	000001
111111111	100001

the final output of the OAM, is a one bit signal having two possible values of “0” and “1”. “1” denotes CBR gets the bandwidth and “0” indicates VBR can be transmitted.

Figure 8.14: The architecture of OAM

8.2.7 Simulation and Synthesis

The simulation results for the whole system are carried out by using FPGA Advantage from Mentor Graphic. The whole circuit is described by Verilog HDL language. Each cell is represented by a 64-bits number indicating its *cell number*, *bit rate* and *arrival time*. The cell flows in the simulation last for 100 ms, enough to observe the functional behaviour of the circuit. These 100 ms cell flows are fed to the C++ simulator and the Verilog simulator. The simulated results are compared later. This is to prove that the behaviour of the circuit and the C++ system are the same. The results for $\alpha=0.3$ are shown in Fig. 8.15. A snapshot of the timing waveforms

for $\alpha=0.3$ is shown in Fig. 8.16 indicating the evolution of the *bestchrom*. In the waveforms, Flow₁ and Flow₂ represents the CBR and VBR flows. Their format is similar to that presented in Fig. 5.5. c_1 and c_2 are the fuzzy variables. c_1 is constant due to its definition. c_2 keeps changing representing the status of BUF₂. In the two figures, signal Alpha represents the value of α in Eq. 4.2.1.. The value “3” represents $\alpha=0.3$. Flow1 and Flow2 represent the cell flow of CBR and VBR respectively. Signals c_1 and c_2 are the outputs from FIFO₁ and FIFO₂ for fuzzy inference respectively. AccCellDelay is the accumulated cell delay for *class*₁ flow. CellNumber is the number of cells sent during the multiplexing process. AveDelay is the average cell delay. It is the quotient of AccCellDelay and CellNumber. The final AveDelay value is “d7d” in Hex format, which equals “3453” in decimal. The time unit of AveDelay is 0.01 microseconds. Thus the final average cell delay is 34.53 microseconds. This value is needed to be compared with the cell delay achieved by the C++ simulator. This value differs significantly compared to the 70 microseconds achieved using the C++ simulation program. The main reason is that in our C++ program, we adopted 5×5 fuzzy rule sets while in the hardware implementation, for the purpose of reducing hardware complexity, we adopted 3×3 fuzzy rule sets. This difference can affect the system performance significantly. Fig. 8.16 shows the evolution process of the *bestchrom* which is configured into the RFIC as in Fig. 8.1 to serve as a working chromosome. Each change indicates that a new working chromosome is evolved.

The simulation results for $\alpha=0.5$ are presented in Fig. 8.17 and Fig. 8.18. The final

AveDelay in Fig. 8.17 is “27d9” in Hex format indicating a 102.01 microseconds average delay. The average delay based on the C++ simulation program is 127.59 microseconds. The difference between these two values is also mainly due to the difference in the number of fuzzy rules. A snapshot of the evolution process of the working chromosome is as shown in Fig. 8.18.

The synthesis is carried out using Synopsys Design Compiler. The target library is TSMC 0.18 standard cell library. The chip operating frequency is 200MHz. The total area of the EFH chip is about 2 million gates which is much less than that of state-of-the-art microprocessor. This indicates that the whole SoC circuit can be easily realized as an ASIC chip. This is the major purpose and contribution of this Chapter.

Figure 8.15: Simulation results of GA block for $\alpha=0.3$

Figure 8.16: Snapshot of evolution process for $\alpha=0.3$

Figure 8.17: Simulation results of GA block for $\alpha=0.5$

Figure 8.18: Snapshot of evolution process for $\alpha=0.5$

Chapter 9

Conclusions and future directions

According to literature survey, there are several challenges to the application of evolvable hardware for solving time critical problems. To our knowledge, no substantial research work has been able to address them effectively to date. We highlighted three issues, namely *online adaptation*, *scalability* as well as *termination of evolution*. To realize EHW capable of intrinsic online evolution, these issues have to be considered. In this thesis, we proposed the EFH scheme, a form of EHW whereby the fuzzy inference scheme is carried out in hardware to achieve real-time operation. The scheme allows for updating of on-line context and domain rules and further incorporating mechanisms to evolve a context appropriate for the application scenario. In order to demonstrate the viability of our proposed EFH, we simulated the control performance of the EFH in cell scheduling and compared the results with some traditional scheduling methods for two different situations: two-input channel cell scheduling and multi-channel cell scheduling.

In the first instance, in order to test the system's controllability and tunability, we considered two scenarios of cell flow traffic. From the simulation results, it is observed that EFH exhibits good controllability and tunability. In order to test the performance of EFH for congestion control in a more realistic traffic pattern, we simulated its performance on a third scenario. Three different λ values satisfying various QoS performance requirements were chosen to further emphasize on the EFH's tunability. The response curves show that EFH is able to control the ATM cell scheduling for the third scenario in a desirable manner. For the purpose of demonstrating the advantages of EFH, we compared EFH with other scheduling schemes such as FIFO and DWPS. In terms of flexibility, EFH is better than the other schemes because it is able to realize desirable $class_1$ delay and cell loss balance characteristics through parametric adjustment of the fitness function. EFH also exhibits very good adaptation, even for drastic changes in cell flow scenarios. It adapts to the changing scenarios very well and maintains similar level of QoS performance. The overall simulation results demonstrated that the EFH not only has good ability of tracking the changes of cell flow pattern, but also exhibits good performance tunability. These two advantages substantiate the EFH's suitability for applications in packet switching networks.

The multiplexing problem in the two-input context represents a particular case of cell scheduling. There are other related cases pertaining to cell scheduling. For example, more than two cell flow inputs may be multiplexed to one output channel. All the input cell flows are of the same classification, for example, constant bit rate flow and

each cell flow has its reserved bit rate. Due to the unpredictable network conditions, some may misbehave. The scheduler should be capable of isolating the QoS of the good performing cell flow channels from being influenced by the non-performing ones. In a more complicated case, the packet size may be variable. For scheduling in this kind of network, conventional schemes may not be flexible enough to satisfy the different QoS requirements. Their inability in adapting to changing cell flow patterns is their main weakness. The EFH scheme proposed is intrinsically evolvable and possesses on-line adaptation capability. It can be extended to solve such complicated scheduling scenarios. In order to extend and further explore the application of EFH, we have also presented EFH as a new traffic control scheme for multi-channel packet-switching networks. By incorporating evolutionary algorithm and the fuzzy control system, it can be regarded as a different approach that is independent of the GPS approach and RR approach. Our analysis of the system computation complexity indicated that EFH is applicable for real-time adaptation and evolution, EFH was simulated on three different input scenarios. In the first scenario, all the three schemes had similar performance as that of GPS in terms of bandwidth allocation. However, when the cell delay QoS was taken into consideration, the performance of EFH was much better than the other two. In the second scenario and the third scenario, EFH showed very good average delay for the good performing flows while MCRR and FIFO achieved much bigger average delay. The bandwidth allocation ratio achieved by EFH was close to that of GPS in terms of mean and deviation. These simulations demonstrated that without adopting the fluid model and RR schemes, the QoS performance for cell scheduling very close to that of

GPS can also be achieved by EFH. Besides effective management of delay and bandwidth allocation, another desirable property of EFH is its QoS tunability as in the two input channel application. This was also demonstrated through simulations on various cell flow scenarios. If different average delay is desired for the good performing flows, the corresponding parameter in the fitness function can be tuned accordingly. In general, larger average delay can be achieved by increasing the corresponding parametric value.

In order to demonstrate the possibility of realizing the whole EFH as a single chip using state-of-the-art SoC technology, we also described the implementation of the whole EFH in this thesis. The hardware implementation of the EFH system includes the realization of GA, RFIC and the *evaluation cell*, the key components of the EFH system. One significant aspect of this EFH is that it does not need an external computation platform as compared to other EHW systems. Another significant aspect of this EFH is that it does not have an interruption for evolution. The simulation and the synthesis results indicate that the designed circuit satisfies the functional and physical requirements of the application domain.

Based on the research work reported in this thesis, EFH can stand on its own as a useful technique to handle dynamic switching applications. Although EFH demonstrated strong performance based on simulations, it is still premature for deployment in real-time applications. More research on the EFH needs to be carried out, in particular, the evolution time needs to be further improved. For the purpose of reducing the computation time and advancing the system performance, further

study on the hardware architecture and hardware implementation of the evaluation process and GA operations – the crucial parts of the EFH, is required.

For the purpose of further extending the application of EFH, other research direction of EFH may be the application of the EFH system on image processing, pattern recognition and signal processing, or other dynamic real-time applications.

Chapter 10

Author's Publications

10.1 Journals and Book Chapters

1. J.H. Li, M.H. Lim, and Q. Cao, “Evolvable fuzzy hardware for real-time embedded control for packet-switching,” *Evolvable Machines: Theory & Practice*, N. Nedjah, L. de Macedo Mourelle (Eds), vol. 161, pp. 205–227, 2004.
2. J.H. Li, M.H. Lim and Q. Cao, “A QoS-tunable scheme for atm cell scheduling using evolutionary fuzzy system,” *Applied Intelligence*, vol. 23, no. 3, pp. 207–218, 2005.
3. M.H. Lim, J.H. Li and Q. Cao, “On-line evolving fuzzy hardware system for ATM cell-scheduling”, *Journal of Systems Architecture*, vol. 52, no. 3, pp. 169–183, 2006.
4. M.H. Lim, Q. Cao, J.H. Li, and W.L. Ng, “Evolvable hardware using con-

text switchable fuzzy inference processor,” *IEE Proc. Comput. Digit. Tech.*, vol. 151, no. 4, pp. 301–311, 2004.

5. Q. Cao, M.H. Lim, J.H. Li, Y.S. Ong, and W.L. Ng, ”A Context Switchable Fuzzy Inference Chip,” to appear in *IEEE Transactions on Fuzzy Systems*.

10.2 Conferences

1. J.H. Li and M.H. Lim, “A framework for evolvable fuzzy hardware,” in *Special session on Evolutionary Computing for Control and System Applications, ICONIP/SEAL/FSKD Joint Conference*, pp. 18–22, 2002.
2. J.H. Li and M.H. Lim, “Evolvable fuzzy system for ATM cell scheduling,” in *ICES’03 Conference*, pp. 17–20, 2003.
3. J.H. Li, M.H. Lim and Q. Cao, “An evolvable fuzzy hardware scheme for packet - switching network,” in *Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology*, Aug. 2004.
4. J.H. Li, M.H. Lim and Q. Cao, “An intrinsic evolvable and online adaptive evolvable fuzzy hardware scheme for packet switching network,” in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2004*, pp. 109–112, 2004.
5. Q. Cao, M.H. Lim and J.H. Li, “A novel aggregation circuit for reconfigurable fuzzy inference chip,” in *Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology*, Aug. 2004.

6. Q. Cao, M.H. Lim, K.V. Ling, C.W. Chan and J.H. Li, “A digital prototyping realization for a context switchable reconfigurable fuzzy inference chip,” in *Proceedings of the 10th International Symposium on Integrated Circuits, Devices & Systems*, 2004.

10.3 Statements

1. Among the above papers, some are my major contribution and whose content is included in this thesis. They are the first three papers in the Journal part and the first four papers in the Conference part. The others are my junior Cao Qi’s major contribution.
2. In this thesis, RFIC is simply described in Section 4.2.4 for the purpose of completeness. It is majorly the controbution of my junior Cao Qi. I laterly modified the architecture of RFIC by myself for my own use according to my system’s characteristics and requirements. The modified RFIC is implemented by Verilog HDL language and the whole EFH is simulated for the functional prove.

Here I would like to express my appreciations to my junior Cao Qi for his support on the description and his consultation service on the implementation of the modified RFIC.

10.4 Others

The paper “Evolvable Fuzzy Hardware for Real-Time Embedded Control for Packet-Switching” was accepted and presented as an entry of the competition of “2004 Human-Competitive Results Awards competition” at the 2004 Genetic and Evolutionary Computation Conference (GECCO-2004), June 26-30, 2004, Seattle, Washington, USA. This paper finally got Honorable Mention Award.

Chapter 11

Symbols and Acronyms

ACO - Ant Colony Optimization

ATM - asynchronous transmission mode

CBR - Constant Bit Rate

DWPS - dynamically weighted priority scheduling

EFH - evolvable fuzzy hardware

EHW - evolvable hardware

EFS - evolvable fuzzy system

FIFO - first-in-first-out

FPGA - field programmable gate array

FPTA - field programmable transistor array

FPU - functional processing units

FSM - finite state machine

GAs - genetic algorithms

GPS - generalized processor sharing

HDL - hardware description languages

MBS - maximum burst size

MCRR - minimized cycle round robin

NN - Neural network

PC - personal computer

PCR - peak cell rate

PLA - programmable logic array

QoS - Quality of Service

RFIC - reconfigurable fuzzy inference chip

RR - round robin

SFQ - start-time fair queuing

SFCQ - self-clocked fair queuing

SoC - system on chip

SPR - static priority

VBR - Variable Bit Rate

WFQ - weighted fair queuing

WF2Q - worst-case fair weighted fair queuing

WRR - weighted round robin

Bibliography

- [1] L. Prodan, G. Tempesti, D. Mange, and A. Stauffer, “Biology meets electronics: the path to a bio-inspired FPGA,” in *Proc. of the Third International Conference on Evolvable Systems: From Biology to Hardware, (ICES 2000), LNCS 1801*, pp. 187–196, 2000.
- [2] D. Mange, E. Sanchez, A. Stauffer, G. Tempesti, P. Marchal, and C. Piguet, “Embryonics: A new methodology for designing field-programmable gate arrays with self-repair and self-replicating properties,” *IEEE Transactions on VLSI Systems*, vol. 6, no. 3, pp. 387–399, 1998.
- [3] C. Ortega and A. Tyrrell, “Reliability analysis in self-repairing embryonic systems,” in *Proc. of the First NASA/DoD Workshop on Evolvable Hardware*, pp. 120–128, 1999.
- [4] M. Sipper, “Fifty years of research on self-replication: an overview,” *Artificial Life*, vol. 4, no. 3, pp. 237–257, 1998.
- [5] J. Torresen, “Evolvable hardware - the coming hardware design method?,” *book: Neuro-fuzzy techniques for Intelligent Information Systems*, N. Kasabov,

- R. Kozma (Eds), pp. 435–449, 1999.
- [6] J. Torresen, “Evolvable hardware - a short introduction,” in *International Conference On Neural Information Processing (ICONIP’97)*, 1997.
- [7] M. H. Lim, S. Rahardja, and B. H. Gwee, “A GA paradigm for learning fuzzy rules,” *Fuzzy Sets and Systems*, vol. 82, pp. 177–186, 1996.
- [8] W. Liu, M. Murakawa, and T. Higuchi, “ATM cell scheduling by function level evolvable hardware,” in *LNCS 1259 (ICES1996)*, pp. 180–192, 1996.
- [9] T. G. W. Gordon and P. J. Bentley, “On evolvable hardware,” in *Soft Computing in Industrial Electronics*, pp. 279–323, 2002.
- [10] H. D. Garis, *Evolvable Hardware: Principles and Practice*.
<http://www.cs.usu.edu/~degarris/papers/CACM-E-Hard.html>.
- [11] M. Iwata, I. Kajitani, H. Yamada, H. Iba, and T. Higuchi, “A pattern recognition system using evolvable hardware,” in *Proc. Int. Conf. Parallel Probl. Solving Nature*, pp. 761–770, 1996.
- [12] E. Sanchez, *Towards evolvable hardware: the evolutionary engineering approach*. New York: Springer, 1996.
- [13] N. Homma, T. Aoki, and T. Higuchi, “Multiplier block synthesis using evolutionary graph generation,” in *Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware*, pp. 79–82, 2004.

- [14] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Transactions on Syst. Man and Cybern. PartC, Applications and Reviews*, vol. 29, no. 1, pp. 87–97, 1999.
- [15] J. Torresen, "Possibilities and limitations of applying evolvable hardware to real-world application," in *proc. of 10th International Conference on Field Programmable Logic and Applications (FPL-2000)*, 2000.
- [16] K. C. Tan, C. M. Chew, K. K. Tan, L. F. Wang, and Y. J. Chen, "Autonomous robot navigation via intrinsic evolution," in *Proc. of the Congress on Evolutionary Computation, 2002*, vol. 2, pp. 1272–1277, 2002. CEC'02.
- [17] J. Langeheine, K. Meier, and J. Schemmel, "Intrinsic evolution of quasi dc solutions for transistor level analog electronic circuits using a CMOS FPTA chip," in *Proc. NASA/DoD Conference on Evolvable Hardware*, pp. 75–84, 2002.
- [18] F. H. Bennett, J. R. Koza, M. A. Keane, J. Yu, W. Mydlowee, and O. Stiffelman, "Evolution by means of genetic programming of analog circuits that perform digital functions," in *Proc. of the Genetic and Evolutionary Computation Conference*, pp. 1477–1483, 1999.
- [19] J. Langeheine, S. Folling, K. Meier, and J. Schemmel, "A CMOS FPTA chip for intrinsic hardware evolution of analog circuits," in *Proc. of the Third NASA/DOD Workshop on Evolvable Hardware*, pp. 172–175, Jul 2001.
- [20] A. Stoica, D. Keymeulen, R. Zebulum, A. Thakoor, T. Daud, Y. Klimeck, R. Tawel, and V. Duong, "Evolution of analog circuits on field programmable

- transistor arrays,” in *Proceedings of The Second NASA/DoD Workshop on Evolvable Hardware*, pp. 99–108, 2000.
- [21] A. Stoica, R. Zebulum, D. Keymeulen, R. Tawel, T. Daud, and A. Thakoor, “Reconfigurable VLSI architectures for evolvable hardware: From experimental field programmable transistor arrays to evolution-oriented chips,” *IEEE Trans. on very large scale integration (VLSI) systems*, vol. 9, no. 1, pp. 227–231, 2001.
- [22] T. Higuchi, M. Iwata, I. Kajitani, M. Murakawa, S. Yoshizawa, and T. Furuya, “Hardware evolution at gate and function levels,” in *Proc. Biologically Inspired Autonomous Systems: Computation, Cognition and Action*, 1996.
- [23] D. Keymeulen, K. Konada, M. Iwata, Y. Kuniyoshi, and T. Higuchi, “Robot learning using gate-level evolvable hardware,” in *Proc. of the Sixth European Workshop on Learning Robots, Lecture Notes in Artificial Intelligence*, 1998.
- [24] M. Iwata, I. Kajitani, Y. Liu, N. Kajihara, and T. Higuchi, “Implementation of a gate-level evolvable hardware chip,” in *LNCS 2210 (ICES2001)*, pp. 38–49, 2001.
- [25] H. Iba, M. Iwata, and T. Higuchi, “Gate-level evolvable hardware: Empirical study and application,” in *Evolutionary Algorithms in Engineering Applications*, pp. 260–275, 1997.
- [26] I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iwata, D. Keymeulen, and T. Higuchi, “A gate-level

- EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI,” in *LNCS 1478 (ICES1998)*, pp. 1–12, 1998.
- [27] H. Iba, M. Iwata, and T. Higuchi, “Machine learning approach to gate-level evolvable hardware,” in *Evolvable Systems: From Biology to Hardware (ICES1996)*, *LNCS 1259*, pp. 327–343, 1997.
- [28] D. Keymeulen, K. Konaka, M. Iwata, Y. Kuniyoshi, and T. Higuchi, “Robot learning using gate-level evolvable hardware,” in *Proc. of Sixth European Workshop on Learning Robots (EWLR-6)*, 1997.
- [29] T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. X. Liu, and M. Salami, “Evolvable hardware at function level,” in *Proc. of 1997 IEEE Int. Conf. on Evolutionary Computation (ICEC97)*, pp. 187–192, 1997.
- [30] M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata, and T. Higuchi, “Hardware evolution at function level,” in *Parallel Problem Solving from Nature-PPSN IV*, *LNCS 1141*, pp. 62–71, 1996.
- [31] M. H. Lim, Q. Cao, J. H. Li, and W. L. Ng, “Evolvable hardware using context switchable fuzzy inference processor,” *IEE Proc. Comput. Digit. Tech.*, vol. 151, no. 4, pp. 301–311, 2004.
- [32] Q. Cao, M. H. Lim, and J. H. Li, “A novel aggregation circuit for reconfigurable fuzzy inference chip,” in *Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology*, 2004.

- [33] Q. Cao, M. H. Lim, K. V. Ling, C. W. Chan, and J. H. Li, "A digital prototyping realization for a context switchable reconfigurable fuzzy inference chip," in *Proceedings of the 10th International Symposium on Integrated Circuits, Devices & Systems*, 2004.
- [34] T. C. Fogarty, J. F. Miller, and P. Thompson, "Evolving digital logic circuits on Xilinx 6000 family FPGAs," in P.K. Chawdry, R. Roy, R.K. Pant (Eds). *Soft Computing in Engineering Design and Manufacturing*, (London), pp. 299–305, Springer-Verlag, 1998.
- [35] C. Slorach and K. Sharman, "The design and implementation of custom architectures for evolvable hardware using off-the-shelf programmable devices," in *Evolvable Systems: From Biology to Hardware (ICES1996)*, pp. 197–207, 2000.
- [36] J. Torresen, "Increased complexity evolution applied to evolvable hardware," in *ANNIE'99*, 1999.
- [37] J. Torresen, "Exploring knowledge schemes for efficient evolution of hardware," in *Proceedings of 2004 NASA/DoD Conference on Evolvable Hardware*, pp. 209–216, 2004.
- [38] J. Torresen, "A scalable approach to evolvable hardware," *Genetic Programming and Evolvable Machines*, vol. 3, no. 3, pp. 259–282, 2002.
- [39] J. Torresen, "Open-ended evolution with or without a priori knowledge included?," in *Proc. of GECCO Workshop on Modularity, Regularity, and Hierarchy in Evolutionary Computation*, 2004.

- [40] J. Torresen, "Evolving multiplier circuits by training set and training vector partitioning," in *proc. of Fifth International Conference on Evolvable Hardware (ICES03)*, pp. 228–237, 2003.
- [41] J. Torresen, "Two-step incremental evolution of a prosthetic hand controller based on digital logic gates," in *proc. of 4th International Conference on Evolvable Hardware (ICES2001)*, 2001.
- [42] J. Torresen, "A divide-and-conquer approach to evolvable hardware," in *proc. of 2nd International Conference on Evolvable Hardware (ICES98)*, pp. 57–65, 1998.
- [43] J. H. Li, M. H. Lim, and Q. Cao, "A QoS-tunable scheme for atm cell scheduling using evolutionary fuzzy system," *Applied Intelligence*, vol. 23, no. 3, pp. 207–218, 2005.
- [44] J. H. Li and M. H. Lim, "Evolvable fuzzy system for atm cell scheduling," in *Proc. of 5th Int. Conf. Evolvable Syst.: From Biology to Hardware (ICES 2003) LNCS 2606*, pp. 208–217, 2003.
- [45] J. H. Li, M. H. Lim, and Q. Cao, "Evolvable fuzzy hardware for real-time embedded control for packet-switching," *Evolvable Machines: Theory & Practice*, N. Nedjah, L. de Macedo Mourelle (Eds), vol. 161, pp. 205–227, 2004.
- [46] J. H. Li, M. H. Lim, and Q. Cao, "An intrinsic evolvable and online adaptive evolvable fuzzy hardware scheme for packet switching network," in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2004*, pp. 109–112, 2004.

- [47] J. H. Li, M. H. Lim, and Q. Cao, "An evolvable fuzzy hardware scheme for packet - switching network," in *Proceedings of the 2nd International Conference on Artificial Intelligence in Engineering and Technology*, 2004.
- [48] Y. S. Shi, R. Eberhart, and Y. C. Chen, "Implementation of evolutionary fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, pp. 109–119, Apr 1999.
- [49] C. H. Wang, T. P. Hong, and S. F. Tseng, "Integrating fuzzy knowledge by genetic algorithms," *IEEE Transactions on Evolutionary Computation*, vol. 2, no. 4, pp. 138–149, 1998.
- [50] J. M. Jou, P. Y. Chen, and S. F. Yang, "Integrating fuzzy knowledge by genetic algorithms," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, pp. 52–60, Feb 2000.
- [51] M. D. Bugajska and A. C. Schultz, "Coevolution of form and function in the design of micro air vehicles," in *Proc. NASA/DoD Conference on Evolvable Hardware, 2002*, pp. 154–163, 2002.
- [52] M. Salami, H. Sakanashi, M. Tanaka, M. Iwata, T. Kurita, and T. Higuchi, "On-line compression of high precision printer images by evolvable hardware," in *Proc. DCC'98 Data Compression Conference, IEEE Computer Society Press*, pp. 219–228, 1998.
- [53] Y. Liu, M. Iwata, T. Higuchi, and D. Keymeulen, "An integrated on-line learning system for evolving programmable logic array controllers," in *Parallel Prob-*

- lem Solving from Nature – PPSN VI: Proc. of the Sixth International Conference on Parallel Problem Solving from Nature*, pp. 589–598, 2000.
- [54] ATM Forum, “ATM traffic management specification 4.0,” in *ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.pdf*, 1996.
- [55] E. P. Rathgeb, “Modeling and performance comparison of policing mechanisms for ATM networks,” *IEEE Journal on Selected Areas of Communication*, vol. 9, pp. 325–334, Apr 1991.
- [56] L. Zhang, “Virtual clock: A new traffic control algorithm for packet switched network,” *ACM Trans. Comput. Syst.*, vol. 9, no. 2, pp. 101–124, 1991.
- [57] T. Lizambri, F. Duran, and S. Wakid, “Priority scheduling and buffer management for atm traffic shaping,” in *Proc. of 7th IEEE Workshop on Future Trends of Distributed Computing Systems, FTDCS’99*, pp. 36–43, 1999.
- [58] B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson, and J. D. Robbins, “Performance models of statistical multiplexing in packet video communications,” *IEEE transaction on Communication*, vol. 36, pp. 834–844, Jul 1988.
- [59] R. Guerin, H. Ahmadi, and M. Naghshineh, “Equivalent capacity and its application to bandwidth allocation in high-speed networks,” *IEEE Journal on Selected Areas of Communication*, vol. 9, pp. 968–981, Sep 1991.
- [60] M. P. Clark, *ATM networks: principles and use*. John Wiley, 1996.

- [61] A. K. Parekh and R. G. Gallager, "A generalized processor sharing approach to flow control in integrated services networks: The single-node case," *IEEE/ACM Trans. Networking*, vol. 1, no. 3, pp. 344–357, 1993.
- [62] Y. Liang, "A simple and effective scheduling mechanism using minimized cycle round robin," in *In Proceedings of IEEE International Conference on Communications*, vol. 4, pp. 2384–2388, 2002.
- [63] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queuing algorithm," *Journal of Internetworking Research and Experience*, pp. 3–26, 1990.
- [64] J. C. R. Bennett and H. Zhang, "Wf2q: Worst-case fair weighted fair queuing," in *Proc. IEEE INFOCOM, 1996*, pp. 120–128, 1996.
- [65] P. Goyal, H. M. Vin, and H. Cheng, "Start-time fair queuing: A scheduling algorithm for integrated services packet switching networks," *IEEE/ACM Trans. Networking*, vol. 5, no. 5, pp. 157–168, 1997.
- [66] J. Davin and A. Heybey, "A simulation study of fair queuing and policy enforcement," *Computer commun. Rev.*, vol. 20, no. 5, pp. 23–29, 1990.
- [67] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Trans. on Networking*, vol. 4, no. 3, pp. 375–385, 1996.
- [68] H. M. Chaskar and U. Madhow, "Fair scheduling with tunable latency: A round-robin approach," *IEEE/ACM Trans. Networking*, vol. 11, no. 4, pp. 592–601, 2003.

- [69] A. Hossain and S. M. Kang, "Fuzzy logic based cell scheduling for input-buffered ATM switches," in *SPIE Proc. Photonics East '95 Conf.*, vol. 2608, pp. 117–121, 1995.
- [70] R. Zhang, Y. A. Phillis, and J. Man, "A fuzzy approach to the balance of drop and delay priorities in differentiated services networks," *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 6, pp. 840–846, 2003.
- [71] M. H. Lim and W. L. Ng, "Iterative genetic algorithm for learning efficient fuzzy rule set," *AIEDAM*, vol. 17, no. 4, pp. 335–347, 2004.
- [72] M. H. Lim, Y. Yuan, and S. Omatu, "Efficient genetic algorithms using simple genes exchange local search policy for the quadratic assignment problem," *Computational Optimization and Applications*, vol. 15, no. 3, pp. 249–268, 2000.
- [73] M. H. Lim, Y. Yuan, and S. Omatu, "Extensive testing of a hybrid genetic algorithms for solving quadratic assignment problems," *Computational Optimization and Applications*, vol. 23, no. 1, pp. 47–64, 2002.
- [74] B. Kosko, *Neural Networks and Fuzzy Systems*. Prentice Hall, Englewood Cliffs, NJ, 1992.
- [75] W. Liu, M. Murakawa, and T. Higuchi, "Evolvable hardware for on-line adaptive traffic control in ATM networks," in *Genetic Programming 1997, Proc. of the Second Annual Conference*, pp. 504–509, 1997.

- [76] R. Jain, “Congestion control and traffic management in atm networks: Recent advances and a survey,” *Computer Networks and ISDN Systems*, vol. 28, pp. 1723–1738, Oct 1996.
- [77] “<http://www.isi.edu/nsnam/ns/doc/ns-doc.pdf>,” 2006.
- [78] R. Jain and S. Routhier, “Packet trains - measurements and a new model for computer network traffic,” *IEEE Journal on Selected Areas in Communications*, vol. 4, no. 6, pp. 986–995, 1986.
- [79] R. Händel, M. Huber, and S. Schröder, *ATM networks: concepts, protocols, applications*. Addison-Wesley, 1998.
- [80] P. H. Bardel, W. H. McAnney, and J. Savir, *Built In Test for VLSI: Pseudo-random Techniques*. John Wiley & Sons, 1987.