Design and implementation of a reconfigurable fuzzy inference processor

Cao, Qi

2006

Cao, Q. (2006). Design and implementation of a reconfigurable fuzzy inference processor.
Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/4935>

<https://doi.org/10.32657/10356/4935>

Nanyang Technological University

Downloaded on 26 Apr 2023 18:02:17 SGT

(On the Spine)

**NANYANG
TECHNOLOGICAL
UNIVERSITY**

**DESIGN AND IMPLEMENTATION OF A
RECONFIGURABLE FUZZY INFERENCE
PROCESSOR**

DESIGN AND IMPLEMENTATION OF A
RECONFIGURABLE FUZZY INFERENCE PROCESSOR

CAO QI

CAO QI

SCHOOL OF ELECTRICAL & ELECTRONIC ENGINEERING

2006

2006

Design and Implementation of A Reconfigurable Fuzzy Inference Processor

Cao Qi

School of Electrical & Electronic Engineering

A thesis submitted to the Nanyang Technological University
in fulfilment of the requirement for the degree of
Doctor of Philosophy

2006

Statement of Originality

I hereby certify that the work presented in this thesis is the result of original research and has not been submitted for a degree to any other university or institution.

Date

Cao Qi

ACKNOWLEDGMENTS

I would like to thank many people who have supported me during my study in Nanyang Technological University. First and foremost, I would like to express my sincere gratitude and appreciation to my supervisor, Assoc. Prof. Lim Meng Hiot, for his professional guidance and support, for his invaluable advice and encouragement throughout my Ph.D. program. I benefit a lot from his supervision over the years. I really appreciate his style of supervision and his patience to his students.

I would like to appreciate Assoc. Prof. Ling Keck Voon, for his apparatus support and helpful discussions in my project. I also appreciate Mr. Sia Liang Poo, Mr. Jimmy Goh Mia Yong and Mr. William Seow Yang Hing, Ms. Chan Nai Hong, Ms. Leong-Tan Min Lin, Ms. Lim-Tan Gek Eng and Mr. Kang Meng Fai for their help in solving many software and hardware technical problems.

I am grateful to my seniors, Mr. Li Ju Hui and Ng Wil Lie, as well as other students in our research group, who have given me the encouragements and help that I need.

I would like to express my truly appreciations to my loving girl-friend, Ms. Shi Xiao Meng for her endless supports and encouragements.

Finally, I want to thank my parents, my girl-friend's parents, my elder sisters and their families for their deep love and encouragements.

Abstract

To achieve real-time performance for some applications of fuzzy systems, it is necessary to realize the inference module in hardware. By the same token, real time applications of *evolvable fuzzy hardware* (EFH) systems require the realization of systems as hardware that can be conveniently reconfigured. A good example of such a scenario is illustrated for the case of *Asynchronous Transfer Mode* (ATM) cell scheduling [1]. Since the domain rule set is expected to evolve during normal operations, it is imperative that the hardware supporting the fuzzy reasoning should be easily reconfigurable. To address this requirement, a *reconfigurable fuzzy inference processor* (RcFIP) that supports online rule set reconfigurability is proposed in this project. The overall methodology of designing EFH systems based on the RcFIP as the core inference processor is presented in this thesis. The concept of a RcFIP-based intrinsic EFH can be viewed as a type of *evolvable hardware* (EHW). The RcFIP is an in-system context-switchable inference processor for evolvable fuzzy hardware. A context refers to a situation or scenario of an application requiring specific domain knowledge. In particular, the focus is on the class of applications involving embedded fuzzy control. The domain knowledge therefore refers to fuzzy rules and membership functions. The kind of applications being considered is real time in nature, which necessitates the implementation of the fuzzy inference in hardware. The chip architecture is described and details on the design of the chip is presented in this thesis.

The proposed RcFIP which achieves 17M *fuzzy logic inference per second* (FLIPS) is able to offer high execution speed for time-critical applications. It is a good candidate to develop a *multi-task fuzzy inference processor* (MtFIP) that is capable of performing multiple tasks or applications concurrently. The development of a MtFIP is another contribution of this project. The detailed hardware architecture of the MtFIP is addressed in this thesis.

Acronyms List in this Thesis

The following acronyms are employed in the thesis.

ADC	Analog-to-Digital Converter
AEM	Address Encoder Module
ASIC	Application Specific Integrated Circuits
ATM	Asynchronous Transfer Mode
CMU	Context Management Unit
COA	Center of Area
DAC	Digital-to-Analog Converter
DWPS	Dynamically Weighted Priority Scheduling
EDA	Electronic Design Automation
EDIF	Electronic Device Interchange Format
EFH	Evolvable Fuzzy Hardware
EHW	Evolvable Hardware
FIFO	First-In-First-Out
FIM	Fuzzy Inference Mapping
FLIPS	Fuzzy Logic Inference per Second

FPGA	Field Programmable Gate Array
FSC	Fuzzy Switching Control
GA	Genetic Algorithms
IC	Integrated Circuits
LSB	Least Significant Bit
MOM	Mean of Maximum
MSB	Most Significant Bit
MtFIP	Multi-task Fuzzy Inference Processor
MUX	Multiplexer
OAM	Output Aggregation Module
PB	Partition Block
PLD	Programmable Logic Device
PWM	Pulse Width Modulation
QoS	Quality of Service
RAM	Random Access Memory
RcFIP	Reconfigurable Fuzzy Inference Processor
RISC	Reduced Instruction Set Computing
ROM	Read Only Memory

Acronyms List in this Thesis

v

RTL	Register-Transistor Level
SOC	System-on-Chip
VHDL	Very High Speed Hardware Description Language

Table of Contents

Abstract	i
Acronyms List	iii
List of Tables	xi
List of Figures	xv
1 Introduction	1
1.1 Evolvable Fuzzy Hardware	1
1.2 Fuzzy Inference Processor	5
1.2.1 Fuzzy Inference Overview	5
1.2.2 Fuzzy Inference Hardware Implementation	9
1.3 Multi-Task Processor	12
1.4 Motivations	15

TABLE OF CONTENTS vii

1.5	Objective	19
1.6	Contributions	21
1.7	Organization of Report	22
2	Fuzzy Inference Processors Review	23
3	EFH Developmental Framework	30
3.1	GA Overview	31
3.2	Genetic Coding	32
3.3	Developmental Framework	34
4	RcFIP Hardware Architecture	38
4.1	RcFIP Architecture	38
4.1.1	FIM	42
4.1.2	CMU	44
4.1.3	AEM	46
4.1.4	OAM	47
4.1.5	Control Unit	54
4.1.6	Circuit Prototyping	56
4.1.7	Evolvable Hardware System	62

TABLE OF CONTENTS

viii

4.2	Configuring FIM	63
4.2.1	General Descriptions	63
4.2.2	Simple Illustration	66
4.3	Potential Tradeoff of RcFIP	70
5	Application Examples	73
5.1	Water Bath Temperature Control	75
5.1.1	Functional Simulation	79
5.2	RcFIP-Based Rotary Inverted Pendulum	81
5.2.1	Overall System Setup	81
5.2.2	ADC Controller	84
5.2.3	Encoder controller	86
5.2.4	PWM controller	89
5.2.5	Hardware Connection Interface	90
5.2.6	Functional Demonstration	91
5.3	RcFIP for ATM Cell Scheduling	94
5.3.1	Simulation results and comparisons	96
6	Multi-Task Fuzzy Inference Processor	102

TABLE OF CONTENTS

ix

6.1	Detailed Architecture of MtFIP	105
6.1.1	FIM	109
6.1.2	Input Block	111
6.1.3	Context Block	112
6.1.4	Output Block	113
6.1.5	Task Manager	114
6.1.6	Control Unit	118
6.2	MtFIP Prototyping and Analysis	119
6.2.1	Situation 1	120
6.2.2	Situation 2	123
6.2.3	Situation 3	124
6.2.4	Inference performance of the MtFIP	124
6.3	MtFIP Applications and Simulations	126
6.3.1	Domain Knowledge	127
6.3.2	MtFIP Simulation Results	129
7	Conclusions	132
7.1	Conclusions	132

TABLE OF CONTENTS

x

7.2	Future Work	135
8	Author's Publications	136
8.1	Journals and Book Chapters	136
8.2	Conferences	137
8.3	Appendix	138
8.4	Special Acknowledgement on this Project	139
	Bibliography	140

List of Tables

4.1	RcFIP circuitry scale reports	57
4.2	RcFIP specifications and comparisons	61
4.3	Data Sets for Configuring FIM Partition Blocks	70
5.1	Configuration Data Set for Grouping<1,1> of the FIM	77
5.2	Determination of arm rotary direction	87
5.3	Three different operating scenarios	93
6.1	MtFIP circuitry scale reports	119
6.2	MtFIP specifications and comparisons	125

List of Figures

1.1	Membership functions for a simple example	6
1.2	Mamdani implication and max-min composition procedure	7
1.3	General block architecture of fuzzy inference processors	10
2.1	Classifications of fuzzy inference processors	24
3.1	One-point crossover	32
3.2	One-point mutation	32
3.3	Genetic coding for a 14-rule context	33
3.4	Developmental framework of the RcFIP-based EFH	35
4.1	Block Architecture of RcFIP	39
4.2	FIM Configuration	43
4.3	Working flowchart of CMU	45

LIST OF FIGURES

xiii

4.4	Hardware architecture of CMU	46
4.5	Schematic of AEM	47
4.6	Hardware architecture of OAM	50
4.7	Flowchart of <i>Avg_2</i> Module	51
4.8	RTL Schematic View for <i>Avg_2</i> Module	52
4.9	Overall Schematic for <i>Avg_3</i> Module	53
4.10	Hardware Block Architecture of Control Unit	55
4.11	RcFIP prototype realization flowchart	56
4.12	Waveforms for RcFIP with new inputs update (Situation 1)	58
4.13	Waveforms for RcFIP with context switch (Situation 2)	59
4.14	Waveforms for both new inputs and context updates (Situation 3)	60
4.15	Functional block diagram for complete system	62
4.16	Linguistic matrix of maximum rule set for two input variables	64
4.17	Linguistic matrices of the $(v \times w)$ groupings	65
4.18	Membership functions of a simplified system	67
4.19	Coverage map of the 9 groupings for configuring the FIM	72
5.1	Overall setup of water bath temperature control system	75

LIST OF FIGURES

xiv

5.2	Membership functions for the water bath temperature controller . . .	76
5.3	Four examples of contexts for the RcFIP	78
5.4	Control responses for heating from 25° to 80°	80
5.5	Overall setup of RcFIP-based system	82
5.6	Diagram of ADC controller	85
5.7	Flowchart for ADC controller circuit	86
5.8	Flowchart for encoder controller circuit	89
5.9	PWM controller circuit	90
5.10	Hardware connection between two boards	91
5.11	Membership Functions for rotary inverted pendulum	92
5.12	Three contexts examples for the RcFIP	92
5.13	Control responses for RcFIP-based system	94
5.14	Architectural framework of EFH for ATM cell scheduling	95
5.15	Cell flow in a realistic traffic scenario	97
5.16	QoS performance for <i>class1</i> and <i>class2</i>	99
6.1	RcFIP Brief Structure	103
6.2	Diagram of Multi-Tasking Architectural Framework	103

LIST OF FIGURES

xv

6.3	Overall Architecture of MtFIP	105
6.4	MtFIP's FIM Configuration	109
6.5	Schematic of the Input Block	111
6.6	Schematic of the Context Block	112
6.7	Schematic of the Output Block	113
6.8	Architecture of the Task Manager	114
6.9	Working flow chart of Task Manager	116
6.10	Waveforms for inputs concurrent updates of 3 tasks (Situation 1) . .	121
6.11	Waveforms for inputs updates of a single task (Situation 2)	122
6.12	Waveforms for inputs and contexts updates of 3 tasks (Situation 3) .	123
6.13	Membership Functions for Task_1	127
6.14	Membership Functions for Task_2	128
6.15	Membership Functions for Task_3	129
6.16	Task Executing Sequence of MtFIP	130
6.17	Results for Task_1, Task_2 and Task_3	131

Chapter 1

Introduction

1.1 Evolvable Fuzzy Hardware

Evolvable systems have attracted a great deal of attention from researchers, both from the hardware architecture and computational modeling perspectives. The idea of evolution in the context of system implementation was inspired by the Darwinian metaphor of natural selection and adaptation. Since the concept of artificial life came about at the end of the 1980s, the study of evolvable systems has since gained significance as an important area in this field. In recent years, interest in this area has gained momentum. There is greater recognition of its potential in addressing certain issues or problems pertinent to real-world systems.

Basically, an evolvable system refers to a system that can alter its own architecture or configuration to accommodate for changes in the operating environment. Such

architectural reconfiguration should occur autonomously and dynamically [1–3]. In principle, it is possible to realize the system’s evolutionary capability using either software or hardware solutions. Strictly speaking, if the evolution is based on a hardware solution, then it is more appropriately referred to as *evolvable hardware* (EHW) systems. In practice, an EHW system in its strict sense is difficult to realize. Hence, to date, most implementations of EHW systems employ a combination of hardware as well as software solutions.

While software solution offers greater flexibility, it may not be able to deliver performance at a level that is able to meet the requirements of some applications. A case in point is the *Asynchronous Transfer Mode* (ATM) cell-scheduling problem, which is described in [4–7]. For such an application, a software solution is not able to meet the time-critical nature of the application. On the other hand, hardware solution or evolvable hardware offers the potential of real-time throughput and is usually more compact. From an implementation point of view, the challenge is to design and configure systems that are able to capitalize on the flexibility of software alternatives to complement the constraints imposed by the hardware.

Research related to evolvable hardware have been reported, some of which describes potential real-life practical applications [1–3, 5, 8–17]. Evolvable hardware offers great potential as an entirely new approach for adaptive hardware and complex electronic circuit design. The architecture can be modified such that reconfiguration of the hardware can be carried out without interrupting its normal mode of operations. It is common to implement evolvable hardware systems using *programmable logic*

devices (PLD), such as *field programmable gate array* (FPGA). Suitable architecture of the system can be determined by using evolutionary algorithms to evolve the appropriate FPGA configuration [17–24].

Many evolutionary computing techniques have been reported, such as *genetic algorithms* (GA) [25, 26], evolutionary programming [27], evolutionary strategies [28], memetic algorithms [29, 30], genetic programming [31] and particle swarm optimization [32]. They have been proven to be successful in many applications to solve multi-objective optimization problems [33, 34]. Among them, GA is the most popular [34, 35]. GA is a powerful search methodology suitable for problems with poorly characterized or unknown solution space. Hence, GA is usually employed to guide the evolution of the EHW.

In general, evolvable hardware can be classified into two groups according to the type of evolution; extrinsic EHW and intrinsic EHW. Extrinsic EHW relies on a simulated evolutionary process to evolve the hardware configuration, which is executed in external computing resources. Only the elite configuration will be downloaded into the hardware device. Some of the works on extrinsic EHW are reported in [36–39]. Intrinsic EHW on the other hand relies on an evolution module embedded in the hardware for online evaluation and evolution. Recent works on intrinsic EHW are reported in [7, 11, 17, 40, 41].

The concept of evolvable hardware is extended to a class of systems with dedicated hardware for fuzzy logic inference. For simplification, such a class of hardware implementation is referred to as *evolvable fuzzy hardware* (EFH). Certain applications

of real-time fuzzy control may require a hardware solution towards fuzzy inference. If the domain rules and membership functions are static, then it is usually not an issue since extensive work on fuzzy inference hardware architecture has addressed this aspect adequately. On the other hand, if the domain rules or possibly the memberships change dynamically, it is imperative that the fuzzy inference hardware evolves according to the changes and lends itself to reconfiguration. Another important aspect of evolvable hardware is the evolutionary engine, which can be realized through software or hardware solutions. The issue of domain rule set reconfigurability is addressed through the conceptualization of a *reconfigurable fuzzy inference processor* (RcFIP) that supports online context switching.

The functional configuration of RcFIP is characterized by the domain rules and membership functions. Through evolutionary mechanisms implemented with an application in mind, a functional fuzzy system is derived. Earlier work by Lim and Ng [42] describes a genetic paradigm for learning the domain rule set with predefined membership functions. It should be emphasized that in principle, evolution of both the rule sets and memberships can occur simultaneously. In practice, the experience has shown that evolving the rule sets with fixed memberships usually offers sufficient coverage for functional adaptability of the fuzzy systems, a notion that was originally described by Thrift [43] and Lim *et al.* [44].

1.2 Fuzzy Inference Processor

1.2.1 Fuzzy Inference Overview

Fuzzy systems has become an active area of research since the conceptualization of fuzzy set theory by Zadeh [45,46]. The versatility of fuzzy systems has been demonstrated through many successful applications. Some examples include washing machine [47], air-conditioner [48], automobile control [49], robotics [50], financial loan management [51], model predictive control [52], image and speech processing [53,54]. Fuzzy systems are efficient in dealing with complex, nonlinear and time-varying processes whose mathematical models are difficult to describe [55].

Usually human experts are able to draw conclusions from imprecise, vague or ambiguous information based on experiences. Fuzzy logic offers a similar flexible approach by admitting imprecision and vagueness in inferencing, making it convenient to incorporate an expert's knowledge. It is common for such knowledge to be represented as a set of linguistic rules and its validation range, which are referred to fuzzy rules and membership functions respectively.

In general, the fuzzy rules can be expressed as *if {Antecedent} then {Conclusion}*. For example, *if {it is RAINY} then {the ground is WET}*; *if {it is DARK and the road is RUGGED} then {drive SLOWLY}*. In these cases, *RAINY*, *WET*, *DARK*, *RUGGED* and *SLOWLY* are referred to as linguistic terms. A collection of the linguistic terms is referred to as the linguistic terms set. The linguistic terms set

and its validation range are defined by the membership functions.

Based on the fuzzy rules and membership functions defined, the fuzzy inferencing process consists of 4 sequential steps; fuzzification, rule evaluation, output composition and defuzzification. Usually both rule evaluation and output composition are incorporated into the inference engine. For convenience of description, a simple example is adopted. The membership functions is as shown in Fig. 1.1. The fuzzy rules in the rules base are:

Rule 1: If Input1 is Small and Input2 is Light then Price is low

Rule 2: If Input1 is Small and Input2 is Heavy then Price is Medium

Rule 3: If Input1 is Big and Input2 is Light then Price is Medium

Rule 4: If Input1 is Big and Input2 is Heavy then Price is High

Fuzzification translates the crisp inputs into a fuzzy set. The newly coming crisp input is applied into the membership functions, to calculate the degree of truth for this rule premise. For example, in Fig. 1.1, if $Input1 = 2.25$, the fuzzification module translates this crisp value into fuzzy set: 0.8 degree of truth to be *Small* and 0.5 degree of truth to be *Big*.

Figure 1.1: Membership functions for a simple example

Figure 1.2: Mamdani implication and max-min composition procedure

According to the *antecedent* (rule premises) of the newly coming inputs after fuzzification, the fuzzy rules in the rule base with the same *antecedent* are picked up within the rule evaluation step. The Mamdani implication is also carried out in this step. The degree of truth for the *conclusion* is calculated in Equation 1.2.1.

$$\mu_{conclusion}(z_{u,v}) = \min[\mu_{input1}(u), \mu_{input2}(v)]; u \in U, v \in V, z \in Z \quad (1.2.1)$$

where $\mu_{input1}(u)$, $\mu_{input2}(v)$ and $\mu_{conclusion}(z_{u,v})$ are the degree of truth of *Input1*, *Input2* and the corresponding *conclusion* respectively. U , V and Z are universe of discourse for *Input1*, *Input2* and *Output* respectively. The Mamdani implication procedure is illustrated in Fig. 1.2. With the application of all the rules, the overall

conclusion is composed of contributions from each of the rule. The extent of the contribution for each rule depends on the degree of matching between the inputs and the *antecedents* of the rule.

In the output composition step, all of the fuzzy subsets assigned as *conclusions* are aggregated to derive a single fuzzy set. The *max-min* composition scheme as modeled in Equation 1.2.2 is adopted for this step.

$$\mu'(z) = \max[\mu_{conclusion1}(z), \mu_{conclusion2}(z), \dots]; z \in Z \quad (1.2.2)$$

In the equation, $\mu'(z)$ is the degree of truth of the combined fuzzy set. This *max-min* composition process is also shown in Fig. 1.2.

Defuzzification converts the output fuzzy set to a crisp value. There are two commonly used defuzzification methods; the center of area (COA) method and the mean of maximum (MOM) method. The COA defuzzification method is defined in Equation 1.2.3.

$$Z_{COA} = \frac{\sum_{j=1}^n z_j \cdot \mu'(z_j)}{\sum_{j=1}^n \mu'(z_j)} \quad (1.2.3)$$

where Z_{COA} is the final crisp output. z_j is the value of the element at the corresponding output membership functions. n is number of the elements in the output fuzzy set.

The MOM defuzzification method is described in Equation 1.2.4.

$$Z_{MOM} = \sum_{j=1}^m \frac{z'_j}{m} \quad (1.2.4)$$

where z'_j is the value of the element with maximal degree of truth. m is the number of such elements.

1.2.2 Fuzzy Inference Hardware Implementation

In practice, depending on the requirements of an application, there are essentially two modes of implementing fuzzy systems that can be considered. If an application is not time-critical, the inference engine of the system can be realized in software or some form of general purpose computing devices. For some real-time applications, it may be necessary to resort to special purpose inference hardware. The implementation of special purpose hardware for fuzzy reasoning is well established. Depending on the speed requirement, a suitable architecture with an appropriate level of parallelism can be configured to achieve the desired processing speed [56]. This type of hardware solution employing dedicated microelectronic circuits to realize the fuzzy inferencing system is referred to as a fuzzy inference processor [57].

Fig. 1.3 shows the general block architecture of a fuzzy inference processor with two inputs and a single output. The fuzzification module adapts discrete valued inputs into a representation appropriate for the inference engine. In general, it serves to transform measured crisp data into a fuzzy set. In Fig.1.3, the two inputs

Figure 1.3: General block architecture of fuzzy inference processors

C_1 and C_2 are typically derived from devices such as analog-to-digital converters or other form of interface to a real world domain. With the inputs C_1' and C_2' , the inference engine applies logical deduction mechanisms to map the inputs to a desired output D based on the pre-specified *if-then* rules and membership functions. The output from the inference engine D is a fuzzy concept. It represents the fuzzy conclusion that needs to be converted into a discrete or crisp output y . The process of conversion is referred to as defuzzification. Typically, the output y represents a precise control action. It can serve as a direct control action or be converted by means of a device such as digital-to-analog converter for interfacing to the real world application domain. Since the rules and membership functions are domain and problem specific, it is logical that most implementation of fuzzy inference processors incorporates a memory module for programming the rules and memberships. Hence, customization of such fuzzy inference processors involves programming the rules and memberships into the memory table.

Usually, there are two ways to realize a fuzzy inference processor. One way is by

means of explicit implementation approach. In the development of a fuzzy system, the input and output variables are first identified. Then, the membership functions and fuzzy rules are formulated, which are collectively referred to as the fuzzy relation knowledge. The fuzzy relation knowledge which is a form of mapping programmed into memory elements and together with the hardware inference circuitry form the basis of a fuzzy inference processor. This is a form of explicit implementation of the fuzzy inference systems. For example, the adaptive fuzzy hardware system depicted in [58] consists of six parts; knowledge base, input fuzzifier, dynamic membership function generator, inference processing unit, defuzzifier and control unit. The knowledge base is stored in four types of memory blocks; input fuzziness memory, membership function memory, rule index memory and rule weight memory. Logic circuitry is employed to achieve the *max-min* composition in the inference processing unit. Usually, a two-stage addition/accumulation and a divisor make up the defuzzifier circuitry. The performance of such a system is limited by the delay of the circuitry in carrying out each logical inference. Other examples of such explicit implementation approaches can be found in [51, 59–63].

Another way to realize a digital fuzzy inference processor is by means of implicit implementation approach. For such an approach to work, a mapping of the inputs and outputs is created based on a software inferencing model. Such a mapping can then be conveniently programmed as a memory map in the processor. The fuzzy inference process is then carried out based on the inputs and the pre-programmed mapping. For example, Lim *et al.* [47] presented a framework for the development

of PLD-based fuzzy controllers. The performance of such systems is usually limited by the access time of the digital memory devices. Hence, the main advantage of the implicit approach is the potentially high speed of inference. However, such an approach is only able to support a single context since the switching of context requires the complete reconfiguration of the whole memory map. Some other examples of implicit implementation approaches can be found in [64–69].

1.3 Multi-Task Processor

There has been rapid development in the past three decades on *integrated circuits* (IC) design and manufacturing techniques. As a result, the time-to-market cycle has been shortened significantly. This continuing development means that versions of IC products have to be updated frequently, thus shortening the life span of each generation of products. Industry is seeking for more flexible design methodologies to extend the products' life cycle and to cut down the design cost. It would be more appealing if products offer added capabilities and good performance for a wide range of applications concurrently. In this respect, multi-tasking IC design approach becomes an interesting research topic. Such kind of multi-tasking products relies on a single processor chip being able to perform multiple tasks or applications concurrently [70, 71]. The flexibility and performance are two crucial factors for multi-task chips.

In general, there are two design approaches for multiple tasking chips; multi-processor

in a single chip and single processor with multi-tasking capability. For the former approach, multiple tasks are handled by multiple processors. Each dedicated processor in the chip serves a specific task. Hence, good performance is achieved by capitalizing on the capability of true parallel computing. Recent *system-on-chip* (SoC) technology shows a trend toward high level of integration capability, making multi-processor cores in a single chip feasible [72]. Olukotun *et al.* [73] believes that multi-processor in a single chip will be one of the hottest architectures. Paulin *et al.* [74] presented a domain-specific multi-processor SoC architecture named *StepNP*. Theelen *et al.* [75] proposed a scalable multi-microprocessor architecture consisting of a number of identical master processors and configurable co-processors. Rutten *et al.* [76] designed a so-called *Eclipse* chip, a heterogeneous multi-processor architecture template for JPEG, MPEG and digital video applications. These multi-processor SoC architectures with high performance are able to meet application-specific requirements [77]. The disadvantage is that the circuit becomes quite complex due to the integration of a complete multi-processor system and it usually requires long development time.

For the latter approach, multiple tasks are manipulated by a single processor with multi-tasking capability. Such single multi-task processor could be a general purpose processor. Great flexibility is offered by general purpose processors for different tasks. However, the processing speed of general purpose processor architecture may not be suitable for some applications with strict performance requirement [78, 79], such as robotics, digital multimedia and high-energy physics experiments [61, 80]. The development of special purpose or application-specific processor architecture

is more appropriate to cater for the design constraints [77, 80]. Fuzzy inference processor is a good candidate, since it is able to offer high execution speed for time-critical applications [81–83].

Fuzzy inference processors have evolved significantly in the past two decades. The inference performance has been improved significantly; 6.25M FLIPS, 6.3M FLIPS and 7.0M FLIPS as reported in [80, 84, 85] respectively. Good flexibility has been achieved by employing reprogramming and reconfiguring methodologies [78, 86]. However, there is still a tradeoff between flexibility and performance for fuzzy inference processors. Some commercial fuzzy processors have been reported to be capable of offering about 10M FLIPS inferencing speed with over several tens of thousands of rules, such as SAE81C99A [87], SAE81C991 [57, 88] and FP-5000 [89]. They are suitable for most applications and able to provide both high flexibility and inferencing performance for multiple tasks. However, their hardware architectures are quite complicated. They have large number of I/O pins, operation modes, and flexible knowledge base structure to facilitate the users to configure them for different tasks. For example, SAE81C991 has 10 operation modes, 8 inference operators and 4 defuzzification methods [57]. In SAE81C991, there are maximally 1024 knowledge bases, each of which is with 128 rules, 4 inputs and 1 output. It can support up to 4096 inputs and 1024 outputs. These dedicated fuzzy inference processors are limited to high-end applications due to the high cost of development [55, 78, 90]. Hence, there are still significant challenges to improve both flexibility and performance with compact architecture for fuzzy inference processors with multi-tasking

capability.

1.4 Motivations

There are four main motivations for this project.

- (1) To design a new kind of fuzzy inference processor which has good flexibility and good reconfigurability at low development cost.
- (2) To design a context switchable fuzzy inference processor, which yields the high extensibility of explicit implementation approach, as well as the high inference speed of implicit implementation approach.
- (3) To incorporate adaptive capability for fuzzy inference processors, with capacity for intrinsic evolvability.
- (4) To propose a multi-tasking fuzzy inference processor, allowing for seamless context switching among different applications.

The details on the motivations are presented next.

The motivation for the initial design of a fuzzy inference processor by Togai and Watanabe [59] was its suitability for applications requiring high inference speed. To realize an application, domain specific knowledge is programmed into the fuzzy inference processor beforehand. The system output is derived through inferencing based on the inputs applied and the domain knowledge. A typical fuzzy inference

processor stores the fuzzy rules and memberships as domain knowledge for specific scenarios. The knowledge base is easily updated for the different scenarios and applications by reprogramming the domain knowledge in the memory section of the fuzzy inference processor. With progress in system design methodology and integrated circuit processing technology, it is possible to cover a wider scenario scope by incorporating larger domain knowledge in the processors. Many dedicated fuzzy inference processors that are capable of handling large rule base and large number of input/output variables have been fabricated. For example, Omron's FP-5000 supports up to 32,639 rules [89], while Siemens' SAE81C991 supports up to 131,072 rules [57]. These fuzzy inference processors yield high performance, good compactness and good programmability. They are suitable for most situations and applications. The disadvantage is their high development cost [55,78,90]. Normally, for most low development cost fuzzy inference processors, only a limited scope of fuzzy context is stored. For dynamic scenarios that exceed the coverage of the current context knowledge, it may be necessary to derive new knowledge context through training. The fuzzy inference processors may have to pause or suspend their operations to wait for the training procedure to complete. It will be desirable if the fuzzy inference processors are capable of reconfiguring themselves to cater the context changes online. Hence, it is necessary to design a new kind of fuzzy inference chip which has good flexibility and good reconfigurability at low cost of development.

The explicit implementation (as described in Section 1.2.2) of fuzzy inference processors adopts hardware circuitry to realize fuzzy inference procedure. It offers high extensibility. However, the performance is limited by the delay of the circuitry that processes the logical inference steps. For example, in [91], the digital fuzzy inference processor is designed with a target operational speed of 36 MHz. However, the practical performance achieved is a speed of 10 MHz, due to a speed-limiting problem of the on-chip two-phase clock generator. On the other hand, the implicit implementation (as defined on the 4th paragraph of Section 1.2.2) of fuzzy inference processors offers a potentially high speed of inference. The performance is only limited by the access time of the digital memory devices. However, such an approach lacks extensibility because it is only able to support a single context. The complete reconfiguration of the whole memory map is required to switch the context for different scenarios. Hence, it is necessary to design a context switchable fuzzy inference chip, which yields the high extensibility of explicit approach, as well as the high inference speed of implicit approach.

In a time-critical application, the scenarios change dramatically. It is not feasible to enumerate the knowledge base to cover all scenarios exhaustively. Therein, it is weak in handling unpredictable situations. In reality, a workable context in one scenario may become unacceptable for other scenarios in situations where changes may be unpredictable, therefore meaningful to incorporate adaptive capability for fuzzy inference processors, with capacity for intrinsic evolvability, just like the natural life. This intrinsic evolvable fuzzy hardware incorporate within the capacity to

modify its own hardware architecture and behavior autonomously. If the scenario changes, the architecture will also adapt accordingly, hence maintaining the system performance at an acceptable level.

Based on the motivations mentioned above, a RcFIP is designed which allows for online context switching in [83,92]. A specific instance of membership functions and domain rules is referred to as a context. Hence, a switch in context is a result of changes either in fuzzy rule sets or membership functions. In the proposed scheme, it is considered that only the rule set is involved for the updating of context. This means, the RcFIP is capable of handling context updating brought about by changes in the fuzzy rule set. More importantly, the design methodology of the RcFIP is the combination of explicit implementation approach and the implicit implementation approach. It is suitable for most time-critical applications due to the high inferencing performance of 17M FLIPS reported in [93]. The detailed architecture of the RcFIP will be outlined in the thesis.

The context switchable RcFIP incorporates the desirable characteristic of real-time reconfigurability, adaptation and high inference speed, making it an ideal platform to develop the multi-task fuzzy inference processor. Based on the RcFIP, there is a potentially useful extension of our work towards a RcFIP-based real time *multi-task fuzzy inference processor* (MtFIP).

For multi-task systems, context switching within the same application is usually faster with less communication overhead. However, dynamic context switching among different applications is often expensive [94]. In this project, based on the

design idea of the RcFIP, a real time MtFIP is proposed making context switching very convenient among different applications. In the proposed MtFIP, multiple tasks share most of the hardware resources within a compact structure. To the best of my knowledge, there are no such fuzzy inference processors reported in literature. In general, the proposed MtFIP is expected to feature easy context switching, high flexibility, high inference performance and compact structure. These expectations are confirmed in this thesis.

1.5 Objective

There are four goals specified for this project.

Firstly, a RcFIP is proposed, capitalizing on the high extensibility of the explicit implementation approach, as well as the high inference speed of implicit implementation approach.

The design of the RcFIP involves a combination of implicit implementation approach and explicit implementation approach. Using a structured scheme of multiple rule level mappings, a formatted memory map is derived, which is referred to as *fuzzy inference mapping* (FIM). For the scheme to work, a novel methodology of conflict resolution or aggregation is incorporated. This is analogous to the process of defuzzification, a necessary step in the explicit implementation of fuzzy reasoning hardware. The whole processor can be implemented with FPGA at low development cost.

Secondly, the RcFIP is implemented to cover all the contexts in the GA search space and be capable of switching between contexts expeditiously.

All the domain knowledge is trained as contexts into the FIM of the RcFIP in the GA search space. Once the FIM of the RcFIP is configured, there is no need to change its configuration unless there is a change in the membership functions. If a context changes due to updating of membership functions, the new FIM will be generated and refreshed into the RcFIP. FIM reconfiguration is not necessary for the switching of context caused by the change in domain rules. Hence, for the RcFIP, the FIM approach is convenient and capable of updating the current configuration for different context. It is also convenient and capable of updating its context for varying applications or tasks. When the scenario changes dramatically, the GA will search for the better chromosome in the GA search space. Subsequently, the better solution can be applied directly based on the existing configuration of the FIM. In this respect, RcFIP offers good flexibility and good reconfigurability.

Thirdly, a developmental structure of an intrinsic evolvable fuzzy hardware using the RcFIP as the core-processor is proposed. The learning phase and evolution are realized online. The normal operation of the evolvable fuzzy hardware is not interrupted during the normal operational stage.

The fourth goal of this project pertains to a RcFIP-based MtFIP proposed for concurrent multi-tasking. It is capable of offering high performance with compact hardware structure for multiple tasks.

1.6 Contributions

The outcome of the work has resulted in several significant contributions.

Firstly, the proposed RcFIP supports online context switching. The RcFIP allows for easy in-system reconfiguration, as well as offering high inference throughput to support time-critical applications. The RcFIP has been developed with VHDL and validated on FPGA. The context switchable RcFIP is the ideal platform to develop evolvable fuzzy hardware.

Secondly, the proposed developmental framework supports the realization of evolvable fuzzy hardware, which uses the RcFIP as the core processor. It is feasible to realize online adaptation and intrinsic evolvable fuzzy hardware all within a single chip under current SoC technology.

Thirdly, the proposed RcFIP-based MtFIP supports seamless context switching among multiple tasks within a compact hardware structure, which is a significant advantage compared to other multi-task processors.

Fourthly, some fresh and novel ideas are explored on how to further apply evolvable fuzzy hardware and multi-task processor to dynamic real-time fuzzy applications.

It is emphasized that a central theme of the work is to design a novel RcFIP and a MtFIP, as well as developing a framework for the implementation of an online adaptive and intrinsic evolvable fuzzy hardware system. It is perceived as a major contribution in the field of fuzzy inference processors and evolvable hardware

research.

1.7 Organization of Report

This chapter has provided a broad introduction of the project, including some basic background on evolvable fuzzy hardware, fuzzy inference processors and multi-task processors. The motivation, objective and contributions of the project have also been presented. The remaining chapters of this thesis will be organized as follows.

In Chapter 2, a literature review on fuzzy inference processors will be given. In Chapter 3, the overall functional flow of the evolvable fuzzy hardware developmental framework based on a RcFIP is outlined. In Chapter 4, the chip architecture of the RcFIP will be described in details. A detailed illustration on the method of partitioning the rule blocks and the process of configuring the RcFIP are also described. In Chapter 5, the functionalities of the RcFIP and RcFIP-based evolvable fuzzy hardware systems are simulated based on three applications. The online context switching capability of the RcFIP is also demonstrated in this chapter. Chapter 6 presents the hardware architecture of the proposed RcFIP-based MtFIP. Finally, this thesis is concluded in Chapter 7.

Chapter 2

Fuzzy Inference Processors Review

Fuzzy inference processors have been an active research area since 1986. Significant contributions on fuzzy inference processors have been reported in literature [48, 53, 56–58, 61, 78, 80, 81, 84, 86–88, 95–103]. The fuzzy inference processors can be classified into several groups based on different criteria, as shown in Fig. 2.1.

In Fig. 2.1(a), according to different design techniques employed, fuzzy inference processors are classified into analog fuzzy inference processors, digital fuzzy inference processors and mixed-signal ones.

- Analog fuzzy inference processors are implemented using analog circuitry. They originated from the works of Yamakawa *et al.* [104, 105]. It is further classified into current-mode [95, 102, 106–110] and voltage-mode fuzzy inference processors [84, 111–116].

- Digital fuzzy inference processor was first proposed by Togai *et al.* [59] in

Figure 2.1: Classifications of fuzzy inference processors

1986. From then on, many forms of digital fuzzy inference processors were proposed [53, 58, 63–66, 80, 81, 86, 87, 96, 97, 99, 101, 117–120]. As mentioned in Section 1.2.2, digital fuzzy inference processors can be further divided into two groups; explicit implementation processors and implicit implementation processors. The design of the proposed RcFIP with 17M FLIPS adopts the combinations of the explicit and implicit implementation approaches [83, 92], as shown in Fig. 2.1.

- The mixed-signal fuzzy inference processors [82, 98, 100, 103, 121–124] are developed by both analog circuits and digital circuits implementations.

The analog fuzzy inference processors yield high inference speed with relatively simple and compact architecture, low power consumption, as well as good compatibility with sensors [95, 105, 112, 125]. The analog fuzzy inference processors are able to support continuous input/output variables. There is no need for analog-digital-conversion (ADC) and digital-analog-conversion (DAC) circuits [113, 122]. However, analogue circuits implementations lack extensibility and flexibility compared to digital implementations [82, 111]. Analog implementations also suffer from relatively low accuracy [84, 102], since they are susceptible to noise disturbances and process variations [124].

Digital fuzzy inference processors offer high precision [84], good programmability and flexibility by employing logic and memory circuits, as well as good compatibility with other digital systems [105, 111]. Nowadays, with powerful formal design methodology and good design tools support, the realization of digital systems is generally more efficient. The digital fuzzy inference processors are also popular since development and prototyping can be achieved in a relatively short time.

Over the years, digital fuzzy inference processors have evolved significantly. Initial implementation of Togai's inference processor accommodates fuzzy subset of 31 elements. The membership value of each element is discretized with 4 binary bits. All 16 rules can be processed in parallel. The fuzzy inference processor can perform approximately 80,000 fuzzy logic inferences per second (80K FLIPS). Watanabe *et al.* [60] continued to enhance the fuzzy inference processor. With the upgrade of memory size and inference speed, the universe of discourse of fuzzy subset is en-

larged to 64 elements and up to 102 fuzzy rules can be processed in parallel. The processor can perform up to 580K FLIPS. Subsequently, the inference speed and performance have improved tremendously as a result of significant advancement in hardware technology. The inference speed of the fuzzy inference processor is up to 7M FLIPS as reported in [85,126] and 10M FLIPS in [57,87,88]. High density, flexible architecture and availability of devices with good programmability feature such as FPGA, offer even greater versatility for system implementation. The potential inference speed of up to 12.5M FLIPS [127] and 35M FLIPS [65] were reported by employing the Xilinx Virtex-E and XC3000-family FPGAs respectively.

FPGA is a good option to implement the digital fuzzy inference processors at relatively low developing cost. Compared to dedicated fuzzy inference processors, the FPGA-based fuzzy inference hardware offers rapid system prototyping and reduces development time significantly [78]. It is suitable for fast realization of a physical circuit and quick hardware verification [118,128]. It is therefore not surprising that FPGA has been a popular medium for implementing fuzzy logic controllers. For example, Kim [78] employed Xilinx XC4013 FPGA to implement the controller. Juang and Hsu [129] proposed a adaptive fuzzy processor for temperature control using Altera FLEX EPF10K100 FPGA. Li *et al.* [128] implemented fuzzy logic controllers into Altera EPF6024 FPGA to guide a car-like mobile robot, while Mas-moudi *et al.* implemented the fuzzy logic processor into Altera APEX20K200E FPGA for the wall-following control of a car-like robot [130]. Singh and Rattan [118] used Xilinx XCV100 FPGA to implement their fuzzy controller for a

DC motor. Lin *et al.* [131] implemented a fuzzy sliding-mode controller using Xilinx Virtex-II XC2V1000 FPGA. Homburg and Palomera-García [132] also adopted Xilinx XC2V1000 FPGA to implement the fuzzy processor. Rani *et al.* [133] implemented the digital fuzzy logic controller with Xilinx Spartan XC2S300E FPGA. Bonilla *et al.* [66] used Xilinx XC4010E FPGA while Yasin *et al.* [120], Jiang and Jin [134] used Altera FLEX10K FPGA to implement the fuzzy logic controllers.

Besides improvement in inference speed, higher resolution for digital fuzzy inference processors has also been achieved. For examples, the input resolution is 10-bit and the resolution of membership functions is 8-bit for Toshiba T/FC150 processor [55], while the input resolution of 12-bit with membership functions resolution of 8-bit have been reported in [62]. These digital fuzzy processors with high inference throughput can support rigorous real-time applications. For example, these digital fuzzy inference processors are useful in applications involving image and speech processing where general purpose hardware based on standard processors or microprocessors may not be able to deliver the necessary inference throughput.

According to the criteria of different developmental directions as shown in Fig. 2.1(b), the development of fuzzy inference processors falls into two directions; the category of high flexibility processors and the high inference performance category. For the high flexibility processors, the most flexible implementation is based on microprogram running on a processor [121]. Surmann and Ungering [67] introduce the fuzzy system implementations using general purpose microcontrollers. Pravadalioglu [135] proposes a fuzzy logic controller for a DC motor control employing a

general purpose RISC (reduced instruction set computing) microcontroller. While Salapura [136] proposes a fuzzy RISC processor, which is extended from a general purpose RISC processor by adding only a few new instructions. This RISC processor adopts subword parallelism technique to support multiple membership degrees accessed in a single cycle. The processing performance has a speed-up of 37% to 85% than the original general purpose RISC processor. The prototyping for this RISC processor utilizes a Xilinx XC4000XL FPGA, as reported in [117]. The approach based on general purpose processors with specific instructions gives great flexibility. However, the performance achieved are worse than those of high inference performance category [81, 106].

Most fuzzy inference processors are aimed at high inference performance. Louverdis and Andreadis [53] present a digital fuzzy inference processor, which exhibits inference performance of 601K FLIPS with 54 rules. A suitable architecture with a certain level of parallelism will serve to enhance inferencing performance [137]. A parallel processing fuzzy architecture *KAFA* is presented in [97]. Guo and Peters [82] propose a fuzzy inference co-processor with a parallel inference engine. The processing speed is up to 4.2M FLIPS with 125 rules. Pipeline architecture technique is also very popular for achieving higher inferencing performance. Aranguren *et al.* [81] present a pipeline sequential fuzzy inference processor with 5 pipeline levels architecture. The inferencing speed achieved is 2.4M FLIPS. Ascia *et al.* [56] design a fuzzy inference processor with pipelined stages. The processing speed achieves 2M FLIPS with 256 rules and 5.2M FLIPS with 32 rules. Falchieri *et al.* [80] present a

fuzzy inference processor involving 16 pipeline stages architecture, with processing speed of about 6.3M FLIPS for 64 rules. Some more fuzzy inference processors with pipelined architecture can be found in [61, 96]. Such fuzzy inference processors are proven to be able to offer high inferencing performance. However, they tend to be restricted being dedicated to certain specific fuzzy applications [78, 136]. The flexibility is limited. Nevertheless, the parallel or pipelined architecture makes it possible to reduce the hardware utilization rate.

The proposed RcFIP-based MtFIP aims to leverage on the advantages of both the high flexibility category and the high inference performance category, as shown in Fig. 2.1(b). This aspect will be presented in Chapter 6 of this thesis.

Chapter 3

EFH Developmental Framework

Many applications are required to operate in scenarios that change dramatically. With conventional fuzzy hardware system, the domain rules are usually fixed regardless of the change in operating conditions. In order for the system to achieve consistent and reliable performance, evolvable fuzzy hardware could be a viable option. It allows the architecture of a hardware system to be altered accordingly to suit the requirements of the operating environment. To be effective, it is also important to talk about the framework to support the development of the EFH. This section covers the overall developmental structure of EFH with the RcFIP as the core processor.

3.1 GA Overview

The important concepts pertinent to the realization of evolvable fuzzy hardware are fuzzy logic and evolutionary algorithm. Through some evolutionary processes, in particular a GA, the system of fuzzy rules relevant to a problem can be made to evolve into a form that makes it efficient in tackling the problem concerned.

A GA refers to the implementation of a computational model inspired by Darwin's theory of evolution. In GA, stochastic evolutionary processes are carried out to achieve global optimization through selection, crossover and mutation operators. GA works on a population of candidate solutions. The population pool of candidate solutions is commonly referred to as chromosomes. Hence, each chromosome is essentially a coding of a potential solution to the problem.

The chromosome is a string of genes. The performance of each chromosome is evaluated by means of the fitness function. The fitness function is formulated according to the problem domain. The fitness value of each chromosome is utilized for parents selection. Selection is achieved by awarding fitter chromosomes in the parent generation with higher chances to reproduce, so as to ensure that useful chromosomal properties are carried forward to the next generation.

GA is able to test and exploit the chromosomes in the search space. The crossover and mutation are the recombination operators used to explore the GA search space. With crossover, an offspring chromosome is derived from two parents. A offspring chromosome tends to be fitter than both the parents, since it inherits useful genetic

characteristics from each of the parents. A simple illustration of crossover is as shown in Fig. 3.1. Starting with binary strings of 101001101 and 001010011 and a randomly chosen crossover site at position 4, the offspring produced is 101010011 or 001001101.

Figure 3.1: One-point crossover

Mutation can be effected by randomly choosing some points in the string and the value stochastically altered. For example, the string 010111001 undergoing mutation at position 5 will result in 010101001, as shown in Fig. 3.2.

Figure 3.2: One-point mutation

3.2 Genetic Coding

GA can be used as a tool for designing and tuning fuzzy systems [138]. For an n -rule fuzzy system, one possible encoding of the genetic structure of a chromosome is as follows:

$$S_{n/len} = (g_1, g_2, g_3, \dots, g_{len}) \quad (3.2.1)$$

where len is the size of the maximal rule set of the fuzzy system. $g_1, g_2, \dots, g_{len}$ are referred to as genes. Each gene is an identifier corresponding to the *conclusion* of a fuzzy rule. For such genetic fuzzy systems, the genetic coding be derived based on a fuzzy linguistic matrix [43, 44].

A linguistic matrix for a 14-rule context is shown in Fig. 3.3. The maximal rule length is 25 ($len = 25$). There are 14 cells filled with the output linguistic terms. An unfilled cell indicates the absence of a term for the output variable, which is referred to a "NULL" cell. For this 14-rule fuzzy system ($n = 14$), the number of NULL entries are equal to $(len - n)$ or 11. A chromosome can thus be obtained from the linguistic matrix by going through each cell in a row-wise manner and coding each *conclusion* term as an integer. In this case, the chromosome obtained in Fig. 3.3 is "4020310200135420340040003", since the integers 1, 2, 3, 4, 5 and 0 correspond to the output five linguistic terms: NH, NL, ZE, PL, PH and NULL respectively.

$x_2 \backslash x_1$	NH	NL	ZE	PL	PH
NH	PL		NL		ZE
NL	NH		NL		
ZE	NH	ZE	PH	PL	NL
PL		ZE	PL		
PH	PL				ZE

Output:
 Null Entry - 0
 NH - 1
 NL - 2
 ZE - 3
 PL - 4
 PH - 5

Figure 3.3: Genetic coding for a 14-rule context

For a n -rule fuzzy system, the size of the GA search space is $C_{len}^n \times u^n$ [44]. u is the

number of linguistic terms defined for the output variable. Considering a 14-rule ($n = 14$) fuzzy system, its output linguistic terms set described for the output variable is $\{NH, NL, ZE, PL, PH\}$ ($u = 5$) and the number of the maximal rule set is 25 ($len = 25$). In this case, the search space for the GA is $C_{25}^{14} \times 5^{14}$ or 2.72×10^{16} .

3.3 Developmental Framework

A fuzzy system is characterized by the domain rules and membership functions. Realization of the system in hardware implies that the rules and memberships are encoded into the hardware. If the domain knowledge constantly requires updating, it is necessary for the hardware to accommodate the knowledge updating. In this respect, EFH incorporates such capability to effect real-time knowledge updating. For this to be possible, the hardware model of the fuzzy system should lend itself to easy reconfiguration, ideally online and without ever disrupting the normal operation of the fuzzy system.

The goal of the RcFIP is ultimately to support evolvable fuzzy hardware systems with intrinsic evolutionary capability. Before the architecture of the RcFIP is depicted, the overall functional flow of the developmental framework of evolvable fuzzy hardware with RcFIP as the core processor is described. Fig. 3.4 illustrates the developmental framework of EFH systems. In general, there are two stages in the development process; evolutionary stage and the RcFIP configuration stage. The role of the RcFIP is to carry out the fuzzy inferencing process. During the evolution-

ary stage, a mechanism of learning through a GA derives a suitable context based on an existing scenario of the application. The representation of the context is in terms of fuzzy rule set and membership functions. In the proposed scheme, since the membership functions are fixed, the updating of context need to involve only the rule set. Since the context is directly applied and configured into the hardware, real-time performance of the system is maintained. Furthermore, the in-system context switching capability means that normal operation of the real-time control system is not affected.

Figure 3.4: Developmental framework of the RcFIP-based EFH

The main task in the development of a RcFIP-based system is the configuration of the RcFIP's memory map, known as FIM. With the FIM configured, the RcFIP can function as an autonomous reasoning system with the necessary interface to external environment. Sampled inputs are digitized by means of analog-to-digital converters (ADC). The digitized k_1 -bit and k_2 -bit inputs form the basis of triggering a complete inferencing process in the RcFIP. The output of the RcFIP is the result of each inferencing process. It is in digital format and depending on needs and requirements, may have to undergo digital-to-analog conversion (DAC) to derive the control action for the application. The evolution module in Fig. 3.4 searches for the optimal rule set by means of a GA. The rule set is updated into the RcFIP only if it is deemed to be a better performing rule set compared to the active rule set.

The context applicable to the problem at any point in time is evolved by a GA. In the proposed approach, a context change manifests itself as a change in the domain rule set. When the fuzzy rule set changes, the architecture of RcFIP is able to conveniently accommodate the change. This work focuses on the evolution of fuzzy rule set as the basis of the genetic-fuzzy learning method.

An important issue pertaining to evolvable hardware systems is the triggering of the evolutionary process. This refers to the process whereby the system initiates a hardware reconfiguration. Two approaches can be considered to achieve this. The first approach is a timer based scheme whereby hardware reconfiguration is triggered periodically. The second approach is performance based triggering of the

evolutionary process. Hence, the evolutionary process is triggered whenever the level of performance of the system falls below a certain acceptable level.

The previous work has presented the framework and some applications of genetic-fuzzy learning for evolving fuzzy rule set [4, 6, 42]. Each chromosome is the coding of a rule set. The length of the rule set depends on the number of membership functions defined for the input variables. For example, a system with two input variables characterized by term sets of v elements for one variable and w elements for the other will result in a rule set of length $(v \times w)$. The FIM of the RcFIP is configured in a manner that is independent of the domain rules but specific to the membership functions defined for the fuzzy terms. This way, the FIM functions as an inference model regardless of the context that is presented to the RcFIP.

Chapter 4

RcFIP Hardware Architecture

4.1 RcFIP Architecture

The RcFIP is a novel implementation of a fuzzy inference processor with capability to support dynamic real-time applications. The scheme employed is a combination of methodologies used in the implicit implementation approach as well as explicit implementation approach to support online context switching. There are two working stages of the RcFIP, the configuration stage and the normal operation stage. During the configuration stage, the FIM is created with the support of a software configuration platform. From this software platform, the necessary digital data required to configure the FIM memory are generated. The representation of a context is in terms of fuzzy rule set and membership functions. In the proposed scheme, since the membership functions are fixed, the possible rule set represents the context, which

is applied to the RcFIP. Since the context is directly applied and configured into the hardware, real-time performance of the system is maintained. Furthermore, the in-system context switching capability means that normal operation of the real-time control system is not disrupted.

Figure 4.1: Block Architecture of RcFIP

The overall block architecture of the RcFIP is as shown in Fig. 4.1. There are five main functional blocks; fuzzy inference mapping (FIM), context management unit (CMU), address encoder module (AEM), output aggregation module (OAM) and control unit.

The FIM in the RcFIP is a structured representation of the fuzzy relational knowl-

edge. The number of partition blocks in the FIM depends on the size of the linguistic terms set for the input variables. Consider the size of the linguistic terms set for a 2-input system to be v and w . Then in general, there are $(v \times w)$ partition blocks in the FIM, $PB_{<1,1>}$, $PB_{<1,2>}$, $PB_{<1,3>}$, $\dots$, $PB_{<v,w>}$. For example, if each input variable is characterized by 5 linguistic terms, the number of partition blocks in the FIM is 25, essentially the size of the maximal rule set. During the process of inferencing, not all the partition blocks in the FIM are active or enabled. Only the contents of the FIM partition blocks enabled by the current context in the CMU are accessed.

Within the CMU, the context of the application is stored in a $(v \times w \times m)$ -bit parallel-load register, where m depends on the size of the linguistic terms set for the output variable. Consider a fuzzy system with output variable described by the linguistic set $\{D_1, D_2, \dots, D_u\}$. In this case, the parameter m is calculated in Equation 4.1.1.

$$m = 1 + \lfloor \log_2 u \rfloor \quad (4.1.1)$$

where u is the number of output linguistic terms. For example, the linguistic terms for the output variable are $\{very\ slow, slow, medium, fast, very\ fast\}$. The number of bits for m is equal to 3, since there are 5 linguistic terms defined for the output variable.

There are $(v \times w)$ segments in the context register of the CMU, $E_{<1,1>}$, $E_{<1,2>}$, $E_{<1,3>}$, $\dots$, $E_{<v,w>}$. Each segment is represented by m bits. Based on the current

context of the application, the CMU generates a m -bit address for accessing each partition block in the FIM. Each m -bit address connects to only one partition block. Hence, the m -bit address derived from $E_{\langle 1,1 \rangle}$ in the CMU connects to $PB_{\langle 1,1 \rangle}$ in the FIM, while the m -bit address derived from $E_{\langle 1,2 \rangle}$ connects to $PB_{\langle 1,2 \rangle}$ in the FIM, and so on.

The function of the AEM is to generate the address for accessing each FIM partition block. The number of bits k_1 and k_2 for inputs depend on the resolution of the input variables. In a practical scenario, both k_1 -bit and k_2 -bit inputs may be obtained from ADCs. For example, choosing both k_1 and k_2 to be 4 as the resolution of the inputs will allow for 16 discrete levels of input values. From the inputs, the AEM generates a (k_1+k_2) -bit address for accessing all the partition blocks in the FIM. Herein, the complete address for accessing each partition block in the FIM is made up of (k_1+k_2+m) bits.

Although the address lines from the AEM feed into all the FIM partition blocks, only the partition blocks that are relevant to the context are accessed. The outputs of all the FIM partition blocks that are being accessed become the inputs to the OAM. In order to achieve the desired aggregation effect, a novel circuit is designed to realize the appropriate mode of conflict resolution. The outcome of the OAM is a crisp digital output of the fuzzy system. The output consists of a whole and fractional part. By incorporating the fractional part, significant rounding off error in the output can be avoided. An additional 4 bits is incorporated for the fractional part. Hence, for the p -bit data on the bi-directional data bus, there are $(p+4)$ bits

of data output from the OAM. It is also the final output of the RcFIP.

A global synchronizing clock signal is derived from the control unit. Besides synchronizing data flow on the bi-directional data bus, the control unit handles the switching between the two functional stages of the RcFIP. During the configuration stage, the FIM is programmed by data from an external configuration program via the FIM configuration interface. For the normal operation stage, data originate from the FIM partition blocks for further computation in the OAM.

4.1.1 FIM

Fig. 4.2 shows the general layout of the FIM to be realized by on-chip RAM (random access memory). The whole FIM is divided into partition blocks ($PB_{<1,1>}$, $PB_{<1,2>}$, $PB_{<1,3>}$, ..., $PB_{<v,w>}$) and the total number of partition blocks is determined by the size of the input linguistic terms set. In this case, the size is denoted to be v for one input and w for the other. Essentially, the FIM acts as the fuzzy relational knowledge base of the application. It is a representation that models the process of inputs fuzzification, rules evaluation and output composition. The process of configuring the FIM involves a configuration procedure that emulates the fuzzy inferencing process. Once configured, there is no need to update the FIM whenever a change in the context occurs. As alluded to in Chapter 3, a change in the context could be a result of change in the domain rules or membership functions. A context change is normally triggered by a change in the scenarios of the application. In this respect, RcFIP is able to handle context switching brought about by a change in

the domain rules. Although in principle, changing membership functions can also bring about a context change, it is argued that it is not as useful as a context change brought about by a change in the domain rules [43, 44].

Figure 4.2: FIM Configuration

All data in each partition block exist in 2's complement format. The width of the data bus is p -bit. The *most significant bit* (MSB) is the sign bit, while the magnitude is represented by the remaining bits. The FIM as shown in Fig. 4.2 must support two modes of operations. During the configuration mode, the partition blocks in the FIM are configured one at a time. This is achieved by selectively enabling

each FIM partition block one by one. The control unit sets the status of all the FIM partition blocks to *Write* mode during the configuration process. During the normal operation mode, all the FIM partition blocks are set to *Read* mode and the context register maintains the current rule set of the application, updated by means of a parallel data load operation.

4.1.2 CMU

The working flowchart of the CMU is as shown in Fig. 4.3. An application context is usually derived at the system level. For example, in an evolvable hardware system, a suitable context is derived by the evolution module of the system. The context is then updated via the context register. With additional logic circuitry, the CMU decodes the current context into enable signals and address to access the FIM. The context may change because of a change scenario during run time. The new context derived from the evolution module is loaded into the context register on the rising edge of the context switching clock.

The circuit block diagram of the CMU is as shown in Fig. 4.4. The main components in the CMU are the context register and the decoding logic block. The $(v \times w \times m)$ bits register is a parallel loading register. In designing the CMU, it is necessary to consider the appropriate scheme of register loading for context updating. On the one hand, if the architecture is to cater for fast and efficient context switching, the number of I/O pins incurred will be high. However, if the speed of context updating is not so critical, the loading of the $(v \times w \times m)$ bits register can be carried out in a

Figure 4.3: Working flowchart of CMU

sequential manner. From the context in the register, the logic block in the CMU decodes the value of each rule in the register to derive the necessary enable signal for each FIM partition block. This is handled by the decoding logic block consisting of OR gates circuitry to generate the enable signals.

On the whole, the context register is made up of segments labeled as $Rule\langle 1,1\rangle$, $Rule\langle 1,2\rangle$, $Rule\langle 1,3\rangle$, ..., $Rule\langle v,w\rangle$. Each segment maintains the active rule represented by m bits, denoting the output as the conclusion part of a rule. For each rule, the m -bit signals become the inputs to the decoding circuit. For an inactive rule, the decoding logic generates a false for the enable signal. For active rules denoted by non-zero values, the decoding logic generates output signals to enable the corresponding partition blocks.

Figure 4.4: Hardware architecture of CMU

4.1.3 AEM

The AEM as shown in Fig. 4.5 is the address encoding block of the RcFIP. It generates the address for accessing the partition blocks in the FIM. During the normal operation mode, the $R/\overline{W}$ signal is pulled high. The AEM takes in k_1 -bit and k_2 -bit data. During the configuration mode, the $R/\overline{W}$ signal is low. This causes the multiplexer (MUX) to select the address counter as the source that generates the address for writing into each partition block in the FIM. The address counter is essentially a (k_1+k_2) -bit counter that automatically generates the address for accessing the FIM partition blocks.

In the RcFIP, the complete address consists of two parts. The first part of the address is derived from the (k_1+k_2) input bits. Each set of inputs generates a unique (k_1+k_2) address bits. From the CMU, the m -bit coding of each rule in the context register is appended to the (k_1+k_2) address bits. This forms the complete address for the FIM partition blocks. Only the contents of the FIM partition blocks

Figure 4.5: Schematic of AEM

enabled by the CMU are accessed.

In principle, each set of inputs will access all the $(v \times w)$ FIM partition blocks. The data accessed from all the partition blocks become the inputs to the OAM for defuzzification. Although the current context in the CMU activates the partition blocks in the FIM by means of the enable signals, outputs are derived only from those partition blocks that contribute to the conclusion according to the inputs into the system.

4.1.4 OAM

The defuzzification operation is handled by the OAM circuit. The inputs to the OAM are the outputs from the FIM partition blocks. For fuzzy inference processors, designing the defuzzification block is a challenge because it is a significant bottleneck in achieving high fuzzy inference speed [101, 139]. Normally, defuzzification process requires multiplication and division, making the fuzzy inference processors complicated and complex. One design approach by Watanabe *et al.* [60]

uses two adders, two registers and one divider to avoid the multiplication step. The numerator and denominator are calculated separately. An adder and a register carry out summation for the numerator while another adder and register carry out summation for the denominator. Finally, the crisp output is calculated by a divider. Jou *et al.* [58] also designed similar circuit. The defuzzifier circuit enhances the performance of their fuzzy inference processors, although a divider is still necessary. Yamakawa [105] designed a defuzzification circuit to eliminate the divider in his analog fuzzy hardware. The defuzzification was accomplished by employing a grade-controllable membership function circuit with feedback loops. Employing look-up table to replace the multiplication and division is another solution for defuzzification, an approach adopted by [48, 63]. For example, a (16×8) ROM-type multiplier lookup table and a (24×9) ROM-type division lookup table are employed in [48]. This approach can achieve very high performance. The drawback is that a potentially large memory space is required to cover the entire range of possibilities.

The method of aggregation adopted for the RcFIP is a form of rule level defuzzification averaging [140]. Conceptually, one can perceive the output from each partition block of the FIM as a form of defuzzified conclusion derived for a one-rule fuzzy sub-system. To cover the entire context, it is necessary to aggregate the outputs from all the FIM partition blocks. However, not all the outputs from partition blocks are relevant. The outputs of inactive or disabled FIM partition blocks are irrelevant data, hence ignored. Therefore, the OAM is only required to aggregate the valid data from FIM partition blocks that are active. The appropriate FIM

partition blocks are activated based on the context applied to the system. Besides that, during the normal operation of the RcFIP, the partition blocks are selectively enabled according to the fuzzy inputs applied. For each inferencing process, the number of active one-rule fuzzy sub-systems may vary in the range of 1 to $(v \times w)$.

To illustrate the aggregation process, consider the general case of $(v \times w)$ partition blocks in the FIM. Let D_{ij} denote the output of $PB_{<i,j>}$ in the FIM. The general form of the equation for averaging is as follows:

$$Z = \frac{\sum_{i=1}^v \sum_{j=1}^w D_{ij}}{N}, \quad (1 \leq N \leq v \times w) \quad (4.1.2)$$

where N is the number of active partition blocks. The output Z from Equation (4.1.2) is the final output of the RcFIP.

OAM Circuit

The circuit to compute Z can be implemented by a summing block and a divider. Although the summing block is relatively straightforward from a circuit implementation point of view, the implementation of the divider is potentially complex. To overcome this, a hierarchical averaging approach is adopted in order to manage the potential complexity of the circuit. Hierarchical averaging involves multiple stages of computation as illustrated in Fig. 4.6. In the circuit, mostly *Avg_2* circuit blocks are employed, with flexibility to incorporate *Avg_3* circuit blocks whenever necessary. The function of the *Avg_2* and *Avg_3* blocks is to do averaging of 2 and 3 p -bit

binary inputs respectively. An *Avg_2* block is realized by an adder and a divide-by-2 circuit while an *Avg_3* block is realized by an adder and a divide-by-3 circuit. The implementation of the divide-by-3 operation in the *Avg_3* block is conveniently realized using a simple look-up table. Although certain degree of error is expected in hierarchical averaging by the circuit in Fig. 4.6, the extent of the error compared to direct averaging based on Equation (4.1.2) is not significant. From simulations on control applications presented in Chapter 5, the error due to hierarchical averaging has no noticeable effect on the functional performance of the RcFIP.

Figure 4.6: Hardware architecture of OAM

Avg_2 Circuit

The *Avg_2* module calculates the average of two inputs. This module consists of two main parts; an adder/subtractor and a divide-by-2 circuit. In the proposed scheme, the signals fed into the OAM are signed data. The sum of two inputs is obtained by

employing the adder/subtractor. Divide-by-2 operation is realized by using a shift register. In this circuit, one more fractional bit is adopted for the output result to enhance precision. Hence, for p -bit input data, the number of bits for output is $p+1$ bits. The working flowchart of the *Avg_2* module is as shown in Fig. 4.7.

Figure 4.7: Flowchart of *Avg_2* Module

The Xilinx Virtex-II XC2V2000 chip is employed to synthesize *Avg_2* circuit. The RTL schematic view for the circuit is as shown in Fig. 4.8.

Avg_3 Circuit

The *Avg_3* module calculates the average of three inputs. This module consists of two main parts; an adder/subtractor and a divide-by-3 circuit. Compared to the divide-by-2 circuit, a divide-by-3 circuit is much more complicated in hardware

Figure 4.8: RTL Schematic View for *Avg_2* Module

realization. In the proposed circuit, a simple look-up table is employed to realize the divide-by-3 operation. Two more fractional bits are added to enhance precision. The overall schematic of the *Avg_3* module is shown in Fig. 4.9.

In Fig. 4.9, *Sum* is the sum for three inputs, while *Count* is the number of valid data among these three inputs. The *Count* signal controls the operation of the multiplexer.

OAM Circuit Discussion

The OAM circuit has been described to realize the defuzzification in the RcFIP. As discussed in the first paragraph of Section 4.1.4, there are two common approaches

Figure 4.9: Overall Schematic for *Avg_3* Module

to realize division for the defuzzification block in digital fuzzy inference processor; external divider chip approach and look-up table approach. The implementation of a divide-by- n circuit is rather complex, especially when the denominator is a random value in the range of 1 to n . The processing speed of such a divider is relatively low. The proposed OAM realization is much simpler. The OAM is made up of *Avg_2* and *Avg_3* modules in a hierarchical structure. The processing speed of the OAM is higher than a normal divider.

The look-up table approach offers very high processing speed, limited only by the access time delay. However, a large memory is required, hence not cost effective. For the OAM approach, with *Avg_3* module which represents a small block in the whole defuzzification step, the memory requirement is much more manageable. An example is given to show the memory space required for the OAM approach. The number of v is chosen to be 5, w to be 5 and p to be 8. Hence, there are (5×5)

partition blocks in the FIM. There are 25 8-bit data fed into the defuzzification module. In this case, the memory space required is $(2^{(8+2)} \times (8+2) + 2^{(10+2)} \times (10+2))$ bits or 58K bits. It is clear that the memory space required in the OAM approach is quite small. The OAM circuit is therefore a more practical approach compared to that of a look-up table approach.

4.1.5 Control Unit

The overall synchronization of the RcFIP is carried out by the control unit. Besides generating the clock signals, the control unit also generates signals to synchronize operations involving the RcFIP and external hardware interface, such as ADC and the FIM configuration interface. The fuzzy logic inferencing is performed during the RcFIP normal operation stage. For situations where the membership functions or the application domain changes, a new set of configuration data is generated by the FIM configuration program. The FIM configuration interface sends a *REQ#* signal to the control unit to initiate a configuration update. A *GRANT#* signal is fed back to the FIM configuration interface, triggering the RcFIP configuration process. The overall handshaking protocol between the control unit and the FIM configuration interface is illustrated in Fig. 4.10(a).

A transfer controller in the control unit coordinates the data transfer to/from the FIM. There are two types of transfer modes; the *Write* and the *Read* transfer modes. During the *Write* mode, data from the FIM configuration interface are written to the FIM via the $(v \times w)$ data bus. The FIM partition blocks are configured one

Figure 4.10: Hardware Block Architecture of Control Unit

by one sequentially. The first FIM partition block configured is $PB_{<1,1>}$. Data is transferred from the FIM configuration interface to $PB_{<1,1>}$ without interruption. Next, $PB_{<1,2>}$ is configured, followed by $PB_{<1,3>}$. This continues until the last block $PB_{<v,w>}$ is configured. For the *Read* mode, all the $(v \times w)$ FIM partition blocks are enabled simultaneously. This way, data can be accessed from the FIM in parallel and fed into the OAM.

The switching between the *Write* and the *Read* transfer mode on the bidirectional data buses are handled by data selection switches. In Fig. 4.10(a), there are $(v \times w)$ bus switches controlled by the transfer controller. The detailed structure of the $(v \times w)$ bus switches is as shown in Fig. 4.10(b). The select signals are set to '0' for the *Write* mode to set up connections between the FIM configuration interface and the FIM. During the *Read* mode, connections are established between the FIM and the OAM.

4.1.6 Circuit Prototyping

The prototype of the RcFIP is coded block by block using VHDL '93, a widely used standard for hardware specification language. The simulation and synthesis environment is Mentor Graphics FPGA Advantage 6.3. It provides a suite of EDA tools, which include ModelSim and LeonardoSpectrum. The various stages of implementation are as shown by the flowchart in Fig. 4.11.

Figure 4.11: RcFIP prototype realization flowchart

After compiling the logic functions description for the digital circuits, the behavioral and timing simulations are carried out with ModelSim for functional and timing

verification. The waveforms of all entities are analyzed based on the test-bench constraints. LeonardoSpectrum is employed to perform circuit synthesis next. The circuit blocks are converted from VHDL language to a gate level netlist of *Electronic Device Interchange Format* (EDIF). The netlist of EDIF file is imported and implemented into a Xilinx Virtex-II FPGA by using the Xilinx ISE5.0. The bit-stream file is then generated and downloaded into the FPGA using Celoxica FTU2 (*File Transfer Utility 2*). In this project, a Xilinx XC2V2000 Virtex-II FPGA is employed for circuit prototyping [141].

For the RcFIP prototyping example, it is considered both inputs to be 6 bits ($k_1, k_2=6$). The number of bits m to code the *consequent* of each rule is chosen to be 3 bits ($m=3$). The width of the data bus is chosen to be 8 bits ($p=8$). The resource utilization report is as shown in Table 4.1. It is observed that the logic gates and circuit size is manageable even though the RcFIP is configured to cover all the contexts in the GA search space.

Device Utilization for XC2V2000BG575			
Resource	Used	Avail	Utilization
IOs	99	408	24.26%
Global Buffers	1	16	6.25%
Function Generators	8722	21504	40.56%
CLB Slices	4361	10752	40.56%
Dffs or Latches	329	22728	1.45%

Table 4.1: RcFIP circuitry scale reports

The timing simulations of this RcFIP are based on 66 MHz clock in ModelSim, which means the clock period is equal to 15 nanosecond (ns). In order to evaluate

Figure 4.12: Waveforms for RcFIP with new inputs update (Situation 1)

the overall performance of the RcFIP, simulation waveforms under 3 situations are analyzed.

(Situation 1): This situation deals with newly updated inputs derived from ADCs, while the context within the CMU is kept unchanged.

(Situation 2): A context switching occurs within the CMU, while the inputs derived from ADCs are kept unchanged.

Figure 4.13: Waveforms for RcFIP with context switch (Situation 2)

(Situation 3): This situation deals with both inputs updated and contexts switching. There are newly updated inputs derived from ADCs, and the context switching occurs at the same time.

The waveforms for Situation 1 are as shown in Fig. 4.12. Referring to the positions of the cursors in the figure, the inputs are updated at Cursor 1 position. The final output from the RcFIP is derived at Cursor 2 position in Fig. 4.12. The total processing time for one fuzzy logic inference is 4 clock cycles in Situation 1.

Figure 4.14: Waveforms for both new inputs and context updates (Situation 3)

The waveforms for Situation 2 and Situation 3 are illustrated in Fig. 4.13 and 4.14 respectively. It is observed that the RcFIP deals with such requests within 4 clock cycles.

From these three figures, the observations show that the RcFIP is capable of processing one fuzzy logic inference within 4 clock cycles. Hence, the performance of the RcFIP is 17M FLIPS (i.e. $\frac{1}{4 \times 15ns}$). This inference speed is high enough for most time-critical applications [64].

The specification of this RcFIP is listed in Table 4.2. Several digital fuzzy inference processors reported in [65, 80, 85, 120, 127, 132, 133] are also listed in the same table. The RcFIP is the combination of the implicit and explicit implementation approaches. The fuzzy processor in [65] adopts the implicit implementation, while the fuzzy processors in [80, 85, 120, 127, 132, 133] adopt the explicit implementation. The RcFIP and the processors in [65, 120, 127, 132, 133] are implemented using FPGAs, while the processors in [80, 85] are implemented with ASICs.

	RcFIP	[65] 1995	[132] 2003	[127] 2003	[120] 2004	[133] 2005	[80] 2002	[85] 2005
Implementation	Hybrid	Implicit	Explicit	Explicit	Explicit	Explicit	Explicit	Explicit
Processor Prototype	Xilinx VirtexII	Xilinx VirtexE	Xilinx VirtexII	Xilinx VirtexE	Altera FLEX10K	Altera Spartan2	ASIC	ASIC
Input Resolution	6-bit	3-bit	6-bit	8-bit	8-bit	7-bit	7-bit	--
Number of Inputs	2	2	6	2	2	2	2	2
Number of Outputs	1	1	1	1	1	1	1	1
Performance (MFIPLS)	17	35	3.5	12.5	5	1.4	6.3	7.0
Online Reconfigurability	Yes	No	No	No	No	No	No	No
Adaptability for EFH	Yes	No	No	No	No	No	No	No

Table 4.2: RcFIP specifications and comparisons

It is observed from Table 4.2 that the proposed RcFIP incorporated the advantages of both implicit and explicit implementations fuzzy inference processors.

4.1.7 Evolvable Hardware System

For conventional fuzzy inference processors, the designers formulate the fuzzy contexts in a subjective manner. It results in performance that may not be optimal [142]. In this project, the RcFIP is configured under the guidance of a GA. An example of RcFIP-based evolvable hardware system configuration is illustrated by the block architecture in Fig. 4.15. The evolution module can be realized by a microcontroller-based GA unit. Functionally, the role of the GA is to evolve a suitable context which is then configured into the CMU of the RcFIP. The RcFIP-based system is capable of maintaining suitable level of performance in a dynamically changing operating environment.

Figure 4.15: Functional block diagram for complete system

In Fig. 4.15, data from the domain environment are fed into the Evolution Module as training data. A GA process is triggered and operates on the training data stored in the memory. GA operations such as selection, crossover, mutation and replacement are carried out in the GA unit. The derived context with the best fitness is updated

into the CMU. As discussed in the second paragraph of Section 4.1.3, the address to access the FIM consists of two parts. One part of it is derived from the system's inputs, while the other part is derived based on the current context in the CMU. With the data accessed from the FIM, the OAM of the RcFIP computes the control output. The whole setup allows for online context switching, a critical feature of intrinsic evolution for evolvable hardware systems.

4.2 Configuring FIM

The application developmental framework for an RcFIP-based system requires a FIM configuration program to compute the data set for configuring the FIM. Once programmed, the RcFIP has the potential for high inference performance with small response delay and good adaptive capability. Furthermore, the gate count and hence hardware complexity is minimized. In this subsection, the whole process of configuring the FIM is illustrated in details.

4.2.1 General Descriptions

The FIM configuration is structured according to the linguistic matrix of a fuzzy system. The linguistic matrix is basically a tabular representation showing the input-output mapping of the fuzzy system [143]. As a general illustration, Fig. 4.16 shows the linguistic matrix for a system of two input variables x_1 and x_2 described by the linguistic sets $\{A_1, A_2, \dots, A_v\}$ and $\{B_1, B_2, \dots, B_w\}$ respectively. The

$x_1 \backslash x_2$	A_1	A_2	A_3	$\dots$	A_v
B_1	$\alpha(A_1, B_1)$	$\alpha(A_2, B_1)$	$\alpha(A_3, B_1)$	$\dots$	$\alpha(A_v, B_1)$
B_2	$\alpha(A_1, B_2)$	$\alpha(A_2, B_2)$	$\alpha(A_3, B_2)$	$\dots$	$\alpha(A_v, B_2)$
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
B_w	$\alpha(A_1, B_w)$	$\alpha(A_2, B_w)$	$\alpha(A_3, B_w)$	$\dots$	$\alpha(A_v, B_w)$

Figure 4.16: Linguistic matrix of maximum rule set for two input variables

maximum number of rules is therefore $(v \times w)$. Each entry $\alpha(A_i, B_j)$ in a cell of the matrix denotes the linguistic value of the *conclusion part* of a rule. For example, the cell $\alpha(A_i, B_j)$ is interpreted as "if $\langle x_1 = A_i \rangle$ and $\langle x_2 = B_j \rangle$ then $\langle y = \alpha(A_i, B_j) \rangle$ " where $\alpha(A_i, B_j)$ is a linguistic value of the variable y . If the linguistic set of the output variable y is considered to be $\{D_1, D_2, \dots, D_u\}$, this will imply that $\alpha(A_i, B_j) \in \bigcup_{k=1 \dots u} \{D_k\}$.

The maximum number of rules for the system described is $(v \times w)$. Each cell in the rule matrix is assigned a value in the linguistic terms set of the output variable y . To facilitate discussion, consider κ as the rule set representing the context of an application. In general κ can be written as follow:

$$\kappa = \bigcup_{i=1 \dots v} \bigcup_{j=1 \dots w} \{\alpha(A_i, B_j)\} \quad (4.2.3)$$

The size of κ is $(v \times w)$. Since $\alpha(A_i, B_j) \in \{D_1, D_2, \dots, D_u\}$, κ is a representation of a context within a mapping space of $(v \times w)^u$. This is based on the assumption that κ is defined for a maximal rule set of $(v \times w)$.

In order to partition the domain knowledge in the FIM, the entire mapping space

is divided into groupings based on the *antecedents* of the rules. According to the linguistic matrix of Fig. 4.16, the cells can be conveniently classified into groupings.

For example, the Grouping<3,5> is representative of the following rules:

If $\langle x_1 = A_3 \rangle$ and $\langle x_2 = B_5 \rangle$ then $\langle y = D_1 \rangle$

If $\langle x_1 = A_3 \rangle$ and $\langle x_2 = B_5 \rangle$ then $\langle y = D_2 \rangle$

If $\langle x_1 = A_3 \rangle$ and $\langle x_2 = B_5 \rangle$ then $\langle y = D_3 \rangle$

$\vdots$

If $\langle x_1 = A_3 \rangle$ and $\langle x_2 = B_5 \rangle$ then $\langle y = D_u \rangle$

Hence, the number of rules that falls into each grouping depends on the size of the linguistic terms set characterizing the *conclusion part* of the rules. All the rules in a grouping have the same *antecedent part*. The overall situation is depicted in Fig.4.17, showing the various groupings.

Figure 4.17: Linguistic matrices of the $(v \times w)$ groupings

In Fig. 4.17, $\alpha(A_1, B_1), \alpha(A_2, B_1), \dots, \alpha(A_v, B_w)$ denote the linguistic values of the conclusion part of the fuzzy rules. In general, for each Grouping< i, j >, only the cell corresponding to "If $\langle x_1 = A_i \rangle$ and $\langle x_2 = B_j \rangle$ then $\langle y = \alpha(A_i, B_j) \rangle$ " is included,

while the other cells are treated as NULL entries. With the $(v \times w)$ groupings, it can provide coverage for all possible conditions in the $(v \times w)$ -rule fuzzy systems. Hence, for any n -rule fuzzy system ($n \leq v \times w$), it can be taken apart into n number of one-rule sub-systems. Each one-rule sub-system is covered by the corresponding grouping shown in Fig. 4.17. In this way, the FIM is capable of handling all the contexts when configured. Furthermore, there is no need to modify the FIM unless there is a change in the membership functions.

The FIM configuration program configures the $(v \times w)$ partition blocks of the FIM based on the $(v \times w)$ groupings. Data for the groupings are loaded into the RcFIP during the configuration stage. The resolution of input membership functions is chosen to be k_1 and k_2 number of bits. For the output, the memberships are described using the singleton functions. The number of m bits is chosen to indicate the output variables. Hence, the width of the address bus to access the FIM is $(k_1 + k_2 + m)$. In the proposed RcFIP, the width of data bus is p bits. The complete FIM configuration procedure is demonstrated by means of a simple illustration. Following this illustration, three practical examples are presented in the next chapter. The functional simulation of the practical example is also incorporated to verify the operational functionality of the RcFIP.

4.2.2 Simple Illustration

To describe the process of configuring the FIM in the RcFIP, a simple example is used to illustrate the steps involved. The application example is intentionally

simplified to facilitate detailed illustration of the steps involved. Consider a system of two inputs x_1 and x_2 and a single output y . The membership functions for the system are as shown in Fig. 4.18.

Figure 4.18: Membership functions of a simplified system

A 2-bit ($k_1, k_2=2$) resolution is used to discretize the input membership functions. The number of bits p are chosen to be 3, and a 2-bit code for the consequent of each rule ($m=2$). The segment of the context register $Rule\langle i,j \rangle$ with m bits in the CMU specifies the conclusion part of a rule, as outlined earlier in Section 4.1.2.

For the system specified, $v=w=3$ and $u=2$. Hence, there are 9 groupings representing the fuzzy system. The FIM configuration program creates the configuration data for all the 9 partition blocks in the FIM. The whole configuration process is illustrated in the following.

Coverage Map

The digitization of the membership functions for variables x_1 , x_2 and y is as shown in Fig. 4.18. Both x_1 and x_2 are described by discretized triangular membership functions. The consequent y is described by singleton membership functions. Each

fuzzy singleton has values for magnitude and the degree of truth. According to Fig. 4.18, the magnitude for linguistic value N (*Negative*) is (-1) in decimal notation or (11.0) in 2's complement binary. When the degree of truth is 1, the *consequent value* stored is "110". For 0.5 degree of truth, the *consequent value* stored is "111". Similarly, for linguistic value P (*Positive*), the *consequent value* is "010" for degree of truth being 1, while it is "001" for 0.5 degree of truth. The coverage map for all the groupings are derived based on the Mamdani implication and *max-min* composition rule of inference. The resulting coverage map is as shown in Fig. 4.19. In the coverage map, each grouping covers a 1-rule fuzzy sub-system. For example, Grouping<1,1> of the FIM incorporates the 1-rule fuzzy sub-system whose rule is "If $\langle x_1 = S \rangle$ and $\langle x_2 = S \rangle$ then $\langle y = \alpha(A_1, B_1) \rangle$ ". There are two digitized values for each input, "00" and "01". There are four digitized values for the output, "111", "110", "001" and "010", where the MSB is the sign bit. The FIM configuration program will configure the rules in Grouping<1,1> based on the following series of well-defined rules.

If $\langle x_1 = 00 \rangle$ and $\langle x_2 = 00 \rangle$ and $[Rule\langle 1,1 \rangle = 01]$, then $\langle y = 110 \rangle$

If $\langle x_1 = 00 \rangle$ and $\langle x_2 = 00 \rangle$ and $[Rule\langle 1,1 \rangle = 10]$, then $\langle y = 010 \rangle$

If $\langle x_1 = 00 \rangle$ and $\langle x_2 = 01 \rangle$ and $[Rule\langle 1,1 \rangle = 01]$, then $\langle y = 111 \rangle$

If $\langle x_1 = 00 \rangle$ and $\langle x_2 = 01 \rangle$ and $[Rule\langle 1,1 \rangle = 10]$, then $\langle y = 001 \rangle$

If $\langle x_1 = 01 \rangle$ and $\langle x_2 = 00 \rangle$ and $[Rule\langle 1,1 \rangle = 01]$, then $\langle y = 111 \rangle$

If $\langle x_1 = 01 \rangle$ and $\langle x_2 = 00 \rangle$ and $[Rule\langle 1,1 \rangle = 10]$, then $\langle y = 001 \rangle$

If $\langle x_1 = 01 \rangle$ and $\langle x_2 = 01 \rangle$ and $[Rule\langle 1,1 \rangle = 01]$, then $\langle y = 111 \rangle$

If $\langle x_1 = 01 \rangle$ and $\langle x_2 = 01 \rangle$ and $[Rule_{\langle 1,1 \rangle} = 10]$, then $\langle y = 001 \rangle$

The data set for the Grouping $\langle 1,1 \rangle$ is as shown in Table 4.3a. Each entry in the table consists of two parts, the address and data. For example, the entry "00 00 01 : 110" means that the data "110" is stored in the memory with location specified by the physical address "000001". This data set is then programmed into the $PB_{\langle 1,1 \rangle}$ of the FIM via the FIM configuration interface during the configuration stage. The coverage map for all the 9 groupings is as shown in Fig. 4.19 with detailed illustration of the well-defined rules for Grouping $\langle 1,1 \rangle$ and Grouping $\langle 3,2 \rangle$ shown.

The FIM configuration program computes the mapping data to configure all the partition blocks of the FIM based on the coverage map. For Grouping $\langle 2,1 \rangle$, the data set of $PB_{\langle 2,1 \rangle}$ is configured as shown in Table 4.3b. Similarly for Grouping $\langle 3,1 \rangle$, the data set of $PB_{\langle 3,1 \rangle}$ is configured as shown in Table 4.3c. This is repeated for all the other groupings as shown in Tables 4.3d to 4.3i.

FIM Configuration Data

All the data sets for the 9 partition blocks are as shown in Table 4.3. These data sets are used to program the partition blocks of the RcFIP through the FIM configuration interface during the configuration stage.

00 00 01:110	00 00 10:010
00 01 01:111	00 01 10:001
01 00 01:111	01 00 10:001
01 01 01:111	01 01 10:001

a. Grouping<1,1>

00 00 01:111	00 00 10:001
00 01 01:110	00 01 10:010
00 10 01:110	00 10 10:010
00 11 01:111	00 11 10:001
01 00 01:111	01 00 10:001
01 01 01:111	01 01 10:001
01 10 01:111	01 10 10:001
01 11 01:111	01 11 10:001

d. Grouping<1,2>

00 10 01:111	00 10 10:001
00 11 01:111	00 11 10:001
01 10 01:111	01 10 10:001
01 11 01:110	01 11 10:010
10 10 01:111	10 10 10:001
10 11 01:110	10 11 10:010
11 10 01:111	11 10 10:001
11 11 01:111	11 11 10:001

h. Grouping<2,3>

00 00 01:111	00 00 10:001
00 01 01:111	00 01 10:001
01 00 01:110	01 00 10:010
01 01 01:111	01 01 10:001
10 00 01:110	10 00 10:010
10 01 01:111	10 01 10:001
11 00 01:111	11 00 10:001
11 01 01:111	11 01 10:001

b. Grouping<2,1>

00 00 01:111	00 00 10:001
00 01 01:111	00 01 10:001
00 10 01:111	00 10 10:001
00 11 01:111	00 11 10:001
01 00 01:111	01 00 10:001
01 01 01:110	01 01 10:010
01 10 01:110	01 10 10:010
01 11 01:111	01 11 10:001
10 00 01:111	10 00 10:001
10 01 01:110	10 01 10:010
10 10 01:110	10 10 10:010
10 11 01:111	10 11 10:001
11 00 01:111	11 00 10:001
11 01 01:111	11 01 10:001
11 10 01:111	11 10 10:001
11 11 01:111	11 11 10:001

e. Grouping<2,2>

10 00 01:111	10 00 10:001
10 01 01:111	10 01 10:001
11 00 01:110	11 00 10:010
11 01 01:111	11 01 10:001

c. Grouping<3,1>

10 00 01:111	10 00 10:001
10 01 01:111	10 01 10:001
10 10 01:111	10 10 10:001
10 11 01:111	10 11 10:001
11 00 01:111	11 00 10:001
11 01 01:110	11 01 10:010
11 10 01:110	11 10 10:010
11 11 01:111	11 11 10:001

f. Grouping<3,2>

00 10 01:111	00 10 10:001
00 11 01:110	00 11 10:010
01 10 01:111	01 10 10:001
01 11 01:111	01 11 10:001

g. Grouping<1,3>

10 10 01:111	10 10 10:001
10 11 01:111	10 11 10:001
11 10 01:111	11 10 10:001
11 11 01:110	11 11 10:010

i. Grouping<3,3>

Table 4.3: Data Sets for Configuring FIM Partition Blocks

4.3 Potential Tradeoff of RcFIP

The key part in the proposed RcFIP is FIM, since the FIM is configured to cover all the possible contexts in the GA search space. The number of possible contexts in the GA search space depends on the number of the linguistic terms defined for the input and output variables [44]. For example, consider a 2-input 14-rule fuzzy system with 5 linguistic terms for each input variable, and 5 linguistic terms for the

output variable. The number of 14-rule contexts in the GA search space is $C_{25}^{14} \times 5^{14}$ or 2.72×10^{16} .

Based on the architecture of the FIM in the proposed RcFIP, the overall scope of the contexts in the GA search space has to be covered within a limited number of fuzzy rules. It is important for the memory space requirement of the FIM to be manageable. Using the same example of a 14-rule fuzzy system, there are 125 fuzzy rules (5 linguistic terms of *Input1* $\times$ 5 linguistic terms of *Input2* $\times$ 5 linguistic terms of *Output*) encoded in the FIM. The whole contexts domain is partitioned into groupings according to *antecedents* of the fuzzy rules. This ensures that the FIM is structured in a manner such that it completely covers all possible contexts.

There is a potential scalability issue for the proposed RcFIP. If the number of linguistic terms defined for input variables and output variable increase, the memory space of the FIM will increase rapidly. However, with the high memory capacity of current integrated circuits, such a tradeoff is tolerable.

Figure 4.19: Coverage map of the 9 groupings for configuring the FIM

Chapter 5

Application Examples

The inference speed of the proposed RcFIP and its real time reconfigurability can be analyzed based on RTL level timing simulation in ModelSim. Besides the EDA tools timing analysis, its functionality needs to be validated by software simulations, in particular its adaptation and real time reconfigurability. How good is the actual performance? Is the inference speed fast enough for real time applications? One way to address these issues is to apply it in actual control problems. Three examples are considered to illustrate the practical functionality of the RcFIP. Two of the applications are evolutionary fuzzy system for ATM cell scheduling [4, 6] and a genetic learning fuzzy system for industrial plant automation [42]. The results reported in earlier publications of the research group can be used as a basis to validate performance and functionality of the proposed RcFIP. The other application is a rotary inverted pendulum which can be validated on actual implementation of the hardware platform.

The first application selected is that of a water bath temperature control, for demonstrating the practical functionality of the RcFIP. A scheme for controlling the temperature of the water bath based on evolutionary learning of fuzzy system has been described in [42]. In this work, the model presented in [144] is adopted to simulate the functionality of the plant controlled by an RcFIP-based fuzzy system. The simulation results and a comparison between a typical fuzzy solution and the RcFIP implementation of the control system are presented.

The second example is a rotary inverted pendulum balance control system [92]. In literature, many researchers illustrated their proposed fuzzy controllers by using the inverted pendulum system [119, 145–149]. Inverted pendulum systems are uncertain nonlinear dynamical systems [145]. It always exhibits various nonlinear behaviors under different operating conditions, external disturbances as well as physical variations [146]. The online context switching capability of the RcFIP is demonstrated using a rotary inverted pendulum system.

The third example pertains to ATM cell scheduling. This application is chosen to illustrate the potential viability of EFH for solving real-time control problem. It serves to illustrate how the RcFIP can be used to support real time online context switching. More importantly, the RcFIP can potentially address the complex issue of intrinsic evolution for evolvable hardware. The simulation results are presented, and compared the results with those of *first-in-first-out* (FIFO) and *dynamically weighted priority scheduling* (DWPS) schemes.

5.1 Water Bath Temperature Control

The water bath temperature control plant is a well-modeled control problem commonly used as demonstration for fuzzy control algorithms and development tools. The objective is to control the temperature of the water bath within the desired temperature range of 25° to 80°. The schematic diagram of the overall setup is shown in Fig. 5.1.

Figure 5.1: Overall setup of water bath temperature control system

In Fig. 5.1, $r(t)$ is the temperature setting of the plant being controlled, or the desired target temperature. The output $\Delta y(t)$ represents the voltage applied to the heater. The parameter $g(t+1)$ refers to the current temperature of water. The heater is controlled by the output signal $\Delta y(t)$ from the fuzzy control system. The value of $\Delta y(t)$, which depends on the desired temperature $r(t)$ and the plant's current temperature $g(t+1)$, is the output of the fuzzy control system. The inputs for the fuzzy system come from temperature sensor in the water bath. According to Fig. 5.1, the evolution module derives the appropriate context of the problem by generating the optimal fuzzy rule set. The membership functions for the fuzzy

system are pre-defined throughout the whole control process.

The membership functions for the input variables x_1 and x_2 , output variable $\Delta y(t)$ are as shown in Fig. 5.2. Both input variables are described by the linguistic set $\{NL, ZE, PL, PM, PH\}$ while the output variable is described by $\{HD, LD, NC, LI, HI\}$. With 5 labels for each input, the maximum number of rules is 25, hence the length of the rule set for each context is 25. 4 bits ($k_1, k_2=4$) are chosen to discretize the memberships and 3 bits ($m=3$) to code the consequent of each rule. The number of bits p for the width of the data bus is chosen to be 5. The digitization of the fuzzy concepts is as shown in Fig. 5.2. Note that the 5-bit data shown as singleton membership functions are in 2's complement format, which include the consequent's magnitude and the corresponding degree of truth.

Figure 5.2: Membership functions for the water bath temperature controller

Since $v=w=5$, $u=5$, there are 25 groupings for the fuzzy system. It is therefore necessary to compute the data set to configure all 25 partition blocks of the FIM. The process is the same as that described earlier (see Section 4.2.2).

For example, the 1-rule fuzzy sub-system with the active rule "If $\langle x_1 = NL \rangle$ and

0000 0000 001:11000	0000 0000 010:11100	0000 0000 011:00000	0000 0000 100:00100	0000 0000 101:01000
0000 0001 001:11010	0000 0001 010:11101	0000 0001 011:00000	0000 0001 100:00011	0000 0001 101:00110
0000 0010 001:11100	0000 0010 010:11110	0000 0010 011:00000	0000 0010 100:00010	0000 0010 101:00100
0000 0011 001:11110	0000 0011 010:11111	0000 0011 011:00000	0000 0011 100:00001	0000 0011 101:00010
0001 0000 001:11010	0001 0000 010:11101	0001 0000 011:00000	0001 0000 100:00011	0001 0000 101:00110
0001 0001 001:11010	0001 0001 010:11101	0001 0001 011:00000	0001 0001 100:00011	0001 0001 101:00110
0001 0010 001:11100	0001 0010 010:11110	0001 0010 011:00000	0001 0010 100:00010	0001 0010 101:00100
0001 0011 001:11110	0001 0011 010:11111	0001 0011 011:00000	0001 0011 100:00001	0001 0011 101:00010
0010 0000 001:11100	0010 0000 010:11110	0010 0000 011:00000	0010 0000 100:00010	0010 0000 101:00100
0010 0001 001:11100	0010 0001 010:11110	0010 0001 011:00000	0010 0001 100:00010	0010 0001 101:00100
...	...	...	...	...
...	...	...	...	...

Table 5.1: Configuration Data Set for Grouping<1,1> of the FIM

$\langle x_2 = \text{NL} \rangle$ then $\langle y = \alpha(A_1, B_1) \rangle$ is stored in Grouping<1,1> of the FIM, where $x_1, x_2 \in \{0000, 0001, 0010, 0011\}$ and $y \in \{11000, 11010, 11100, 11110, 11101, 11111, 00000, 00001, 00010, 00011, 00100, 00110, 01000\}$. The FIM configuration program creates the data set for Grouping<1,1> as shown in Table 5.1. The data set is programmed into the $PB_{\langle 1,1 \rangle}$ via the FIM configuration interface during the hardware configuration stage.

The remaining 24 partition blocks are also configured in the same manner. After the FIM is configured, the RcFIP can function efficiently at potentially high inference speed.

The evolution module generates fuzzy rules for the water bath temperature control plant. The parameters of evolutionary algorithm are set as follows:

- Population Size = 100;
- Maximum Number of Generations = 9999;
- Crossover Probability = 0.5;
- Mutation Probability = 0.2;

$x_2 \backslash x_1$	NL	ZE	PL	PM	PH
NL			LI		HI
ZE	HD	NC		HI	HI
PL	HD			HI	
PM		HD	NC		HI
PH					

a. A 12-rule context

$x_2 \backslash x_1$	NL	ZE	PL	PM	PH
NL	NC		NC	NC	HI
ZE		NC			
PL		HD		HI	
PM	NC	HD	LI		HI
PH		LD	LD	LI	

b. A 14-rule context

$x_2 \backslash x_1$	NL	ZE	PL	PM	PH
NL			NC		NC
ZE		NC	HI	LI	HI
PL		HD	HI	NC	HI
PM	NC		LD	NC	LI
PH			NC	HD	

c. A 16-rule context

$x_2 \backslash x_1$	NL	ZE	PL	PM	PH
NL	LI			LI	LI
ZE	LD	NC	HI		
PL	NC	LD	LD	HI	HI
PM	LD	HD	HD	HI	
PH		HD			NC

d. A 17-rule context

Inputs:

NL=Negative Low

ZE = Zero

PL=Positive Low

PM=Positive Medium

PH=Positive High

Output:

Null Entry - 0

HD - 1

LD - 2

NC - 3

LI - 4

HI - 5

Figure 5.3: Four examples of contexts for the RcFIP

- Maximum Rule Size = 25.

An example of a 12-rule fuzzy system as linguistic rule matrix is as shown in Fig. 5.3a. The coding for the fuzzy rule set is "0405513055100500130500000". This represents the context of the domain problem loaded into the context register (See Fig. 4.4). For this 12-rule context, the entire fuzzy system is represented by Grouping<1,2>, Grouping<1,3>, Grouping<2,1>, Grouping<2,2>, Grouping<2,4>, Grouping<3,4>, Grouping<4,1>, Grouping<4,2>, Grouping<4,3>, Grouping<5,1>, Grouping<5,2> and Grouping<5,4>. These 12 partition blocks in the FIM of the RcFIP are activated during inferencing. Three other contexts (14-rule, 16-rule and 17-rule) are also as shown in Fig. 5.3. From the viewpoint of the RcFIP, any of the contexts specified in Fig. 5.3b, 5.3c and 5.3d can be effected by loading the corresponding codes "3033503000010503140502240", "0030303545015353023400350" and "4004423500322552115001003" respectively into the context register. The RcFIP functions accordingly, regardless of the context pre-

sented to the system via the context register.

5.1.1 Functional Simulation

The function of the RcFIP using these four different contexts is simulated and compared the results with those of the fuzzy control system in Lim and Ng [42]. Four cases of simulation are shown based on the RcFIP being used as the plant controller. The results are presented side by side with the control response of a typical fuzzy control implementation [42]. The plots of the control response for 4 different contexts at the same temperature settings (from 30° to 80°) are presented in Fig. 5.4.1, 5.4.2, 5.4.3 and 5.4.4. Each plot shows a control response for 100 time samples, corresponding to 50 minutes. For each figure, the upper plot is the temperature response based on the input applied to the heater as shown by the lower control voltage plot.

In general, all four plots in Fig. 5.4 show that the RcFIP can serve as an effective implementation of a fuzzy control application. From the plots, the RcFIP-based controller achieves the desired target temperature to within a tolerance of 0.2%. There is no noticeable difference in the control response shown by the RcFIP-based plant control system. All the four contexts were applied without the need to re-configure the FIM. This clearly illustrates the context-independent feature of the RcFIP. The results are significant, bearing in mind that the RcFIP handles context switching efficiently, hence an ideal platform for further work on evolvable fuzzy hardware, which is described next.

5.1 Water Bath Temperature Control

80

5.4.1: Control response for 12-rule context

5.4.2: Control response for 14-rule context

5.4.3: Control response for 16-rule context

5.4.4: Control response for 17-rule context

Figure 5.4: Control responses for heating from 25° to 80°

5.2 RcFIP-Based Rotary Inverted Pendulum

In this section, the online context switching capability of the RcFIP is demonstrated. The chosen test bed is a PP-300 rotary inverted pendulum [150]. The RcFIP-based system is employed to balance the pendulum. Several contexts are derived for the system. It is required to switch between the various contexts without interrupting the normal operation of the control system. This requirement is significant as it serves to illustrate how the RcFIP can be used to support online context switching. More importantly, the RcFIP can potentially address the complex issue of intrinsic evolvable hardware due to its online context switching capability.

5.2.1 Overall System Setup

The PP-300 rotary inverted pendulum as shown in Fig. 5.5 is a test-bench hardware for real-time control applications. In Fig. 5.5, the rotary inverted pendulum setup is made up of three parts; the arm, the pendulum and the motor. The arm driven by the motor, rotates in the horizontal plane. A pendulum is attached at the end of the arm. The connection from the pendulum to the arm is almost frictionless, allowing the pendulum to swing freely in the vertical plane. The motor controls the rotary direction and speed of the arm. A servo potentiometer is employed to measure the angular position of the pendulum. The rotary direction and speed of the arm are measured by using a built-in optical encoder at the motor. Complementing the PP-300 apparatus is the UC96 servo drive board [151]. This board provides buffering

Figure 5.5: Overall setup of RcFIP-based system

for the signals from the servo potentiometer and optical encoder. It also provides buffers for the PWM signals to drive the motor. The UC96 servo drive board is also the source of power supply to the ADC chip and the motor.

When in operation, an analog voltage ranging from 0V to 5V is derived according to the current position of the pendulum. The analog voltage is converted to serial digital data by a MCP3201C/PI ADC chip. The MCP3201C/PI chip is a successive approximation 12-bit analog-to-digital converter [152]. The serial digital data are

transmitted to the ADC controller via a standard SPI (*Serial Peripheral Interface*)-compatible interface. The ADC controller is the input interface between the ADC chip and the RcFIP. It generates k_1 -bit digital data. The k_1 ($k_1 \leq 12$) bits parallel data serve as *Input 1* of the RcFIP.

The current rotary direction and speed of the arm are measured by a dual channel optical encoder, namely encoder *Channel A* and *Channel B*. Each channel generates 1000 quadratic square pulses when the rotary arm completes one revolution. The signals of *Channel A* and *Channel B* encoder are fed into the encoder controller. The encoder controller is the hardware interface between the optical encoder and the RcFIP. It derives the k_2 -bit digital data. The most significant bit of the data denotes the rotating direction of the arm, where '1' represents *clockwise* and '0' for *counter-clockwise*. The remaining (k_2-1) bits denote the number of pulses per 10 milliseconds. These k_2 bits parallel data serve as *Input 2* of the RcFIP.

In the evolution module, a GA engine is used to search for an appropriate context online. The evolving contexts are switched and configured into the RcFIP directly to balance the pendulum. The normal operation of the RcFIP is not affected at all during the online context switching process. Based on *Input 1*, *Input 2* and the current context, the RcFIP carries out the fuzzy inferencing. A ($p+4$) bits digital output is derived from the RcFIP.

The PWM controller derives the PWM direction and the PWM duty cycle based on the output from the RcFIP. The duty cycle is the percentage of the pulse duration over the total pulse period. The PWM direction denotes the driving direction of

the motor, where '1' is for clockwise and '0' for counter-clockwise. A motor IC on the UC96 servo drive board takes the PWM signal and direction from the PWM controller to drive the DC servo motor [151]. It outputs voltage ranging from -5V to +5V according to the PWM direction and the duty cycle. The speed of the motor depends on the duty cycle of a digital PWM signal. As the duty cycle increases, the speed of the motor increases, and vice versa. The motor of the rotary inverted pendulum determines the rotary direction and speed of the arm to balance the pendulum.

In this project, there are three hardware interface circuits for the RcFIP-based rotary inverted pendulum system; the ADC controller, the encoder controller and the PWM controller circuits. The designs of these hardware interface circuits are presented in the following subsections.

5.2.2 ADC Controller

The ADC chip converts an analog voltage, which represents the position of the pendulum, into a 12-bit serial digital data. The ADC controller serves as the interface to translate the data between the ADC chip and the RcFIP, since the readable input for the RcFIP is the parallel digital data. A diagram for the ADC controller is as shown in Fig. 5.6.

In Fig. 5.6, the RcFIP derives the system clock to the ADC controller. The ADC chip receives an active-low chip select (nCS) input signal and a clock (ADC_Clk)

Figure 5.6: Diagram of ADC controller

signal from the ADC controller. The 12-bit serial data are outputted to the ADC controller. The ADC controller derives k_1 -bit parallel data to the AEM of the RcFIP consequently. The number of bits k_1 depends on the design specification of the RcFIP. In this case, k_1 is chosen to be 6 bits in this project.

According to the specifications of MCP3201 ADC chip [152], the ADC chip cannot run higher than 1.6 MHz or lower than 10 KHz clock frequency. In this case, the frequency of ADC_Clk is equal to 1.365 MHz, derived from the control unit of the RcFIP. During the working stage of the ADC chip, when the nCS signal is pulled low, the analog voltage from the rotary inverted pendulum is sampled and converted into a 12-bit serial data during the first two clock periods of the ADC_Clk signal. In the third ADC_Clk period, the ADC chip will output a low null bit. In the subsequent clock periods, the ADC chip will output the 12-bit serial data bit by bit, starting with MSB. After the LSB (*least significant bit*) has been sent out, the ADC controller should pull the nCS signal high and remain so for more than 625 nanoseconds, before pulling it low again to start another new sampling cycle [152].

The ADC controller is configured to pull the nCS signal high after the 15th ADC_Clk period. It lets the nCS signal remain high for 2 ADC_Clk periods (732 nanoseconds),

Figure 5.7: Flowchart for ADC controller circuit

which provides sufficient refresh time to start a new sampling cycle. The ADC controller receives 12 bits of data serially from the ADC chip and outputs them in parallel. These correspond to k_1 bits which are sent to the AEM of the RcFIP. The total working flowchart of the ADC controller circuit is as shown in Fig. 5.7.

5.2.3 Encoder controller

Input 2 of the RcFIP, derived from the rotary direction and speed of the arm, is detected by the dual channel optical encoder. For each channel, a complete

revolution produces 1000 quadratic square pulses. The phase difference between *Channel A* and *Channel B* is 90° . When the arm is rotating clockwise, the pulses of *Channel B* leads those of *Channel A* by a phase difference of 90° . When the arm is rotating counter-clockwise, the pulses of *Channel A* leads those of *Channel B* by 90° . Table 5.2 summarizes the rotary direction of the arm according to the signals of *Channel A* and *Channel B*. To detect the direction that the arm is moving, the encoder controller first samples the falling edge of *Channel A* and the logic status of *Channel B*. According to Table 5.2, if *Channel B* is low on the falling edge of *Channel A*, the encoder controller deduces that the signal of *Channel A* leads that of *Channel B*. Hence, the arm is rotating counter-clockwise. Likewise, if *Channel B* is high on the falling edge of *Channel A*, the arm is rotating clockwise, and so on.

Channels Diagram	Description	Rotary Direction
	<i>Channel A</i> rises, <i>Channel B</i> = 0	Clockwise
	<i>Channel A</i> falls, <i>Channel B</i> = 1	
	<i>Channel B</i> rises, <i>Channel A</i> = 1	
	<i>Channel B</i> falls, <i>Channel A</i> = 0	
	<i>Channel A</i> rises, <i>Channel B</i> = 1	Counter-Clockwise
	<i>Channel A</i> falls, <i>Channel B</i> = 0	
	<i>Channel B</i> rises, <i>Channel A</i> = 0	
	<i>Channel B</i> falls, <i>Channel A</i> = 1	

Table 5.2: Determination of arm rotary direction

The rotary speed of the arm is determined by counting the number of rising and falling edges of *Channel A* pulses every 10 ms. However, for safety precaution, the

PP-300 rotary inverted pendulum will never exceed 60 pulse edges per 10 ms [150]. To generate a period of 10 ms, a 100 Hz frequency clock (*Encoder_Clk*) is derived from the control unit of the RcFIP. The *Encoder_Clk* is specially configured to be of very low duty cycle, thus making the duration of the low level to be approximately 10 ms. There are 2000 pulse edges for each arm rotation, since a complete revolution produces 1000 pulses for *Channel A*. For example, if the encoder controller counts 30 pulse edges per 10 ms, then the rotary speed of the arm can be determined as follows:

$$2000 \text{ pulse edges} \Rightarrow 2\pi \text{ radian}$$

$$1 \text{ pulse edge} \Rightarrow 2\pi / 2000 \text{ rad}$$

$$30 \text{ pulse edges} \Rightarrow 30 \times 2\pi / 2000 = 0.094 \text{ rad}$$

$$\Rightarrow \text{Rotary speed of arm} = 0.094 \text{ rad} / 10 \text{ ms} = 9.4 \text{ rad/s}$$

Hence, the encoder controller circuit is designed to capture the rotary direction and speed of the arm depending on the signals from *Channel A* and *Channel B* encoder. The information is encoded as k_2 -bit digital data, where the MSB denotes the rotary direction. The k_2 -bit data serves as the *Input 2* of the RcFIP. The flowchart of the encoder controller circuit is as shown in Fig. 5.8.

Figure 5.8: Flowchart for encoder controller circuit

5.2.4 PWM controller

The PWM direction and duty cycle are derived from the PWM controller to drive the DC motor of rotary inverted pendulum. The PWM controller circuit is as shown in Fig. 5.9. The PWM controller receives the digital output from the OAM of the RcFIP. It consists of a 9-bit PWM data register and a 8-bit frame counter. The resolution of the PWM controller is 256. For the 9-bit output from the data register, the MSB represents the direction of the PWM. The remaining 8 bits represent the

Figure 5.9: PWM controller circuit

magnitude of the modulated pulses for driving the motor. Hence, there are 2^8 or 256 different discretized states of drive torque for the motor. An output of "11111111" denoting the 255^{th} state gives the highest drive torque, whereas "00000000" denoting the 0^{th} state does not activate the motor at all. However, the value of drive torque is limited at the 80^{th} state, since higher drive torque may damage the PP-300 rotary inverted pendulum apparatus itself [150]. The clock frequency of the PWM controller is 24.576 MHz and is derived from the control unit of the RcFIP.

5.2.5 Hardware Connection Interface

The overall RcFIP-based fuzzy inference system circuit can be potentially realized and fabricated on current system-on-chip technology. In this thesis, the functionality and feasibility of the proposed RcFIP-based system is demonstrated through FPGA implementation. A Xilinx Virtex-II FPGA, which is affixed on a RC-200 development board, is employed [153]. As presented in Fig. 5.5 earlier, a UC96 servo drive board serves as the bridge between the PP-300 rotary inverted pendu-

Figure 5.10: Hardware connection between two boards

lum apparatus and the RcFIP-based processor circuit. A 50-pin expansion acts as a data transfer route between the RC200 development board and the UC96 servo drive board. The general hardware connection interface is as shown in Fig. 5.10.

5.2.6 Functional Demonstration

In this subsection, the online context switching procedure involving several different contexts is illustrated. The membership functions for this rotary inverted pendulum system is as shown in Fig. 5.11. The angular position of the pendulum corresponds to *Input 1*, while the angular velocity of the arm corresponds to *Input 2*. Both input variables are described by the linguistic terms set $\{NH, NL, Z, PL, PH\}$. The motor driven voltage is the output of the system. The output variable is described by linguistic set $\{NF, NS, ZR, PS, PF\}$.

Three examples of contexts derived from the evolution module are as shown in Fig.

5.2 RcFIP-Based Rotary Inverted Pendulum

92

Figure 5.11: Membership Functions for rotary inverted pendulum

5.12. It illustrates how the contexts can be switched online and yet still maintaining the normal operation mode.

$x_2 \backslash x_1$	NH	NL	ZE	PL	PH
NH		NH	NH	PL	NL
NL	NH	NH			PL
ZE		NL	PH	ZE	
PL	PH	NL	PL	PL	ZE
PH		NL			NH

a. A 17-rule context

Inputs:

NH=Negative High
NL=Negative Low
ZE = Zero
PL=Positive Low
PH=Positive High

$x_2 \backslash x_1$	NH	NL	ZE	PL	PH
NH	PL		NL		
NL	NH		NL		
ZE	NH	ZE	PH	PL	NL
PL		ZE	PL		
PH	PL				ZE

b. A 13-rule context

Output:

Null Entry - 0
NH - 1
NL - 2
ZE - 3
PL - 4
PH - 5

$x_2 \backslash x_1$	NH	NL	ZE	PL	PH
NH		ZE		ZE	
NL	PL	NH	NH	ZE	ZE
ZE		NL	PH	PL	
PL			PL	PL	
PH	NH		ZE	PL	

c. A 15-rule context

Figure 5.12: Three contexts examples for the RcFIP

The genetic coding for the contexts in Fig. 5.12 are "0114211004025305244302001", "4020010200135420340040003" and "0303041133025400044010340" respectively. The contexts are switched online every 10 seconds to track the scenarios change. In or-

	Under fixed context	Under multi-context switching
Scenario 1	mass of pendulum = 0.0319 kg Applying Context 1	mass of pendulum = 0.0319 kg Applying Context 1
Scenario 2	mass of pendulum = 0.0419 kg Applying Context 1	mass of pendulum = 0.0419 kg Applying Context 2
Scenario 3	mass of pendulum = 0.0519 kg Applying Context 1	mass of pendulum = 0.0519 kg Applying Context 3

Table 5.3: Three different operating scenarios

der to compare the performance of the system under a fixed context and under multi-context switching, three operating scenarios are set as shown in Table 5.3. Disturbance to the inputs during the switching process are generated randomly.

The control response of the RcFIP-based rotary inverted pendulum system is as shown in Fig. 5.13. The solid lines are control responses under multi-context switching. The dashed lines are for operation under a fixed context. The solid and dashed lines overlap exactly in Scenario 1, since the same context is being applied. In Scenario 2 and Scenario 3, there is a distinct difference in performance as shown by the plots. The plots show that the RcFIP is more responsive to scenario changes under multi-context switching than that of a fixed context. From the plots traced out by the solid lines, the pendulum remains balanced, even when the RcFIP was undergoing contexts switching. This clearly indicates that the RcFIP is capable of handling online context switching without affecting the normal operation of the whole rotary inverted pendulum system.

Figure 5.13: Control responses for RcFIP-based system

5.3 RcFIP for ATM Cell Scheduling

In ATM cell scheduling, it is envisaged that the pattern of cell flow traffic may change dynamically. The objective of cell scheduling is for service provider to deliver a guaranteed level of *Quality of Service* (QoS) performance. The major parameters of QoS are cell loss and cell delay. There are many algorithms designed to solve the problem of cell scheduling. The common switching schemes are FIFO and DWPS schemes [154]. However, they are not very good in terms of QoS performance. The DWPS scheme is a significant improvement over the FIFO scheme. It adjusts the priority according to the cell flow scenarios [154]. However, its adaptation scheme

is simple and may not be efficient if the cell flow changes dramatically. The DWPS scheme is also difficult to implement in hardware.

Figure 5.14: Architectural framework of EFH for ATM cell scheduling

The concept of RcFIP-based evolvable fuzzy hardware is proposed to solve the ATM cell scheduling problem. The architectural framework of the ATM cell scheduling system is as shown in Fig. 5.14. Both *class1* and *class2* are the two classes of cell flow traffic. They are scheduled by the *Multiplexer*. The evolution engine searches for the most appropriate context. The optimal context is updated into the *fuzzy switching control* (FSC) module directly. Based on the current context and the current cell flow traffic, the FSC module schedules a cell from *Buffer1* and *Buffer2* to the *OUT* channel via the *Multiplexer*. It is clear from the framework that the FSC module should be able to support online context switching for intrinsic system evolvability. To demonstrate the applicability of the RcFIP, It simulates the ATM cell scheduling system using the RcFIP to emulate the FSC function. The simulation results of the evolvable fuzzy hardware using the RcFIP as the core-processor are compared with the results of using the FIFO and DWPS schemes. The parameters setting of the GA are as follows:

- Population Size = 10;
- Maximum Number of Generations = 14;
- Crossover Probability = 0.6;
- Mutation Probability = 0.05;
- Maximum Rule Size = 25.

The fitness function used in the evolution module is described by Equation 5.3.1.

$$F = \tau - |AveDelay - \lambda \times MaxDelay| \quad (5.3.1)$$

For the fitness function, the parameter τ is a very large constant. This way, F will be proportional to the performance of the chromosome. The larger the fitness value is, the better is the chromosome. *AveDelay* is the average delay of *class1* cell units. *MaxDelay* refers to the maximum delay possible for *class1* cell units. λ is an adjustable coefficient used to tune the desired average cell delay of *class1* cell units. In the simulation, the value of λ is chosen to be 0.2.

5.3.1 Simulation results and comparisons

Work on evolutionary fuzzy system for ATM cell scheduling has been reported in [4, 6]. The detailed QoS performance analysis and evaluation of evolutionary fuzzy system for ATM cell scheduling were discussed in [4]. The RcFIP offers a hardware alternative to realize evolutionary fuzzy system for ATM cell scheduling. The time-critical performance requirement of ATM cell scheduling is ideal to test

the applicability of the RcFIP in real-time dynamic control.

In order to test the system's performance, a realistic traffic scenario lasting for 2 seconds is adopted for simulation. Realistic traffic flow is governed by the connection admission control mechanism of ATM. This means that a connection can only be accepted if there are sufficient bandwidth resources [155]. It is therefore necessary to adopt realistic traffic patterns to verify whether the evolvable fuzzy system is effective for practical applications. One such realistic traffic scenario is as shown in Fig. 5.15.

Figure 5.15: Cell flow in a realistic traffic scenario

In Fig. 5.15, *class1* refers to delay-sensitive *constant bit rate* (CBR) traffic with cell bit rate of 80 MHz, while *class2* is a bursty traffic flow that is sensitive to cell loss. Let the *OUT* channel capacity of Fig. 5.14 be 155.52 MHz. Hence, the connection for *class2* should have a prescribed peak bit rate of 75.52 MHz (i.e. 155.52 minus 80). Assume that due to the malfunction of some nodes in the network, the peak bit rate of *class2* is about 100 MHz. This scenario is realistic to simulate the robustness and controllability of the RcFIP-based evolvable fuzzy hardware for ATM cell scheduling.

In the proposed methodology, the evolution engine is triggered periodically to derive an approximate rule set to cope with the current traffic scenario. During the search, the evolution engine evaluates the evolved contexts via the scheduling module. After each evolutionary cycle, the domain rule set in the RcFIP is updated without interrupting the switching operation.

The simulation results are as shown in Fig. 5.16. The QoS performance of the RcFIP-based evolvable fuzzy hardware are compared with those of FIFO and DWPS scheduling schemes. In order to describe the performance quantitatively, a metric is defined for balance factor η as follows:

$$\eta = 1 - \left| \frac{\rho_1}{\rho_1 + \rho_2} - \frac{\rho_2}{\rho_1 + \rho_2} \right| \quad (5.3.2)$$

where ρ_1 and ρ_2 are the slopes of the performance characteristic curves for *class1* and *class2* respectively. A larger η corresponds to a higher degree of balance between the 2 classes being compared. In Fig. 5.16.1, the slope for FIFO ρ_{1F} is approximately 1350 cells/second, while that of RcFIP-based EFH ρ_{1R} is approximately 1000 cells/second, and that of DWPS ρ_{1D} is zero. The FIFO scheme shows a maximum *class1* cell loss, while the DWPS scheme, on the other hand, shows a minimum *class1* cell loss. In Fig. 5.16.2, the slope for FIFO ρ_{2F} is approximately 8500 cells/second, while ρ_{2R} is approximately 10000 cells/second, and ρ_{2D} is approximately 10500 cells/second. Hence, for the FIFO scheme, the cell loss for *class2* is minimal, while the DWPS scheme shows a maximum *class2* cell loss in Fig. 5.16.2.

5.16.1: Cell loss of class1

5.16.2: Cell loss of class2

5.16.3: Cell delay of class1

5.16.4: Cell delay of class2

Figure 5.16: QoS performance for *class1* and *class2*

For ATM cell scheduling, better QoS performance means a good balance of cell loss between *class1* and *class2* cell flow under the constraint of less cell delay. In Fig. 5.16.1 and Fig. 5.16.2, for the FIFO scheme, the balance factor η between *class1* cell loss and *class2* cell loss according to Equation 5.3.2 is 0.274. The balance factor for the RcFIP-based evolvable fuzzy hardware scheme is 0.182 while that of DWPS scheme is 0.

Fig. 5.16.3 and 5.16.4 show the cell delay results for all three schemes. In Fig. 5.16.3, the cell delay for the RcFIP-based evolvable fuzzy hardware scheme is about 140 μs , for the DWPS scheme is about 150 μs and for the FIFO scheme is approximately 310 μs . In Fig. 5.16.4, the cell delay for the RcFIP-based EFH scheme is about 250 μs . The cell delay for the FIFO scheme is about 320 μs while the cell delay for the DWPS scheme is about 330 μs . Clearly the RcFIP-based evolvable fuzzy hardware has the smallest cell delay of both *class1* and *class2* cell flow among the three schemes. This clearly distinguishes the RcFIP-based EFH scheme from the FIFO and DWPS schemes. From the observations made on the plots in Fig. 5.16, it is evident that the RcFIP-based evolvable fuzzy hardware is more superior in scheduling the *class1* and *class2* cell flow. The RcFIP-based EFH scheme can manage *class1* and *class2* cell flow more efficiently and achieve a better QoS performance level.

The proposed evolvable fuzzy hardware using the RcFIP as the core-processor is capable of self-adaptation in a dynamically changing traffic environment to deliver the desired QoS. The key to the hardware realization of such evolvable fuzzy hardware is the proposed RcFIP, which provides real-time reconfiguration and online adaptation.

The simulation results have shown that the proposed RcFIP-based evolvable fuzzy hardware is able to handle the unpredictable scenarios in a real-time application, the ATM cell scheduling.

Chapter 6

Multi-Task Fuzzy Inference Processor

The hardware architecture of the RcFIP has been presented in 4.1. For convenience, the overall simplified architecture of the RcFIP is shown again in Fig. 6.1. The idea of the MtFIP is motivated by the online context switchable RcFIP with several desirable characteristics; real time reconfigurability, adaptation and high inference speed. It is capable of online context switching for a specific task. However, it would be desirable if the fuzzy inference processor can deliver high performance for multiple tasks concurrently and allow for seamless online context switching among the tasks. This is the main objective of the proposed MtFIP. As presented earlier in Section 1.4, the proposed MtFIP is expected to feature easy context switching, high flexibility, high inference performance and compact structure. These expectations are validated in this chapter.

Figure 6.1: RcFIP Brief Structure

Figure 6.2: Diagram of Multi-Tasking Architectural Framework

In principle, a MtFIP can be configured to handle several tasks concurrently. In this work, the architecture of the MtFIP is illustrated by considering 3 concurrent tasks. The first task considered is a rotary inverted pendulum balance control. The second is a truck backer upper control task while the third is a water bath temperature control task.

From a functional point of view, the block architecture for executing three tasks is as shown in Fig. 6.2. There are three RcFIPs incorporated, each of which is

capable of offering good performance for a specific task. This means that each task is executed by a pre-configured task-specific RcFIP, i.e. Task_1, Task_2 and Task_3 are executed by RcFIP_1, RcFIP_2 and RcFIP_3 respectively. On the whole, high performance can be maintained, as each task is executed in a manner similar to that of a single RcFIP. High flexibility is also achieved as the framework is easily scalable to accommodate the number of tasks to be handled.

In Fig. 6.2, the operations of the multiplexer and de-multiplexer are handled by the Task Manager. The Task Manager manages the various tasks by means of a rotation scheduling scheme. The tasks are handled concurrently on a time-sharing basis. This can be realized through multiplexing and de-multiplexing. Data derived from each task environment are digitized by input interfaces, such as ADC. Digital input signals are fed selectively into the corresponding RcFIP by the multiplexer and the de-multiplexer. Each RcFIP computes the control output for a particular task. The outputs are delivered to the corresponding task environment via the multiplexer and de-multiplexer again, as well as output interfaces, such as DAC. The whole structure allows for great flexibility to accommodate switching among several tasks. Data from the tasks are fed to the *Evolution Module* as training data. Each task can be serviced by a dedicated microcontroller-based GA unit. The evolutionary processes are triggered and the GA operations are carried out in the GA units. The derived contexts with the best fitness from each GA unit are updated into the CMU of the corresponding RcFIPs. The whole setup allows for online context switching among the tasks by incorporating multiple RcFIPs according to the number of tasks

to be handled.

6.1 Detailed Architecture of MtFIP

Referring to the architectural framework shown in Fig. 6.2, it is evident that integration of these RcFIPs into a single chip with current SoC technique will be beneficial. To achieve the integration of multiple RcFIPs, the overall architecture of the proposed MtFIP for 3 tasks is as shown in Fig. 6.3.

Figure 6.3: Overall Architecture of MtFIP

The MtFIP consists of six main functional blocks as described below.

- (1) FIM: the system's memory that maintains a compact structured representation of the fuzzy relational knowledge for multiple tasks.
- (2) Input Block: the input interface between multiple tasks and the MtFIP, which consists of an input register, a multiplexer and an AEM. It encodes input data from several tasks, for the FIM and the Task Manager.
- (3) Context Block: the context interface between the Evolution Module and the MtFIP. It consists of a context register, a multiplexer and a CMU. It handles context switching as well as generation of address to enable/disable the FIM partition blocks.
- (4) Output Block: the output interface which computes the final defuzzified output for the MtFIP and delivers the output to the corresponding task. It is made up of an OAM, a de-multiplexer and an output register.
- (5) Task Manager: the scheduler to execute the multiple tasks efficiently. It generates address for accessing each FIM partition block. It also manipulates the multiplexer of the Input Block and the Context Block using the task selection signal. The de-multiplexing at the Output Block is realized using the delayed task selection signal.
- (6) Control Unit: Overall synchronization of the MtFIP. This includes synchronization involving the MtFIP and external hardware interface, such as ADC and the FIM configuration interface.

The FIM is the key part in the design of the MtFIP. It is configured specific to the

tasks. As shown in Fig. 6.3, the whole FIM is divided into partition blocks. The FIM is partitioned such that the data stored in the T-1 memory-space correspond to the configuration data for Task_1, while data for Task_2 and Task_3 are stored in the T-2 and T-3 memory-space respectively. The FIM of the MtFIP integrates the functionality of all the RcFIPs' FIM. Each $mPB_{<i,j>}$ of the MtFIP shown in Fig. 6.3 integrates the corresponding $PB_{<i,j>}$ blocks of several RcFIPs, each of which is similar to the one shown in Fig. 6.1. The FIM of these three RcFIPs are maintained within T-1 to T-3 memory-space accordingly. In other words, the data for $PB_{<i,j>}$ of the first RcFIP is programmed in the T-1 memory-space of the MtFIP's $mPB_{<i,j>}$, the data for the second RcFIP is programmed in the T-2 memory-space of the MtFIP's $mPB_{<i,j>}$, and so on. Other parts of the RcFIPs, i.e., AEM, CMU, OAM and control unit are incorporated into the corresponding parts of the MtFIP. The details are presented in Sections 6.1.2, 6.1.3, 6.1.4 and 6.1.6.

The address to access the FIM consists of three parts as shown in Fig. 6.3; the first part being derived from the Task Manager, the second part being derived from the CMU, and the last part being derived from the AEM.

- (1) The Task Manager generates address to access T-1, T-2 and T-3 memory-space based on the task index. The incorporation of the task index into the address facilitates task switching during run-time.
- (2) Based on the current context of the task, the CMU generates the second part of the address. Any change in the context is handled by changing this part of

the address, making online context switching among multiple tasks convenient.

- (3) The third part of the address is generated by the AEM based on the inputs from the tasks.

Similar to that of the RcFIP, although the address lines feed into all the FIM partition blocks, only the partition blocks that are relevant to the context are accessed. The ones enabled are determined by the current context maintained in the CMU. The ENABLE signals derived by the CMU manipulate each FIM partition block respectively. The outputs of all the FIM partition blocks that are being accessed become the inputs to the OAM. The defuzzification is carried out in the OAM. The OAM of the MtFIP is similar to that of the RcFIP outlined in Section 4.1.4. The outcome of the OAM is the crisp final output of the MtFIP.

The Task Manager coordinates the tasks utilizing a rotation scheduling scheme. A task selection signal is derived to operate the multiplexer in the Input Block and the Context Block for task switching. To ensure that the output computed according to inputs from one task is assigned to the corresponding task, the de-multiplexer in the Output Block is synchronized by a selection signal. This selection signal is intentionally delayed by the Task Manager. The delay time is configured according to the maximal critical-path delay from the AEM to the OAM.

Similar to that of the RcFIP, the bi-directional data bus is utilized for data transfer to/from the FIM. The synchronization of the bi-directional data bus is coordinated by the control unit.

Each functional block is further described in the following sub-sections.

6.1.1 FIM

Figure 6.4: MtFIP’s FIM Configuration

The FIM essentially acts as the fuzzy relational knowledge base of the tasks. It is a representation that models the process of inputs fuzzification, rules evaluation and output composition. Fig. 6.4 shows the general layout of the FIM of the MtFIP. Similar to the FIM of the RcFIP, there are $(v \times w)$ partition blocks in the FIM of the

MtFIP; $mPB_{<1,1>}$, $mPB_{<1,2>}$, $mPB_{<1,3>}$, ..., $mPB_{<v,w>}$. The number of partition blocks required is determined based on the size of the linguistic sets for all the inputs of the tasks considered. For example, if the maximum required is 5 linguistic terms for one input, and 7 linguistic terms for the other among all the tasks, the number of partition blocks required in the FIM is 35.

Each FIM partition block consists of $(t+m+k_1+k_2)$ address bits as shown in Fig.

6.4. The scope of addressing is briefly described below:

- (1) The Task Manager derives t address bits to access the different memory-space in the partition blocks, i.e., T-1, T-2 and T-3. The width of this address field depends on the number of tasks executed concurrently by the MtFIP. In this case, t is equal to 2, since there are three tasks being considered.
- (2) As described in 4.1.2, the CMU maintains the current context in a CMU register with $(v \times w)$ segments. Each segment is represented by m bits. Based on the current context of the MtFIP, the CMU generates a m -bit address, each of which connects to only one partition block. Hence, the m -bit address derived from $E_{<1,1>}$ in the CMU connects to $mPB_{<1,1>}$ in the FIM, while the one derived from $E_{<1,2>}$ connects to $mPB_{<1,2>}$, and so on.
- (3) The number of bits k_1 and k_2 for inputs depend on the resolution of the input variables. The AEM takes in k_1 -bit and k_2 -bit inputs to generate a (k_1+k_2) -bit address. Hence, the complete address for accessing each FIM partition block is made up of $(t+m+k_1+k_2)$ bits.

6.1.2 Input Block

The Input Block which includes an input register, a multiplexer and an AEM shown in Fig. 6.5 acts as the input interface of the MtFIP. It receives the inputs from the tasks and stores them in the input register. In a practical scenario, all k_1 -bit and k_2 -bit inputs may be obtained from ADCs. These inputs derived from the tasks are maintained in sub-registers accordingly. Whenever new inputs are updated into each sub-register, the input controller pulls high the *iReady* bit of the status register stored in the Task Manager. The *iReady* signal is important since it forms part of the information required for the Task Manager to schedule the tasks. Detailed procedure for this are presented in Section 6.1.5. The *iReady* signals are derived by the Input Block, reporting on the current status of the inputs for the tasks.

Figure 6.5: Schematic of the Input Block

A t -bit task selection signal is derived from the Task Manager for the multiplexer to selectively load the inputs from the sub-registers into the AEM. Two inputs, *Input 1* and *Input 2* signals are fed into the AEM via the multiplexer. The AEM is the address encoding block of the MtFIP. It generates the address for accessing each

FIM partition block. During the normal operation mode, the $R/\overline{W}$ signal derived from the control unit is pulled high. The AEM takes in *Input 1* and *Input 2* data. Each pair of inputs generate a unique (k_1+k_2) -bit address. Hence, the (k_1+k_2) -bit address bits are derived from the Input Block.

6.1.3 Context Block

Figure 6.6: Schematic of the Context Block

The Context Block serves as the interface between the MtFIP and the *Evolution Module*. It consists of three parts as shown in Fig. 6.6; a context register, a multiplexer and a CMU. The contexts derived from the *Evolution Module* for three tasks are loaded into the context register as and when necessary. Each rule identifier in the context is represented by m bits. Hence, a context stored in the context register

for each task requires $(v \times w \times m)$ bits.

The Task Manager derives the same t -bit task selection signal to the Context Block and the Input Block. This synchronizes the MtFIP's loading of the inputs and context of the corresponding task for fuzzy inferencing. A selected context loaded into the CMU via the multiplexer is maintained in the CMU register.

6.1.4 Output Block

Figure 6.7: Schematic of the Output Block

The Output Block consists of an OAM, a de-multiplexer and an output register as shown in Fig. 6.7. A de-multiplexer is utilized to deliver the output to the corresponding task. A delayed task selection signal is derived from the Task Manager. The extent of delay required is equal to the critical path delay from the AEM to the OAM. From the de-multiplexer, the output is fed into the corresponding sub-register

in the output register. Each sub-register is dedicated to a specific task.

6.1.5 Task Manager

Figure 6.8: Architecture of the Task Manager

The Task Manager coordinates and schedules the execution sequence of the tasks. A status register is maintained for each task in the Task Manager as shown in Fig. 6.8. The Task Manager determines the tasks execution sequence based on information stored in the status registers. Each status register maintains a $(t+2)$ -bit data. The purpose of the various status bits is explained below.

- (1) The first bit represents the *iReady* status, indicating the readiness of inputs for a particular task. The input controller in the Input Block sets the *iReady* bit to high whenever new inputs are upgraded into the corresponding input sub-register. It denotes that this task is ready for further execution. The *iReady*

bit is cleared to '0' whenever the Task Manager finishes with the execution of this task. This means that the MtFIP has responded to this application and hence disables the corresponding application's *iReady* status bit.

- (2) The second bit in the status register denotes the execution priority for a task. A high *Priority* bit means that this task is allocated a high priority to be executed next. Among all the *Priority* bits of the status registers, only one *Priority* bit is being set to high at any time. Others are set to low denoting the lower priorities of execution. Generally speaking, the *Priority* bit of a task which is next to the newly accomplished task will be set to high. This task always gets the highest priority to be executed during the next clock cycle.
- (3) The last t bits represent the index of the task. The number of bits t is derived from Equation (6.1.1), which is based on the total number of tasks.

$$t = \begin{cases} \log_2 n, & (n = 2, 4, 8, 16, 32, \dots) \\ \lfloor \log_2 n \rfloor + 1, & (n = \text{others}) \end{cases} \quad (6.1.1)$$

where n is the total number of the tasks. In this work, n is chosen to be 3.

In the Task Manager, the *Priority* bits' setting for all the status registers is based on a rotating priority scheme. The first task is given the highest priority to be serviced initially. Its inputs on standby in the input register are fed into the MtFIP for fuzzy inferencing during the first clock cycle. The highest priority is then passed to the second task for the next round of task selection. However, if the inputs of the first and second tasks are not ready, while the third task is ready during the first clock

Figure 6.9: Working flow chart of Task Manager

cycle, then the third task is selected. The *Priority* bit of the fourth task is then set to high during the next clock cycle. After all these tasks are executed in turn, the highest priority is then passed back to the first task. The task selection depends not only on the *Priority* bits' setting, but also the *iReady* bits' setting in the status registers. According to the flow chart as shown in Fig. 6.9, the overall scheduling scheme utilized by the Task Manager is as follows:

- (1) All the *iReady* bits are set to low. The *Priority* bit for the first task is set to

high initially, while all others are low.

- (2) The input controller asserts the corresponding *iReady* bits to high whenever there are newly updated inputs loaded. Only tasks with high *iReady* bits are considered for scheduling.
- (3) Check the *iReady* bit of the task with *Priority* bit asserted high on every rising clock edge. If its *iReady* bit is high, then the task index is fed into the Task Index Register for further execution involving the FIM, Input Block, Context Block and Output Block, as shown in Fig. 6.8. The *iReady* bit and the *Priority* bit of the newly accomplished task are then reset to low. The *Priority* bit of the next task in the sequence is set to high for the next selection round.

However, if its *iReady* bit is low, then the next closest task in sequence whose *iReady* bit is high is selected for execution. The *iReady* bit of this newly accomplished task is then reset to low, and the priority is passed to the next one.

- (4) Repeat the procedure from Step(2) to Step(3) until the last task in the sequence has been executed. The *Priority* bit of the first task is then set to high.
- (5) Repeat the procedure from Step(2) to Step(4).

The Task Manager is triggered by the rising clock edge to select one task using the rotation scheduling scheme. After synthesis, the Task Manager is capable of finish

a scheduling every two clock cycles. If the MtFIP is synthesized utilizing 66 MHz operating frequency, this means that the Task Manager schedules one task in every 30 ns. Such a scheduling speed is fast enough to execute multiple tasks efficiently. In Fig. 6.8, the selected task's index serves as four functional control signals; as part of the address for accessing FIM partition blocks, the task selection signal for the multiplexer in the Input Block, the task selection signal for the Context Block and the delayed task selection signal for the Output Block. As described earlier in Section 6.1, the function of the delay buffer is for overall synchronization of inputs and outputs for each task. For example, it would be inappropriate if the MtFIP takes in the inputs derived from Task_A, while delivering the corresponding outputs to Task_B.

6.1.6 Control Unit

The control unit coordinates the synchronization of the MtFIP and synchronizes the operations involving the MtFIP and external hardware interfaces. From a functional viewpoint, the control unit of the MtFIP is the same as that of the RcFIP, as presented earlier in Sec. 4.1.5. Hence, the architecture of the control unit is also the same as the one shown in Fig. 4.10.

6.2 MtFIP Prototyping and Analysis

The VHDL prototyping of the MtFIP is developed in Mentor Graphics FPGA Advantage 6.3 platform. After circuit simulation and synthesis, it is implemented on Xilinx Virtex-II FPGA. The implementation steps are similar to that of the RcFIP reported in Section 4.1.6.

For the MtFIP prototype, there are 6 inputs, each to be 6-bit (i.e. two inputs from each task respectively). The number of bits m to code the *consequent* of each rule is chosen to be 3 bits ($m=3$). The width of the data bus is chosen to be 8 bits ($p=8$).

The resource utilization report is as shown in Table 6.1.

Device Utilization for XC2V2000BG575			
Resource	Used	Avail	Utilization
IOs	289	408	70.83%
Global Buffers	7	16	43.75%
Function Generators	20689	21504	96.21%
CLB Slices	10345	10752	96.21%
Dffs or Latches	1249	22728	5.50%

Table 6.1: MtFIP circuitry scale reports

The timing simulations of this MtFIP are based on 66 MHz clock in ModelSim. There are 4 clock cycles delay from the AEM to the OAM within the proposed MtFIP as shown in Fig. 6.3. Hence, the delayed task selection signal derived from the Task Manager is configured to delay 4 clock cycles than the task selection signal.

In order to evaluate the inferencing performance of the proposed MtFIP, the simulation waveforms for the following 3 situations are analyzed.

(Situation 1): This situation deals with newly updated inputs derived from 3 tasks simultaneously. How much time does MtFIP need to deal with these 3 requests?

(Situation 2): After responding to Situation 1, there are newly updated inputs from Task_2. How much time does MtFIP need to deal with this single request?

(Situation 3): This situation deals with both inputs updated and contexts switching. There are newly updated inputs derived from 3 tasks simultaneously. The context switching for these 3 tasks occur at the same time. How much time does MtFIP need in this situation?

The simulation waveforms of the MtFIP will be analyzed in the 3 following subsections.

6.2.1 Situation 1

The waveforms for Situation 1 are shown in Fig. 6.10. The inputs for 3 tasks are updated simultaneously at 1085 ns position (Cursor 1 in Fig. 6.10). Within the Task Manager of the MtFIP, Task_1 has the highest priority initially. Hence, the task selection signal derived from the Task Manager is assigned to Task_1 first, shown at the position of Cursor 2 (at 1128 ns position in Fig. 6.10).

The task selection signal enable the multiplexer of the Input Block and Context Block to take in the inputs and context of Task_1 for inferencing. As observed

Figure 6.10: Waveforms for inputs concurrent updates of 3 tasks (Situation 1)

in Fig. 6.10, the *Task_Manager_2_FIM_Address*, the *Context_Block_2_FIM_Address*, the *Input_Block_2_FIM_Address*, and the *Context_Block_2_FIM_Enable* signals are asserted after 1 clock cycle, based on the values of the inputs and the current context of Task_1. The data is derived from the FIM to the OAM at 1176 ns position (Cursor 3). Under the control of the delayed task selection signal, the Task_1 final output is derived from the MtFIP at 1208 ns position (Cursor 4). Two clock cycles later,

Figure 6.11: Waveforms for inputs updates of a single task (Situation 2)

the Task_2 final output is derived. At 1272 ns position (Cursor 5), the Task_3 final output is derived from the MtFIP.

In Situation 1, the first final output data for Task_1 is derived within 8 clock cycles. While the total execution time of the MtFIP for three tasks is 12 clock cycles, as shown in Fig. 6.10.

Figure 6.12: Waveforms for inputs and contexts updates of 3 tasks (Situation 3)

6.2.2 Situation 2

In Situation 2, only Task_2 has newly updated inputs. The waveforms of Situation 2 are shown in Fig. 6.11. The operation within the MtFIP is much simpler than Situation 1. In Fig. 6.11, the new inputs of Task_2 are updated at 2265 ns position (Cursor 1). The task selection signal is then derived from the Task Manager three clock cycles later. The *Priority* bit of Task_3 is asserted to indicate the highest

priority in the next round. The final output for Task_2 is derived from the MtFIP at 2392 ns position (Cursor 2). Hence, the total execution time of the MtFIP for a single task is 8 clock cycles as shown in Fig. 6.11.

6.2.3 Situation 3

In Situation 3, three tasks have new inputs updated and simultaneous context switching. The waveforms are shown in Fig. 6.12. The new inputs and new contexts are taken into the MtFIP at 3274 ns position (Cursor 1). This time, Task_3 wins the selection at 3320 ns position (Cursor 2), since Task_3 has the highest priority after Situation 2. It is observed in Fig. 6.12 that the total execution time for three tasks with both inputs and context switching is derived within 12 clock cycles.

6.2.4 Inference performance of the MtFIP

According to the simulation and synthesis results, the proposed MtFIP can provide around 8.3M FLIPS (i.e. $\frac{1}{8 \times 15ns}$) for a single task. It can provide around 5.6M FLIPS (i.e. $\frac{1}{12 \times 15ns}$) for multiple tasks concurrently.

The specification of this proposed MtFIP is listed in Table 6.2. Some high flexibility fuzzy inference processors and high inference performance processors reported in [85, 93, 117, 127, 132, 135] are also listed in the same table. From a performance viewpoint, the MtFIP aims to incorporate the advantages of both high flexibility and high inference performance. The fuzzy inference processors reported in [117, 135]

	MtFIP	[93] RcFIP	[132] 2003	[127] 2003	[117] 2001	[135] 2005	[85] 2005
Developmental Direction	Hybrid	Hybrid	High Performance	High Performance	High Flexibility	High Flexibility	High Performance
Processor Prototype	Xilinx VirtexII	Xilinx VirtexII	Xilinx VirtexII	Xilinx VirtexE	Xilinx XC4000	Micro- controller	ASIC
Number of Inputs	6	2	6	2	2	2	2
Number of Outputs	3	1	1	1	1	1	1
Performance (MFIPLS)	8.3 (single task) 5.6 (multi tasks)	17	3.5	12.5	--	--	7.0
Online Reconfigurability	Yes	Yes	No	No	No	No	No
Adaptability for EFH	Yes	Yes	No	No	No	No	No
Multi-tasking Capability	Yes	No	No	No	No	No	No

Table 6.2: MtFIP specifications and comparisons

fall into the high flexibility category, while the fuzzy processors in [85, 127, 132] fall into the high inference performance category. The MtFIP, RcFIP and the processors in [117, 127, 132] are implemented using FPGAs. The processor in [85] is implemented with ASIC, while the processor in [135] is implemented with Microcontroller.

From Table 6.2, it is observed that the proposed MtFIP offers good flexibility with more I/O capability and better fuzzy rules coverage. The inference speed of the MtFIP is still comparable with that of high inference performance aimed processors.

As the counterpart of the proposed MtFIP, multiple processors in a single chip approach has the outstanding advantage with true parallel executions for multiple tasks concurrently. Each processor can provide all its resources to execute a single task. High throughput and reliability can be achieved. The communications between each processor and the corresponding task are quite straightforward. However, the

communications and resource sharing among the processors are not so efficient. Usually there is a master processor or host to coordinate the information sharing among the processors. On the contrary, the MtFIP facilitates resource sharing. The potential tradeoff of the MtFIP is that the performance may not be as good as that of multiple processors in the single chip approach, since MtFIP executes multiple tasks on a time-sharing basis.

6.3 MtFIP Applications and Simulations

The practical multi-task functionality of the MtFIP is demonstrated in this section. The MtFIP is utilized to handle three tasks concurrently; a rotary inverted pendulum, a truck backer upper control task and a water bath temperature control. The context for each task is switched online. It is required to switch between various contexts and various tasks without interrupting the normal operation of the MtFIP. The MtFIP is able to handle successfully these three tasks concurrently, as well as supporting online context switching. This point is verified by the simulation results presented in this section.

6.3.1 Domain Knowledge

Task_1

A RcFIP has been employed to balance a PP-300 system by supporting online context switching in Sec. 5.2. This PP-300 balancing system serves as the first task of the MtFIP. The angular position of the pendulum corresponds to *Input 1*, while the angular velocity of the arm corresponds to *Input 2*. The motor driven voltage is the output of Task_1. Both input variables are described by the linguistic set {NH, NL, Z, PL, PH} while the output variable is described by {NF, NS, ZR, PS, PF}.

The membership functions for Task_1 are as shown in Fig. 6.13.

Figure 6.13: Membership Functions for Task_1

Task_2

The second task is for the truck backer upper parking. The truck is required to move backward a fixed distance at every time step until it arrives at the loading dock at right angle ($\phi=90^\circ$). Several control models employing fuzzy logic methodology have been reported, such as Kong and Kosko's model [156], Wang and Mendel's

model [157], Chen and Zhang's model [158], etc. In this work, Kong and Kosko's model [156] is adopted, except for some minor modifications to the membership functions. The truck's position on the horizontal axis, x is chosen as *Input 1*, which is described by linguistic set $\{LE, LC, CE, RC, RI\}$. *Input 2*, ϕ is the angle of the truck, which is described by $\{RB, RU, RV, VE, LV, LU, LB\}$. The steering angle θ is chosen as the output, described by $\{NB, NM, NS, ZE, PS, PM, PB\}$. The membership functions for Task_2 is as shown in Fig. 6.14.

Figure 6.14: Membership Functions for Task_2

Task_3

A water bath temperature control plant is chosen as the third task of the MtFIP. The membership functions for the input variables and output variable are as shown in Fig. 6.15. The *Input 1*, *Input 2* and output variables are described by the linguistic set $\{NL, ZE, PL, PM, PH\}$, $\{NL, ZE, PL, PM, PH\}$ and $\{HD, LD, NC, LI, HI\}$

respectively.

Figure 6.15: Membership Functions for Task_3

6.3.2 MtFIP Simulation Results

According to the linguistic terms sets defined for these three tasks as shown in Fig. 6.13, Fig. 6.14 and Fig. 6.15, the maximum size of the linguistic set for *Input 1* is 5, while the maximum size for that of *Input 2* is 7. Therefore, it is needed to cater for 35 partition blocks in the FIM. For the MtFIP example, both inputs are considered to be 6 bits ($k_1, k_2=6$). The number of bits derived from the CMU is chosen to be 3 bits ($m=3$). The width of the data bus is chosen to be 8 bits ($p=8$). Three tasks are executed concurrently.

For Task_1, there are 1000 execution steps for the entire control session, each of which takes place every 0.01 second. This means that *Context-1* of Task_1 has to be reloaded into the CMU of the MtFIP 1000 times within 10 seconds.

For Task_2, the process of the truck backer upper parking is started from a random initial position. It continues executing until it arrives at the loading dock with

the angle ϕ being 90° . The coordinates of the loading dock is (50,100). Hence, the number of execution steps, each of which takes place every 1 second, is not constant. It depends on the initial position of the truck, a random coordinates (x,y) within a $[0,100] \times [0,100]$ plane. In other words, the *Context-2* of Task_2 is reloaded into the CMU of the MtFIP M number of times within M seconds, where M is an integer. For Task_3, there are 100 execution steps, with each execution taking place every 30 seconds. Hence, *Context-3* of Task_3 needs to be activated on the MtFIP 100 times within 3000 seconds.

The task execution sequence is as shown in Fig. 6.16. The MtFIP executes three tasks concurrently during the whole procedure. The contexts of the tasks are loaded and switched $(M+1100)$ times into the CMU of the MtFIP during run time. The context switching among the tasks are easily handled by the system. Hence, the normal operation of the MtFIP is not interrupted at all during context switching.

Figure 6.16: Task Executing Sequence of MtFIP

The objective for Task_1 is to balance the pendulum under random disturbances applied to the inputs. The contexts are switched every execution step, i.e. every

Figure 6.17: Results for Task_1, Task_2 and Task_3

0.01 second. The simulation results are as shown in Fig. 6.17(a). The MtFIP executes Task_1 for 10 seconds. It is observed that the pendulum remains balanced, even when context switching is taking place.

For Task_2, the MtFIP is utilized to park the truck at the loading dock from a random initial position. The simulation results for a random case are as shown in Fig. 6.17(b). In this case, the initial position selected is (76.384,38.217) with the angle ϕ being $(-71.532)^\circ$. The context is switched every second. In this instance, it takes 111 seconds to park the truck at the loading dock. Hence, Task_2 is executed for 111 seconds for the plot shown in Fig. 6.17(b).

The MtFIP controls the temperature of the water bath for Task_3 within the desired temperature range of 25°C to 80°C . The associated context for this task is applied every 30 seconds. The results shown in Fig. 6.17(c) are based on simulation to achieve a desired target temperature of 67.5°C . From the plot shown in Fig. 6.17(c), the temperature increases from 28.772°C and stabilizes at 67.541°C .

Chapter 7

Conclusions

7.1 Conclusions

Evolvable fuzzy hardware is suitable for applications of real-time fuzzy control where the scenarios of the application may change dramatically. The real-time requirement necessitates the implementation of the fuzzy inference as embedded hardware system. In this respect, the main challenge is the implementation of hardware that allows for easy in-system reconfiguration. In this thesis, this challenge is overcome by means of a RcFIP which supports online context switching. Although a switch in context as a result of changes in membership functions is also possible, the proposed processor is designed to handle context switching brought about by the changes in the domain rules. More importantly, the RcFIP can offer very high inference throughput to support time-critical applications. With the aid of VHDL and EDA tools,

the prototype is realized in FPGA to demonstrate the functionality. The inferencing speed of the RcFIP achieved is up to 17M FLIPS.

A developmental framework has been outlined to support the realization of evolvable fuzzy hardware based on the RcFIP as the core processor. The key to the context independence of the RcFIP is the novel structuring of the FIM partition blocks, which implicitly encode the fuzzy relational knowledge and the inference model. Any change in the context or domain rule set will not affect the FIM configuration. It is only necessary to update the context in the RcFIP via a context register, which is relatively straightforward. In this respect, the bottleneck of reconfiguring the whole FIM is avoided, making it a viable alternative for realizing intrinsically evolvable fuzzy hardware systems.

Three examples to illustrate the functionality of the RcFIP are presented. The first example pertains to a water bath temperature control. It clearly illustrates the RcFIP's versatility in handling context-independent fuzzy inferencing. The second example is an RcFIP-based rotary inverted pendulum balancing system. It illustrates the viability of the RcFIP as a device to support online context switching. The third example of ATM cell scheduling is a real-time switching application. From the simulation results obtained, better QoS performance is observed. This observation validates the potential utility of the RcFIP as a context independent inference processor to support the implementation of evolvable fuzzy hardware. Based on the work presented in this thesis, a complete RcFIP-based evolvable fuzzy hardware system can be realized based on current system-on-chip technology.

The RcFIP incorporates the in-system reconfigurability, flexibility and high inference performance. It is a viable alternative for realizing a MtFIP with multi-tasking capability. The specifically configured architecture of the MtFIP can achieve similar inferencing performance as that of a RcFIP. The MtFIP endows the chip with the capacity to handle several tasks concurrently. The MtFIP has a compact structure as a result of the sharing of hardware resources. The FIM of the MtFIP covers the specific configuration data for these tasks in its partition blocks. The task's switching is regarded as a change of address signals accessing the FIM in the MtFIP.

The MtFIP is more versatile for its ability to handle context independent inferencing both in the same task and among several tasks. The implication of context independence is that it allows the processor to be programmed to handle any situation pertaining to the tasks without the need for offline reprogramming. With instantaneous context switching, situations where the task scenarios change dramatically can be handled seamlessly. Such ability of the MtFIP is demonstrated by executing three tasks concurrently. The context switchings among three tasks happen hundreds to thousands of times during the whole procedure, without interrupting the normal operations of the MtFIP. Based on observations from the simulation results, it can be concluded that the MtFIP is suitable for multi-tasking fuzzy inferencing.

7.2 Future Work

Future research and development can be carried out towards the ASIC (application specific integrated circuits) implementation and the practical applications of the RcFIP, as well as the MtFIP. The RcFIP and MtFIP should be further evaluated by extending their range of applications. To fully capitalize on the capability and unique features, applications or problem areas in a dynamic control environment should be the primary focus.

Chapter 8

Author's Publications

8.1 Journals and Book Chapters

1. M.H. Lim, Q. Cao, J.H. Li, and W.L. Ng, "Evolvable hardware using context switchable fuzzy inference processor," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 4, pp. 301–311, Jul. 2004.
2. Q. Cao, M.H. Lim, J.H. Li, Y.S. Ong, and W.L. Ng, "A context switchable fuzzy inference chip," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 4, pp. 552–567, Aug. 2006.
3. Q. Cao, M.H. Lim, K.V. Ling, J.H. Li, and C.W. Chan, "Online context switching for real time control applications using reconfigurable fuzzy inference chip," *International Journal of Computer Applications in Technology*, vol. 27, no. 2/3, pp. 153–165, 2006.

4. Q. Cao, M.H. Lim, X. Shi, and J.H. Li, "A real time multi-task fuzzy inference processor," *Submitted to IEEE Transactions on Fuzzy Systems*, Under review, 2006.
5. J.H. Li, M.H. Lim, and Q. Cao, "Evolvable fuzzy hardware for real-time embedded control for packet-switching," *Evolvable Machines: Theory & Practice*, Springer-Verlag, Heidelberg, vol. 161, pp. 205–227, 2004.
6. 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, Dec. 2005.
7. 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, Mar. 2006.

8.2 Conferences

1. 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*, pp. 835-839, Aug. 2004.
2. 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, Sep. 2004.

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*, pp. 686-690, 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.

8.3 Appendix

1. The paper titled “Evolvable hardware using context switchable fuzzy inference processor” covers Section 1.1, Chapter 3, and Chapter 5.
2. The paper titled “A context switchable fuzzy inference chip” covers Section 1.2, Chapter 2, Chapter 4, and Chapter 5.
3. The paper titled “Online context switching for real time control applications using reconfigurable fuzzy inference chip” covers Chapter 2, Section 4.1, and Section 5.2.
4. The paper titled “A real time multi-task fuzzy inference processor” covers Section 1.3, Chapter 6.

8.4 Special Acknowledgement on this Project

I would like to thank my senior, Mr. Li Ju Hui, for his helps on the RcFIP simulation of ATM cell scheduling, and his works on the evolution scheme. I also want to thank my senior, Mr. Ng Wil Lie, for his simulation results of his proposed model on Water Bath Temperature control, to facilitate the results comparisons between his model and my model.

Bibliography

- [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, Nov. 2002.
- [2] M. H. Lim, “Reconfigurable hardware architecture for evolvable fuzzy systems,” in *Proceedings of 9th International Symposium on Integrated Circuits, Devices & Systems*, pp. 18–22, Sep. 2001.
- [3] D. Keymeulen, K. Konada, M. Iwata, Y. Kuniyoshi, and T. Higuchi, “Robot learning using gate-level evolvable hardware,” in *Proceedings of 6th European Workshop on Learning Robots*, pp. 173–188, 1998.
- [4] 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, Dec. 2005.
- [5] J. H. Li and M. H. Lim, “Evolvable fuzzy system for ATM cell scheduling,” in *Proceedings of 5th International Conference on Evolvable Systems, LNCS 2606*, pp. 208–217, 2003.

-
- [6] 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.
- [7] 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.
- [8] H. Sakanashi, M. Iwata, and T. Higuchi, "Evolvable hardware for lossless compression of very high resolution bi-level images," *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 4, pp. 277–286, Jul. 2004.
- [9] X. Yao and T. Higuchi, "Promises and challenges of evolvable hardware," *IEEE Transactions on Systems, Man, and Cybernetics, Part C*, vol. 29, no. 1, pp. 87–97, Feb. 1999.
- [10] 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 Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 1, pp. 227–232, Feb. 2001.
- [11] Y. Zhang, S. L. Smith, and A. M. Tyrrell, "Digital circuit design using intrinsic evolvable hardware," in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2004*, pp. 24–26, 2004.

-
- [12] T. Higuchi, M. Iwata, D. Keymeulen, H. Sakanashi, M. Murakawa, I. Kajitani, E. Takahashi, K. Toda, N. Salami, N. Kajihara, and N. Otsu, “Real-world applications of analog and digital evolvable hardware,” *IEEE Transactions on Evolutionary Computation*, vol. 3, no. 3, pp. 220–235, Sep. 1999.
- [13] N. Forbes, “Evolution on a chip: evolvable hardware aims to optimize circuit design,” *IEEE Computational Science and Engineering*, vol. 3, no. 3, pp. 6–10, May 2001.
- [14] H. D. Garis, “Evolvable hardware: Genetic programming of a darwin machine,” in *Proceedings of the International Conference on Artificial Neural Nets and Genetic Algorithms*, pp. 441–449, 1993.
- [15] V. Sahni and V. P. Pyara, “An embryonic approach to reliable digital instrumentation based on evolvable hardware,” *IEEE Transactions on Instrumentation and Measurement*, vol. 52, no. 6, pp. 1696–1702, Dec. 2003.
- [16] T. Higuchi, H. Iba, and B. Manderick, “Evolvable hardware,” *Massively Parallel Artificial Intelligence*, R.F. Albrecht, C.R. Reeves and N.C. Steele (Eds), pp. 398–421, 1994.
- [17] J. C. Gallagher, S. Vigham, and G. Kramer, “A family of compact genetic algorithms for intrinsic evolvable hardware,” *IEEE Transactions on Evolutionary Computation*, vol. 8, no. 2, pp. 111–126, Apr. 2004.

-
- [18] A. M. Tyrrell, R. A. Krohling, and Y. Zhou, "Evolutionary algorithm for the promotion of evolvable hardware," *IEEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 4, pp. 267–275, Jul. 2004.
- [19] M. Tommiska and J. Vuori, "Implementation of genetic algorithms with programmable logic devices," in *Proceedings of 2nd Nordic Workshop on Genetic Algorithms and their Applications*, pp. 71–78, 1996.
- [20] A. P. Shanthi and R. Parthasarathi, "Exploring FPGA structures for evolving fault tolerant hardware," in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2003*, pp. 174–181, 2003.
- [21] D. Keymeulen, R. S. Zebulum, Y. Jin, and A. Stoica, "Fault-tolerant evolvable hardware using field-programmable transistor arrays," *IEEE Transactions on Reliability*, vol. 49, no. 3, pp. 305–316, Sep. 2000.
- [22] N. Raichman, R. Segev, and E. Ben-Jacob, "Evolvable hardware: genetic search in a physical realm," *Physica A: Statistical Mechanics and its Applications*, vol. 326, no. 1-2, pp. 265–285, Aug. 2003.
- [23] B. Shackleford, E. Okushi, M. Yasuda, K. H. K. Seo, T. Iwamoto, and H. Yasuura, "A high-performance hardware implementation of a survival-based genetic algorithm," in *Proceedings of the 1997 International Conference on Neural Information Processing and Intelligent Information Systems*, pp. 686–691, 1997.

-
- [24] C. C. Santini, J. F. M. Amaral, M. A. C. Pacheco, and R. Tanscheit, “Evolutionability and reconfigurability,” in *Proceedings of 2004 IEEE International Conference on Field-Programmable Technology*, pp. 105–112, 2004.
- [25] J. Holland, *Adaptation in Natural and Artificial Systems*. University of Michigan Press, 1975.
- [26] J. Holland, *Genetic Algorithms in Search, Optimization and Machine Learning*. Addison-Wesley, 1989.
- [27] L. J. Fogel, A. J. Owens, and M. J. Walsh, *Artificial Intelligence through a Simulation of Evolution*. John Wiley & Sons, 1966.
- [28] H. G. Beyer and H. P. Schwefel, “Evolution strategies: a comprehensive introduction,” *Natural Computing*, vol. 1, no. 1, pp. 3–52, May 2002.
- [29] P. Moscato, “Memetic algorithms: a short introduction,” *New Ideas in Optimization*, D. Corne and M. Dorigo and F. Glover (Eds), pp. 219–234, 1999.
- [30] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*. Springer, 2004.
- [31] J. R. Koza, *Genetic Programming: On the Programming of Computers by Means of Natural Selection*. MIT Press, 1992.
- [32] J. Kennedy and R. C. Eberhart, “Particle swarm optimization,” in *Proceedings of IEEE International Conference on Neural Networks*, pp. 1942–1948, 1995.

-
- [33] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*. Springer, 2003.
- [34] L. C. Jain and N. M. Martin, *Fusion of Neural Networks, Fuzzy Sets & Genetic Algorithms: Industrial Applications*. CRC Press, 1998.
- [35] A. Misevičius, T. Blažauskas, J. Blonskis, and J. Smolinskas, “An overview of some heuristic algorithms for combinatorial optimization problems,” *Information technology and control*, vol. 1, no. 30, pp. 21–31, 2004.
- [36] T. Kalganova, “Bidirectional incremental evolution in extrinsic evolvable hardware,” in *Proceedings of 2nd NASA/DoD Workshop on Evolvable Hardware*, pp. 65–74, 2000.
- [37] J. C. Gallagher, “A neuromorphic paradigm for extrinsically evolved hybrid analog/digital device controllers: initial explorations,” in *Proceedings of 3^d NASA/DoD Workshop on Evolvable Hardware*, pp. 48–55, 2001.
- [38] E. Sanchez, *Towards evolvable hardware: the evolutionary engineering approach*. New York: Springer, 1996.
- [39] I. Baradavka and T. Kalganova, “Assembling strategies in extrinsic evolvable hardware with bidirectional incremental evolution,” in *Proceedings of 6th European Conference on Genetic Programming*, pp. 276–285, 2003.
- [40] J. Hereford and D. Gwaltney, “Design space issues for intrinsic evolvable hardware,” in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2004*, pp. 231–234, 2004.

- [41] D. A. Gwaltney and M. I. Ferguson, "Intrinsic hardware evolution for the design and reconfiguration of analog speed controllers for a DC motor," in *Proceedings of NASA/DoD Conference on Evolvable Hardware-2003*, pp. 81–90, 2003.
- [42] M. H. Lim and W. L. Ng, "Iterative genetic algorithm for learning efficient fuzzy rule set," *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, vol. 17, no. 4, pp. 335–347, Sep. 2003.
- [43] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proceedings of 4th International Conference on Genetic Algorithms*, pp. 509–513, 1991.
- [44] M. H. Lim, S. Rahardja, and B. H. Gwee, "A GA paradigm for learning fuzzy rules," *Fuzzy Sets and Systems*, vol. 82, no. 2, pp. 177–186, Sep. 1996.
- [45] L. A. Zadeh, "Fuzzy sets," *Information and Control*, vol. 8, no. 3, pp. 338–353, Jun. 1965.
- [46] L. A. Zadeh, "Fuzzy logic," *IEEE Computer*, vol. 21, no. 4, pp. 83–93, Apr. 1988.
- [47] M. H. Lim, J. Y. Leong, and K. T. Lau, "A general framework for the development of PLD-based fuzzy controllers," *Journal of the Institution of Engineers*, vol. 35, no. 1, pp. 7–12, Feb. 1995.
- [48] N. E. Evmorforpoulos and J. N. Avaritsitis, "Adaptive digital fuzzy hardware in application-specific integrated circuits," in *Proceedings of 6th IEEE Inter-*

- national Conference on Electronics, Circuits and Systems*, pp. 1635–1638, Sep. 1999.
- [49] N. Al-Holou, T. Lahdhiri, D. Joo, J. Weaver, and F. Al-Abbas, “Sliding mode neural network inference fuzzy logic control for active suspension systems,” *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, pp. 234–246, Apr. 2002.
- [50] S. M. Yang, H. Li, M. Q. H. Meng, and P. X. Liu, “An embedded fuzzy controller for a behavior-based mobile robot with guaranteed performance,” *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 4, pp. 436–446, Aug. 2004.
- [51] M. H. Lim and Y. Takefuji, “Implementing fuzzy rule-based systems on silicon chips,” *IEEE Expert*, vol. 5, no. 1, pp. 31–45, Feb. 1990.
- [52] S. Mollov, R. Babuska, J. Abonyi, and H. B. Verbruggen, “Effective optimization for fuzzy model predictive control,” *IEEE Transactions on Fuzzy Systems*, vol. 12, no. 5, pp. 661–675, Oct. 2004.
- [53] G. Louverdis and I. Andreadis, “Design and implementation of a fuzzy hardware structure for morphological color image processing,” *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 13, no. 3, pp. 277–288, Mar. 2003.
- [54] C. C. Lee, “Fuzzy logic in control systems: Fuzzy logic controller - Part I,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 20, no. 2, pp. 404–418, Mar. 1990.

- [55] A. Costa, A. D. Gloria, P. Faraboschi, A. Pagni, and G. Rizzotto, "Hardware solutions for fuzzy control," *Proceedings of the IEEE, Special Issue on Fuzzy Logic*, vol. 83, no. 3, pp. 422–434, Mar. 1995.
- [56] G. Ascia, V. Catania, and M. Russo, "VLSI hardware architecture for complex fuzzy systems," *IEEE Transactions on Fuzzy Systems*, vol. 7, no. 5, pp. 553–570, Oct. 1999.
- [57] H. Eichfeld, T. Künemund, and M. Menke, "A 12b general-purpose fuzzy logic controller chip," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, pp. 460–475, Nov. 1996.
- [58] J. M. Jou, P. Y. Chen, and S. F. Yang, "An adaptive fuzzy logic controller: Its VLSI architecture and applications," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 8, no. 1, pp. 52–60, Feb. 2000.
- [59] M. Togai and H. Watanabe, "Expert system on a chip: An engine for real-time approximate reasoning," *IEEE Expert*, vol. 1, no. 3, pp. 55–62, Aug. 1986.
- [60] H. Watanabe, W. D. Dettloff, and K. E. Yount, "A VLSI fuzzy logic controller with reconfigurable, cascable architecture," *IEEE Journal of Solid-State Circuits*, vol. 25, no. 2, pp. 376–382, Apr. 1990.
- [61] A. Gabrielli and E. Gandolfi, "A fast digital fuzzy processor," *IEEE Micro*, vol. 19, no. 1, pp. 68–79, Jan. 1999.

- [62] K. Nakamura, N. Sakashita, Y. Nitta, K. Shimomura, and T. Tokuda, "A 12-bit resolution 200 kFLIPS fuzzy inference processor," *IEICE Transactions on Electronics*, vol. E76-C, no. 7, pp. 1102–1110, Jul. 1993.
- [63] D. L. Hung, "Dedicated digital fuzzy hardware," *IEEE Micro*, vol. 15, no. 4, pp. 31–39, Aug. 1995.
- [64] M. McKenna and B. M. Wilamowski, "Implementing a fuzzy system on a field programmable gate array," in *Proceedings of International Joint Conference on Neural Networks*, pp. 189–194, Jul. 2001.
- [65] M. A. Manzoul and D. Jayabharathi, "FPGA for fuzzy controllers," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 25, no. 1, pp. 213–216, Jan. 1995.
- [66] J. E. Bonilla, V. H. Grisales, and M. A. Melgarejo, "Genetic tuned FPGA based PD fuzzy LUT controller," in *Proceedings of 10th IEEE International Conference on Fuzzy Systems*, pp. 1084–1087, Dec. 2001.
- [67] H. Surmann and A. P. Ungering, "Fuzzy rule-based systems on general-purpose processors," *IEEE Micro*, vol. 15, no. 4, pp. 40–48, Aug. 1995.
- [68] R. Corder and J. Teichrow, "Mapping fuzzy systems into ROM structures," in *Proceedings of 3rd International Fuzzy Systems Association World Congress (IFSA)*, pp. 390–391, Aug. 1989.

- [69] M. A. Manzoul and D. Jayabharathi, "Fuzzy controller on fpga chip," in *Proceedings of IEEE International Conference on Fuzzy Systems, FUZZ-IEEE '92*, pp. 1309–1316, Mar. 1992.
- [70] L. E. Seward and J. A. Nachlas, "Availability analysis for multitask production systems," *International Journal of Flexible Manufacturing Systems*, vol. 16, no. 1, pp. 91–110, Jan. 2004.
- [71] H. Oh and S. Ha, "Hardware-software cosynthesis of multi-mode multi-task embedded systems with real-time constraints," in *proceedings of 10th International Symposium on Hardware/Software Codesign*, pp. 133–138, May 2002.
- [72] W. O. Cesario, D. Lyonnard, G. Nicolescu, Y. Paviot, S. Yoo, A. A. Jer-raya, L. Gauthier, and M. Diaz-Nava, "Multiprocessor SoC platforms: A component-based design approach," *IEEE Design & Test of Computers*, vol. 19, no. 6, pp. 52–63, Nov. 2002.
- [73] R. Bergamaschi, I. Bolsens, R. Gupta, R. Harr, A. Jerraya, K. Keutzer, K. Olukotun, K. Vissers, and W. Wolf, "Are single-chip multiprocessors in reach," *IEEE Design & Test of Computers*, vol. 18, no. 1, pp. 82–89, Jan. 2001.
- [74] P. G. Paulin, C. Pilkington, E. Bensoudane, M. Langevin, and D. Lyonnard, "Application of a multi-processor SoC platform to high-speed packet forwarding," in *Design, Automation and Test in Europe Conference and Exhibition Designers' Forum*, vol. 3, pp. 58–63, Feb. 2004.

- [75] B. D. Theelen, A. C. Verschueren, V. V. R. Surez, M. P. J. Stevens, and A. Nu, "A scalable single-chip multi-processor architecture with on-chip RTOS kernel," *Journal of Systems Architecture, Special Issue on Synthesis and Verification*, vol. 49, no. 12-15, pp. 619–639, Dec. 2003.
- [76] M. J. Rutten, J. T. J. v. Eijndhoven, E. G. T. Jaspers, P. v. d. Wolf, E. J. D. Pol, O. P. Gangwal, and A. Timmer, "A heterogeneous multiprocessor architecture for flexible media processing," *IEEE Design & Test of Computers*, vol. 19, no. 4, pp. 39–50, Jul. 2002.
- [77] D. Lyonnard, S. Yoo, A. Baghdadi, and A. A. Jerraya, "Automatic generation of application-specific architectures for heterogeneous multiprocessor system-on-chip," in *Proceedings of 38th Conference on Design Automation*, pp. 518–523, Jun. 2001.
- [78] D. Kim, "An implementation of fuzzy logic controller on the reconfigurable FPGA system," *IEEE Transactions on Industrial Electronics*, vol. 47, no. 3, pp. 703–715, Jun. 2000.
- [79] F. Chayab, L. Kirischian, and L. Szajek, "Architecture-to-task optimization system (ATOS) for parallel multi-mode data-flow architectures on a base of a partially reconfigurable computing platform," in *Proceedings of International Conference on Parallel Computing in Electrical Engineering*, pp. 27–32, Sep. 2002.

- [80] D. Falchieri, A. Gabrielli, and E. Gandolfi, “Very fast rate 2-input fuzzy processor for high energy physics,” *Fuzzy Sets and Systems*, vol. 132, no. 2, pp. 261–272, Dec. 2002.
- [81] G. Aranguren, L. A. L. Nozal, X. Basogain, J. L. Martín, and J. L. Arroyabe, “Hardware implementation of a pipeline fuzzy controller and software tools,” *Fuzzy Sets and Systems*, vol. 128, no. 1, pp. 61–79, May 2002.
- [82] S. Guo and L. Peters, “A high-speed fuzzy co-processor implemented in analogue/digital technique,” *Computers & Electrical Engineering*, vol. 24, no. 1-2, pp. 89–98, Jan. 1998.
- [83] M. H. Lim, Q. Cao, J. H. Li, and W. L. Ng, “Evolvable hardware using context switchable fuzzy inference processor,” *IEE Proceedings - Computers and Digital Techniques*, vol. 151, no. 4, pp. 301–311, Jul. 2004.
- [84] H. Peyravi, A. Khoei, and K. Hadidi, “Design of an analog CMOS fuzzy logic controller chip,” *Fuzzy Sets and Systems*, vol. 132, no. 2, pp. 245–260, Dec. 2002.
- [85] S.-H. Huang and J.-Y. Lai, “A high-speed VLSI fuzzy inference processor for trapeziod-shaped membership functions,” *Journal of Information Science and Engineering*, vol. 21, pp. 607–626, May 2005.
- [86] F. A. Samman and E. Y. Syamsuddin, “Programmable fuzzy logic controller circuit on CPLD chip,” in *Proceedings of Asia-Pacific Conference on Circuits and Systems, APCCAS’02*, vol. 2, pp. 561–564, Oct. 2002.

- [87] H. Eichfeld, M. Klimke, M. Menke, J. Nolles, and T. Kunemund, "A general-purpose fuzzy inference processor," *IEEE Micro*, vol. 15, no. 3, pp. 12–17, Jun. 1995.
- [88] H. Eichfeld, A. Mertens, T. Brandmeier, F. Waidelich, R. Graumann, and M. Schwab, "Applications of SAE 81C99x fuzzy coprocessors," in *Proceedings of 1998 IEEE International Conference On Fuzzy Systems*, vol. 1, pp. 19–24, May 1998.
- [89] K. Shimizu, M. Osumi, and F. Imae, "The digital fuzzy processor FP-5000," in *Proceedings of 2nd International Conference on Fuzzy Logic and Neural Networks*, pp. 539–542, Dec. 1992.
- [90] Available from: <http://www.atip.org/public/atip.reports.94/omron.94.html>.
- [91] H. Watanabe, W. D. Dettloff, and J. R. S. ad K. E. Yount, "VLSI fuzzy chip and inference accelerator board system," in *Proceedings of 21st International Symposium on Multiple-Valued Logic*, pp. 120–127, May 1991.
- [92] Q. Cao, M. H. Lim, K. V. Ling, J. H. Li, and C. W. Chan, "Online context switching for real time control applications using reconfigurable fuzzy inference chip," *To appear in International Journal of Computer Applications in Technology, Special Issue on Fuzzy Control Applications*, 2006.
- [93] Q. Cao, M. H. Lim, J. H. Li, Y. S. Ong, and W. L. Ng, "A context switchable fuzzy inference chip," *IEEE Transactions on Fuzzy Systems*, vol. 14, no. 4, pp. 552–567, Aug. 2006.

-
- [94] G. Lipari and S. K. Baruah, “Efficient scheduling of real-time multi-task applications in dynamic systems,” in *Proceedings of 6th IEEE Real-Time Technology and Applications Symposium (RTAS 2000)*, pp. 166–175, May 2000.
- [95] G. F. Marshall and S. Collins, “Fuzzy logic architecture using subthreshold analogue floating-gate devices,” *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 1, pp. 32–43, Feb. 1997.
- [96] S. H. Huang and J. Y. Lai, “A high-speed VLSI fuzzy logic controller with pipeline architecture,” in *Proceedings of 10th IEEE International Conference on Fuzzy Systems*, pp. 1054–1057, Dec. 2001.
- [97] Y. D. Kim and H. Lee-Kwang, “High speed flexible fuzzy hardware for fuzzy information processing,” *IEEE Transactions on Systems, Man and Cybernetics-Part A*, vol. 27, no. 1, pp. 45–56, Jan. 1997.
- [98] I. Baturone, S. Sánchez-Solano, A. Barriga, and J. L. Huertas, “Implementation of CMOS fuzzy controllers as mixed-signal integrated circuits,” *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 1, pp. 1–19, Feb. 1997.
- [99] R. Chidambaram, “VLSI based fuzzy logic controller enabled adaptive interactive multiple model for target tracking,” *Integration, the VLSI Journal*, vol. 35, no. 1, pp. 1–10, Jun. 2003.
- [100] C. Dualibe, P. Jespers, and M. Verleysen, “On designing mixed-signal programmable fuzzy logic controllers as embedded subsystems in standard CMOS

- technologies,” in *Proceedings of 14th Symposium on Integrated Circuits and System Design*, pp. 194–200, Sep. 2001.
- [101] M. J. Patyra and J. L. Grantner, “Hardware implementations of digital fuzzy logic controllers,” *Information Sciences*, vol. 113, no. 1-2, pp. 19–54, Jan. 1999.
- [102] A. Rodriguez-Vazquez, R. Navas, M. Delgado-Restituto, and F. Vidal-Verdu, “A modular programmable CMOS analog fuzzy controller chip,” *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 46, no. 3, pp. 251–265, Mar. 1999.
- [103] R. Amirkhanzadeh, A. Khoei, and K. Hadidi, “A mixed-signal current-mode fuzzy logic controller,” *International Journal of Electronics and Communications*, vol. 59, no. 3, pp. 177–184, Jun. 2005.
- [104] T. Yamakawa and T. Miki, “The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process,” *IEEE Transactions on Computers*, vol. C-35, no. 2, pp. 161–167, Feb. 1986.
- [105] T. Yamakawa, “A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control,” *IEEE Transactions on Neural Networks*, vol. 4, no. 3, pp. 496–522, May 1993.
- [106] C. Y. Chen, Y. T. Hsieh, and B. D. Liu, “Circuit implementation of linguistic-hedge fuzzy logic controller in current-mode approach,” *IEEE Transactions on Fuzzy Systems*, vol. 11, no. 5, pp. 624–646, Oct. 2003.

- [107] G. Feng, W. Zhang, and Y. F. Liu, "A new current mode fuzzy logic controller with extended state observer for DC-to-DC converters," in *Proceedings of 9th Nineteenth Annual IEEE Applied Power Electronics Conference and Exposition*, pp. 1771–1777, 2004.
- [108] E. Franchi, N. Manaresi, R. Rovatti, A. Bellini, and G. Baccarani, "Analog synthesis of nonlinear functions based on fuzzy logic," *IEEE Journal of Solid-State Circuits*, vol. 33, no. 6, pp. 885–895, Jun. 1998.
- [109] G. Lin and B. Shi, "A multi-input current-mode fuzzy integrated circuit for pattern recognition," in *Proceedings of 2nd International Conference on Intelligent Processing and Manufacturing of Materials*, pp. 687–693, Jul. 1999.
- [110] L. Lemaitre, M. J. Patyra, and D. Mlynek, "Analysis and design of CMOS fuzzy logic controller in current mode," *IEEE Journal of Solid-State Circuits*, vol. 29, no. 3, pp. 317–322, Mar. 1994.
- [111] S. Guo, L. Peters, and H. Surmann, "Design and application of an analog fuzzy logic controller," *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, pp. 429–438, Nov. 1996.
- [112] G. Giustolisi, G. Palmisano, and G. Palumbo, "An efficient fuzzy controller architecture in SC technique," *IEEE Transactions on Circuits and Systems II: Analog and Digital Signal Processing*, vol. 49, no. 3, pp. 208–218, Mar. 2002.

- [113] N. Manaresi, R. Rovatti, E. Franchi, R. Guerrieri, and G. Baccarani, “A silicon compiler of analog fuzzy controllers: From behavioral specifications to layout,” *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, pp. 418–428, Nov. 1996.
- [114] F. A. Samman and R. S. Sadjad, “Analog MOS circuit design for reconfigurable fuzzy logic controller,” in *Proceedings of 2002 Asia-Pacific Conference on Circuits and Systems*, pp. 151–156, Oct. 2002.
- [115] Z. Jaworski, M. Niewczas, M. Grygolec, and W. Kuzmich, “Architecture of a testable analog fuzzy logic controller,” *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, pp. 502–505, Nov. 1996.
- [116] N. M. Yu and J. Chen, “An analog fuzzy processor using neuron-MOS technology,” in *Proceedings of 7th International Conference on Solid-State and Integrated Circuits Technology*, pp. 1587–1590, Oct. 2004.
- [117] M. Gschwind, V. Salapura, and D. Maurer, “FPGA prototyping of a RISC processor core for embedded applications,” *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 9, no. 2, pp. 241–250, Apr. 2001.
- [118] S. Singh and K. S. Rattan, “Implementation of a fuzzy logic controller on an FPGA for a DC motor,” in *Electrical Insulation Conference and Electrical Manufacturing & Coil Winding Technology Conference*, pp. 387–394, Sep. 2003.
- [119] J. H. Li, T. H. S. Li, M. C. Tsai, and S. H. Lin, “Design and implementation of dynamic weighted fuzzy sliding-mode controller for an FPGA-based

- inverted pendulum car,” in *the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, pp. 628–633, Jul. 2003.
- [120] F. M. Yasin, A. Tio, M. S. Islam, M. I. Reaz, and M. S. Sulaiman, “The hardware design of temperature controller based on fuzzy logic for industrial application employing FPGA,” in *Proceedings of 16th International Conference on Microelectronics*, pp. 157–160, Dec. 2004.
- [121] A. Wojtasik, Z. Jaworski, W. Kumzicz, A. Wielgus, A. Walkanis, and D. Sarna, “Fuzzy logic controller for rate-adaptive heart pacemaker,” *Applied Soft Computing*, vol. 4, no. 3, pp. 259–270, Aug. 2004.
- [122] J. L. Huertas, S. Sanchez-Solano, I. Baturone, and A. Barriga, “Integrated circuit implementation of fuzzy controllers,” *IEEE Journal of Solid-State Circuits*, vol. 31, no. 7, pp. 1051–1058, Jul. 1996.
- [123] S. Orcioni, G. Biagetti, and M. Conti, “A mixed signal fuzzy controller using current mode circuits,” *Analog Integrated Circuits and Signal Processing, Special Issue on Current Mode Circuit Techniques*, vol. 38, no. 2-3, pp. 215–231, Feb. 2004.
- [124] J. Tombs, A. Torralba, and L. G. Franquelo, “Design of a fuzzy controller using mixed analog and digital techniques,” in *Proceedings of 3rd International Conference on Fuzzy Systems*, pp. 1755–1758, Dec. 1994.

- [125] V. Catania, A. Puliafito, M. Russo, and L. Vita, "A VLSI fuzzy inference processor based on a discrete analog approach," *IEEE Transactions on Fuzzy Systems*, vol. 2, no. 2, pp. 93–106, May 1994.
- [126] M. Sasaki, F. Ueno, and T. Inoue, "7.5 MFLIPS fuzzy microprocessor using SIMD and logic-in-memory structure," in *Proceedings of 2nd IEEE International Conference on Fuzzy Systems*, pp. 527–534, Sep. 1993.
- [127] A. Gaona, D. Olea, and M. Melgarejo, "Distributed arithmetic in the design of high speed hardware fuzzy inference systems," in *Proceedings of 22nd International Conference of the North American Fuzzy Information Processing Society*, pp. 116–120, Jul. 2003.
- [128] T. H. S. Li, S. J. Chang, and Y. X. Chen, "Implementation of human-like driving skills by autonomous fuzzy behavior control on an FPGA-based car-like mobile robot," *IEEE Transactions on Industrial Electronics*, vol. 50, no. 5, pp. 867–880, Oct. 2003.
- [129] C.-F. Juang and C.-H. Hsu, "Temperature control by chip-implemented adaptive recurrent fuzzy controller designed by evolutionary algorithm," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 52, pp. 2376–2384, Nov. 2005.
- [130] M. S. Masmoudi, I. Song, F. Karray, M. Masmoudi, and N. Derbel, "FPGA implementation of fuzzy wall-following control," in *Proceedings of 16th International Conference on Microelectronics*, pp. 133–139, Dec. 2004.

-
- [131] F.-J. Lin, D.-H. Wang, and P.-K. Huang, "FPGA-based fuzzy sliding-mode control for a linear induction motor drive," *IEEE Proceedings - Electric Power Applications*, vol. 152, pp. 1137–1148, Sep. 2005.
- [132] F. Homburg and R. Palomera-García, "A high-speed scalable and reconfigurable fuzzy controller," in *Proceedings of IEEE International Symposium on Circuits and Systems*, pp. 797–800, May. 2003.
- [133] S. P. J. V. Rani, P. Kanagasabapathy, and A. S. Kumar, "Digital fuzzy logic controller using VHDL," in *Proceedings of IEEE INDICON 2005 Conference*, pp. 463–466, Dec. 2005.
- [134] H. Jiang and D. Jin, "Self-adaptation of fuzzy controller optimized by hardware-based GA," in *Proceedings of 6th IEEE International Conference on Solid-State and Integrated-Circuit Technology*, pp. 1147–1150, Oct. 2001.
- [135] S. Pravadaliloglu, "Single-chip fuzzy logic controller design and an application on a permanent magnet dc motor," *Engineering Applications of Artificial Intelligence*, vol. 18, pp. 881–890, Oct. 2005.
- [136] V. Salapura, "A fuzzy RISC processor," *IEEE Transactions on Fuzzy Systems*, vol. 8, no. 6, pp. 781–790, Dec 2000.
- [137] D. G. Zrilic, J. Ramirez-Angulo, and B. Yuan, "Hardware implementations of fuzzy membership functions, operations, and inference," *Computers & Electrical Engineering*, vol. 26, no. 1, pp. 85–105, Jan. 2000.

-
- [138] O. Cordón, F. Herrera, F. Hoffmann, and L. Magdalena, *Genetic Fuzzy Systems: Evolutionary Tuning and Learning of Fuzzy Knowledge Bases*. World Scientific, Singapore, 2001.
- [139] M. J. Patyra, J. L. Grantner, and K. Koster, “Digital fuzzy logic controller: Design and implementation,” *IEEE Transactions on Fuzzy Systems*, vol. 4, no. 4, pp. 439–459, Nov. 1996.
- [140] Q. Cao, M. H. Lim, and J. H. Li, “A novel aggregation circuit for reconfigurable fuzzy inference chip,” in *Proceedings of 2nd International Conference on Artificial Intelligence in Engineering and Technology*, pp. 835–839, Aug. 2004.
- [141] Xilinx Inc., *Virtex-II Platform FPGAs: Complete Data Sheet*, Ver. 3.3, Jun. 2004.
- [142] D. Kim, “A design of CMAC-based fuzzy logic controller with fast learning and accurate approximation,” *Fuzzy Sets and Systems*, vol. 125, no. 1, pp. 93–104, Jan. 2002.
- [143] B. Kosko, *Neural Networks and Fuzzy Systems*. Prentice Hall, Englewood Cliffs, NJ, 1992.
- [144] J. Tanomaru and S. Omatu, “Process control by on-line trained neural controllers,” *IEEE Transactions on Industrial Electronics*, vol. 39, no. 6, pp. 511–521, Dec. 1992.

- [145] C. H. Wang, T. C. Lin, T. T. Lee, and H. L. Liu, "Adaptive hybrid intelligent control for uncertain nonlinear dynamical systems," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 32, no. 5, pp. 583–597, Oct. 2002.
- [146] R. J. Wai, J. D. Lee, and L. J. Chang, "Development of adaptive sliding-mode control for nonlinear dual-axis inverted-pendulum system," in *Proceedings of the 2003 IEEE/ASME International Conference on Advanced Intelligent Mechatronics (AIM 2003)*, pp. 815–820, Jul. 2003.
- [147] L. X. Wang, "Stable adaptive fuzzy controllers with application to inverted pendulum tracking," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 26, no. 5, pp. 677–691, Oct. 1996.
- [148] H. K. Lam, F. H. Leung, and P. K. S. Tan, "Design and stability analysis of fuzzy model-based nonlinear controller for nonlinear systems using genetic algorithm," *IEEE Transactions on Systems, Man and Cybernetics, Part B: Cybernetics*, vol. 33, no. 2, pp. 250–257, Apr. 2003.
- [149] M. Hojati and S. Gazor, "Hybrid adaptive fuzzy identification and control of nonlinear systems," *IEEE Transactions on Fuzzy Systems*, vol. 10, no. 2, pp. 198–210, Apr. 2002.
- [150] KentRidge Instrument Pte Ltd., *KRi Inverted Pendulum Model PP300: User and Service Manual*, Ver. 2.2, Feb. 2000.

-
- [151] KentRidge Instrument Pte Ltd., *Universal Controller UC-96 User Guide & Specification*, Ver. 2.0, 1999.
- [152] Mircochip Technology Inc., *MCP3201C/PI Analogue-to-Digital Converter Data Sheet*, 1999.
- [153] Celoxica Ltd., *RC-200 Hardware and Installation Manual*, Ver. 2.1, 2003.
- [154] 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*, pp. 36–43, 1999.
- [155] R. Händel, M. N. Huber, and S. Schröder, *ATM networks: concpets, protocols, applications*. Addison-Wesley, 1998.
- [156] S.-G. Kong and B. Kosko, “Adaptive fuzzy systems for backing up a truck-and-trailer,” *IEEE Transactions on Neural Networks*, vol. 3, no. 2, pp. 211–223, Mar. 1992.
- [157] L. X. Wang and J. M. Mendel, “Generating fuzzy rules by learning from examples,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 22, no. 6, pp. 1414–1427, Nov.-Dec. 1992.
- [158] G. Chen and D. Zhang, “Back-driving a truck with suboptimal distance trajectories: a fuzzy logic control approach,” *IEEE Transactions on Fuzzy Systems*, vol. 5, no. 3, pp. 369–380, Aug. 1997.