This document is downloaded from DR-NTU, Nanyang Technological University Library, Singapore.

Title	A context switchable fuzzy inference chip(Published version)
Author(s)	Cao, Qi; Lim, Meng-Hiot; Li, Ju Hui; Ong, Yew Soon; Ng, Wil Lie
Citation	Cao, Q., Lim, M. H., Li, J. H., Ong, Y. S., & Ng, W. L. (2006). A context switchable fuzzy inference chip. IEEE Transactions on Fuzzy Systems, 14(4), 552-567.
Date	2006
URL	http://hdl.handle.net/10220/4636
Rights	© 2006 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE. This material is presented to ensure timely dissemination of scholarly and technical work. Copyright and all rights therein are retained by authors or by other copyright holders. All persons copying this information are expected to adhere to the terms and constraints invoked by each author's copyright. In most cases, these works may not be reposted without the explicit permission of the copyright holder. http://www.ieee.org/portal/site.

A Context Switchable Fuzzy Inference Chip

Qi Cao, Meng Hiot Lim, Ju Hui Li, Yew Soon Ong, *Member, IEEE*, and Wil Lie Ng

***Abstract*—This paper describes a novel design of a fuzzy inference chip that allows for real-time online context switching. A context refers to a situation or scenario of an application requiring specific domain knowledge. In particular, our focus is on the class of applications involving embedded fuzzy control. The domain knowledge therefore refers to fuzzy rules and memberships. The kind of applications being considered is real-time in nature, which necessitates the implementation of hardware for fuzzy inferencing. The chip architecture is described and details on the design of the chip is presented.**

***Index Terms*—Context switching, embedded fuzzy control, evolvable systems, fuzzy inference, reconfigurable fuzzy chip.**

I. Introduction

FUZZY systems have been an active area of research since the conceptualization of fuzzy set theory by Zadeh [1][2]. There have been successful applications in many areas of control. Some examples include washing machine [3], air-conditioner [4], automobile control [5], robotics [6], financial loan management [7], model predictive control [8], and image and speech processing [9][10]. Fuzzy systems are efficient in dealing with complex, nonlinear and time-varying processes whose mathematical models are difficult to describe [11].

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 and depending on the speed requirement, a suitable architecture with an appropriate level of parallelism can be configured to achieve the desired processing speed [12]. This type of hardware solution employing dedicated microelectronic circuits to realize the system is referred to as a fuzzy inference chip.

The motivation for the initial design of a fuzzy inference chip by Togai and Watanabe [13] was its suitability for applications requiring high inference speed. To realize an application, domain specific knowledge is programmed into the fuzzy chip beforehand. The system output is derived through inferencing based on the inputs applied and the domain knowledge. A typical fuzzy inference chip 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 chip.

For conventional fuzzy systems, the rules are usually fixed regardless of the change in operating conditions. This may not be efficient for some problems where scenarios may change dramatically. In order to achieve consistent performance for the system, evolvable fuzzy hardware could be a viable option. Evolvable fuzzy hardware is an attractive alternative since it allows the architecture of a hardware system to be altered accordingly to suit the requirements of the operating environment. Certain applications of real-time fuzzy control may require a hardware solution towards fuzzy inference. If the domain rules and memberships are static, then it is usually not an issue since extensive work on fuzzy inference chips has addressed this aspect adequately. On the other hand, if the domain rules or possibly the membership functions are dynamic, it is imperative that the hardware that handles the fuzzy inference lends itself to reconfiguration. For this purpose, we design a reconfigurable fuzzy inference chip (RFIC), which allows for online context switching. A switch in the context refers to a change in either the rule set or the membership functions. It should be emphasized that in principle, the changes of both the rule set and memberships can occur simultaneously. In practice, experience has shown that evolving the rule sets with fixed membership usually offers sufficient coverage for functional adaptability of fuzzy systems, a notion that was originally described by Thrift [14] and Lim *et al.* [15]. Hence, in our scheme, we consider the updating of context to involve only the rule set.

In the next section, we give a general overview of fuzzy inference chips. In Section III, we describe the architecture of the RFIC. In Section IV, a detailed illustration on the process of configuring the RFIC is presented. The method of inferencing and partitioning of the rule blocks is described in details. In Section V, we simulate the functionality of the RFIC based on an actual control application. To further demonstrate online context switching capability of the RFIC, we simulate an evolvable fuzzy hardware system with the proposed RFIC as the core-processor. We conclude this paper in Section VI.

II. Fuzzy Inference Chips Review

Issues pertaining to hardware implementation of fuzzy systems have been covered extensively in literature [3], [4], [7], [13], [16]–[27]. Fuzzy inference chip was first proposed by Togai *et al.* [13] in 1986, while Yamakawa *et al.* designed the first analog fuzzy chip [21]. Analog fuzzy chips yield high inference speed with relatively simple and compact architecture,

Manuscript received September 23, 2003; revised February 8, 2005 and October 17, 2005.

Q. Cao and J. H. Li are with the Centre for Integrated Circuits and Systems, the School of EEE, Nanyang Technological University, Singapore 639798, Singapore.

M. H. Lim is with the School of EEE, Nanyang Technological University, Singapore 639798, Singapore.

Y. S. Ong is with the School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore.

W. L. Ng is with the Centre for Advanced Information Systems, the School of Computer Engineering, Nanyang Technological University, Singapore 639798, Singapore.

Digital Object Identifier 10.1109/TFUZZ.2006.876735

as well as good compatibility with sensors [24], [28]. However, analog implementation lacks extensibility and flexibility compared to digital implementation [29]. Digital fuzzy chips have good programmability and flexibility by employing logic and memory circuits, as well as good compatibility with other digital systems [24], [29]. Nowadays, with powerful formal design methodology and good design tools support, the realization of digital systems is generally more efficient. Digital fuzzy chips are also popular since development and prototyping can be achieved in a relatively short time.

Over the years, digital fuzzy inference chips have evolved significantly. Initial implementation of Togai's inference chip accommodates fuzzy subset of 31 elements. The membership value of each element is discretized using four binary bits. All 16 rules can be processed in parallel. The fuzzy chip can perform approximately 80 000 fuzzy logic inferences per second (80 K FLIPS). Watanabe *et al.* [18] continued to enhance the fuzzy chip. With the memory size and inference speed upgraded, the universe of discourse of fuzzy subset is enlarged to 64 elements and up to 102 fuzzy rules can operate in parallel. The chip can perform up to 580 K 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 chip is up to 7.5 M FLIPS as reported in [20] and 10 M FLIPS in [16], [17]. High density, flexible architecture, and availability of devices with good programmability feature such as FPGA, offer even greater versatility for system implementation. A potential inference speed of up to 60 M FLIPS was reported in [30] by employing the Xilinx XC3000-family FPGA. Besides improvement in inference speed, higher resolution for fuzzy chips 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 chip [11], while the input resolution of 12-bit with membership functions resolution of 8-bit have been reported in [23]. These fuzzy chips with high inference throughput can support rigorous real-time applications [31]. For example, these chips 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.

Usually, there are two ways to realize a digital fuzzy chip. 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 is a form of mapping programmed into memory elements and together with the hardware inference circuitry form the basis of a fuzzy chip system. This is a form of explicit implementation of the fuzzy inference systems. For example, the adaptive fuzzy hardware system depicted in [25] 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 divider 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 [7], [13], [18], [19], [23], and [26].

Another way to realize digital fuzzy inference chip 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 chip. The fuzzy inference process is then carried out based on the inputs and the preprogrammed mapping. For example, Lim *et al.* [3] 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 able to support only 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 [22], [27], and [30].

For fuzzy inference chip, designing the defuzzification block is a challenge because it is a significant bottleneck in achieving high fuzzy inference speed [32]. Normally, defuzzification process requires multiplication and division, making the fuzzy inference chip complicated and complex. One design approach by Watanabe *et al.* [18] 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.* [25] also designed similar circuit. The defuzzifier circuit enhances the performance of their digital fuzzy inference chips, although a divider is still necessary. Yamakawa [24] designed a defuzzification circuit to eliminate the divider in his analog fuzzy inference chip. 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 [4], [26]. For example, a (16×8) ROM-type multiplier lookup table and a (24×9) ROM-type division lookup table are employed in [4]. 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.

III. RFIC

The design of the RFIC is generally more complicated than either implicit or explicit implementation approaches. It involves a combination of methodologies employed in the implicit implementation approach as well as explicit implementation approach to support online context switching. Using a structured scheme of multiple rule level mappings, we derive a formatted memory map, which we refer to as a fuzzy inference mapping (FIM). For the scheme to work, a novel methodology of conflict resolution or aggregation is incorporated. This is

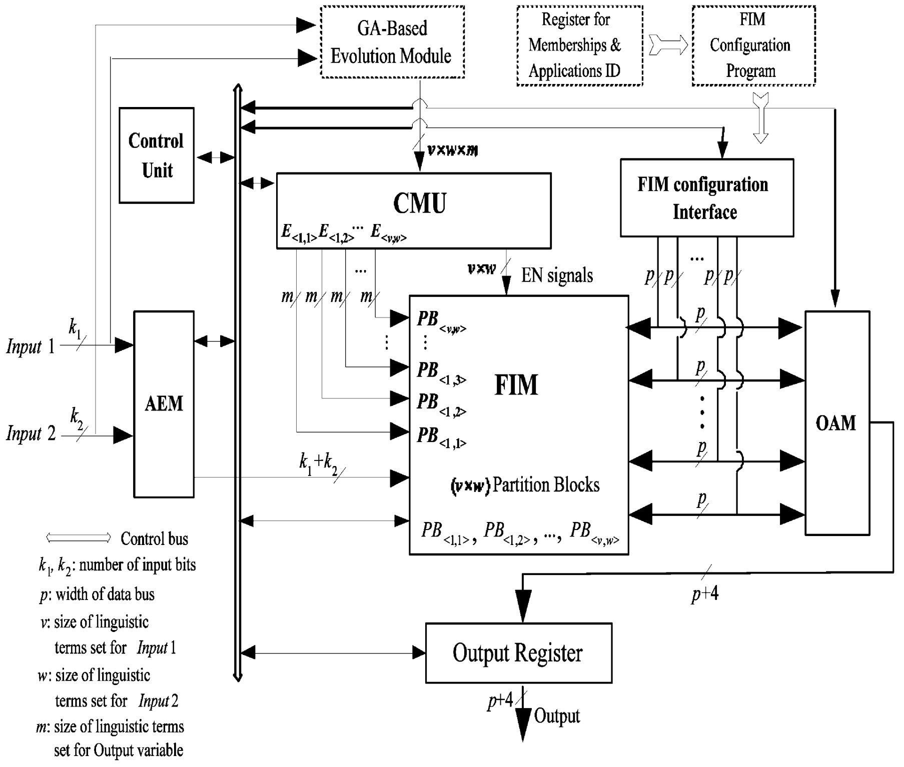

GA-Based Evolution Module
Register for Memberships & Applications ID
FIM Configuration Program
Control Unit
v×w×m
CMU
E<1,1> E<1,2> ... E<v,w>
FIM configuration Interface
v×w
EN signals
m m m m
p p p p
PB<v,w>
PB<1,3>
PB<1,2>
PB<1,1>
FIM
(v×w) Partition Blocks
PB<1,1>, PB<1,2>, ..., PB<v,w>
Input 1
Input 2
k1
k2
AEM
k1+k2
OAM
p+4
Output Register
Output
Control bus
k1, k2: number of input bits
p: width of data bus
v: size of linguistic terms set for Input 1
w: size of linguistic terms set for Input 2
m: size of linguistic terms set for Output variable

Fig. 1. Block architecture of RFIC.

analogous to the process of defuzzification, a necessary step in the explicit implementation of fuzzy reasoning hardware.

For fuzzy hardware, the inference engine deduces the fuzzy output based on the system's inputs, rules and membership functions. Within the RFIC, the key to fuzzy inference is the FIM. The FIM is a representation that covers all the contexts for a fuzzy application. The adaptation of the RFIC to various contexts is handled by the partition blocks in the FIM. The relevant partition blocks in the FIM are activated according to the context of the problem and the inputs to the system. From the outputs of the FIM partition blocks, a deterministic action is derived. This represents the output of the RFIC.

There are two working stages of the RFIC, 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 our scheme, since the membership functions are fixed, the possible rule set represents the context, which is applied to the RFIC. 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.

Once the FIM of the RFIC is configured, there is no need to change its configuration unless there is a change in the membership functions. The new FIM configuration data set is generated only if a context is updated due to a change in the membership functions. FIM reconfiguration is not necessary for switching of context due to a change in domain rules. Hence, the RFIC is effective in supporting context switching for different operating environment without interrupting the normal inferencing process. The hardware architecture of the RFIC is described next.

A. Architecture

Fig. 1 shows the overall block architecture of the RFIC. It consists of five basic function blocks; FIM, context management unit (CMU), address encoder module (AEM), output aggregation module (OAM), and control unit.

The FIM in the RFIC is a structured representation of the fuzzy relational knowledge. 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 two-input system to be v and w. Then in general, there are $(v \times w)$ partition blocks in the FIM, $\mathrm{PB}_{\langle 1,1\rangle}, \mathrm{PB}_{\langle 1,2\rangle}, \mathrm{PB}_{\langle 1,3\rangle}, \ldots, \mathrm{PB}_{\langle v,w\rangle}$. For example, if each input variable is characterized by five 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. The ones enabled are determined by the current context maintained in the CMU.

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. There are $(v \times w)$ segments in the context register, $E_{<1,1>}, E_{<1,2>}, E_{<1,3>}, \ldots, E_{<v,w>}$. 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

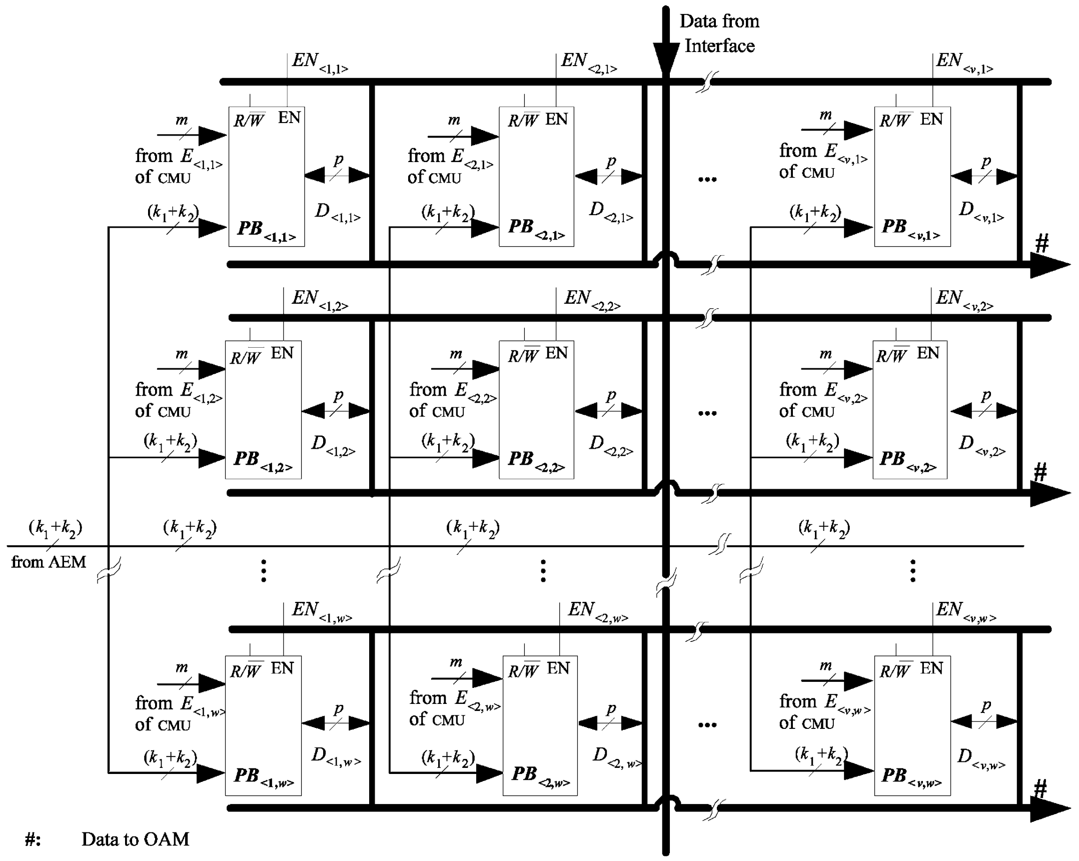


Fig. 2. FIM configuration.

only one partition block. Hence, the m-bit address derived from $E_{<1,1>}$ in the CMU connects to $\mathrm{PB}_{<1,1>}$ in the FIM, while the m-bit address derived from $E_{<1,2>}$ connects to $\mathrm{PB}_{<1,2>}$ 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 analog-to-digital converters. For example, choosing both k_1 and k_2 to be four 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, we design a novel circuit 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 RFIC.

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 RFIC. 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.

B. FIM

Fig. 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 $(\mathrm{PB}_{<1,1>}, \mathrm{PB}_{<1,2>}\mathrm{PB}_{<1,3>}\ldots,$ $\mathrm{PB}_{<v,w>})$ and the total number of partition blocks is determined by the size of the input linguistic terms set. In this case, we denote the size 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 earlier, 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, RFIC 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, we argue that it is not as useful as a context change brought about by a change in the domain rules [14], [15].

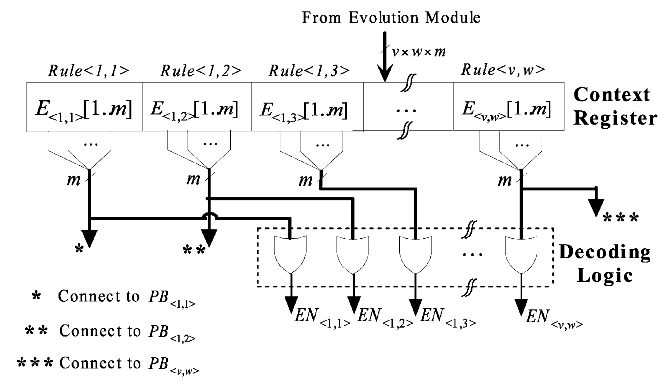


Fig. 3. Hardware architecture of CMU.

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. 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.

C. CMU

With fixed membership functions, a context is an instance of rule set. The length of the rule set depends on the number of membership functions defined for the input variables. The rule set is loaded directly into the context register of the CMU. The circuit block diagram of the CMU is as shown in Fig. 3. 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 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 $< 1,1 >$, Rule $< 1,2 >$, Rule $< 1,3 >$ $,\ldots,$ Rule $< v,w >$. 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.

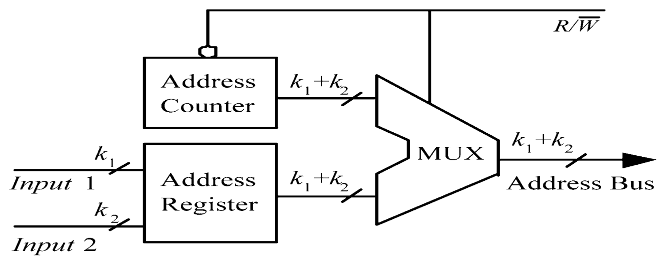


Fig. 4. Schematic of AEM.

D. AEM

The AEM as shown in Fig. 4 is the address encoding block of the RFIC. It generates the address for accessing the partition blocks in the FIM. During the normal operation mode, the $R/\bar{W}$ signal is pulled high. The AEM takes in k_1-bit and k_2-bit data. During the configuration mode, the $R/\bar{W}$ signal is low. This causes the 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 RFIC, 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 enabled by the CMU decoding logic are accessed.

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. Further details on this aspect are given in Section IV by means of an illustrative example.

E. OAM

The defuzzification operation is handled by the OAM circuit. The method of aggregation adopted for the RFIC is a form of rule level defuzzification averaging [33]. 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 RFIC, 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

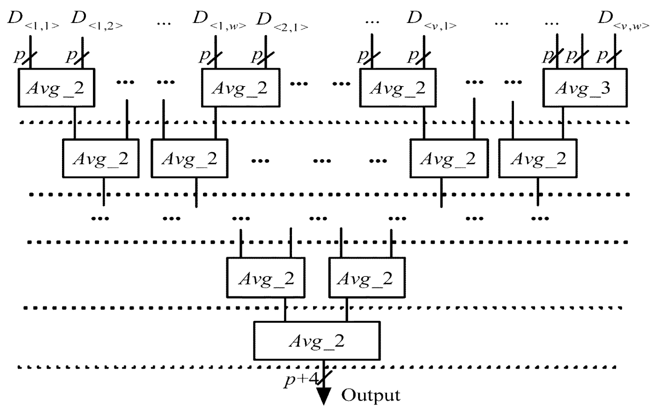


Fig. 5. Hardware architecture of OAM.

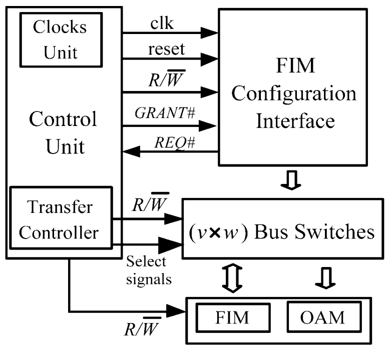


Fig. 6. Hardware block architecture of control unit.

output of $\mathrm{PB}_{<i,j>}$ in the FIM. The general form of the equation for averaging is as follows:

$$Z = \frac{\sum_{i=1}^{v}\sum_{j=1}^{w} D_{ij}}{N}, \qquad (1 \le N \le v \times w) \tag{1}$$

where N is the number of active partition blocks. The output Z from (1) is the final output of the RFIC.

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, we adopt a hierarchical averaging approach in order to manage the potential complexity of the circuit. Hierarchical averaging involves multiple stages of computation as illustrated in Fig. 5. In the circuit, we use mostly Avg_2 circuit blocks, 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. 5, the extent of the error compared to direct averaging based on (1) is not significant. From simulations on control applications presented in Section V, the error due to hierarchical averaging has no noticeable effect on the functional performance of the RFIC.

F. Control Unit

The overall synchronization of the RFIC is carried out by the control unit. Besides generating the clock signals, the control unit also generates signals to synchronize operations involving the RFIC and external hardware interface, such as ADC and the FIM configuration interface. The fuzzy logic inferencing is performed during the RFIC 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 RFIC configuration process. The overall handshaking protocol between the control unit and the FIM configuration interface is illustrated in Fig. 6.

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 by one sequentially. The first FIM partition block configured is $\mathrm{PB}_{<1,1>}$. Data is transferred from the FIM configuration interface to $\mathrm{PB}_{<1,1>}$ without interruption. Next, $\mathrm{PB}_{<1,2>}$ is configured, followed by $\mathrm{PB}_{<1,3>}$. This continues until the last block $\mathrm{PB}_{<v,w>}$ 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. 6, there are $(v \times w)$ bus switches which are controlled by the transfer controller. 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.

G. Circuit Prototyping

Circuit realization for the RFIC is carried out using VHDL (very high-speed integrated circuit hardware description language) in Mentor Graphics FPGA Advantage 6.3 environment. A Xilinx XC2V2000 Virtex-II FPGA is employed for circuit prototyping. For our RFIC prototyping example, we consider 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 device utilization and critical path report is as shown in Table I. The critical path delay is 57.38 ns (from the AEM to the OAM), out of which 41.8 ns is delay attributed to the OAM. Based on this critical path delay, the inference speed of the RFIC is estimated to be 17 M FLIPS. This inference speed is high enough for most time-critical applications.

H. Evolvable Hardware System

An example of RFIC-based evolvable hardware system configuration is illustrated by the block architecture in Fig. 7. The *Evolution Module* can be realized by a microcontroller-based

TABLE I
CIRCUITRY AREA AND DELAY REPORTS

Device Utilization for XC2V2000BG575			
Resource	Used	Avail	Utilization
IOs	8	408	1.96%
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%
Critical Path Report			
Propagate Clock Delay	4.89 ns (worst case)		
Ideal Data Arrival Time	52.49 ns		
Data Arrival Time	57.38 ns		
Data Required Time	57.68 ns		
Slack	0.30 ns		

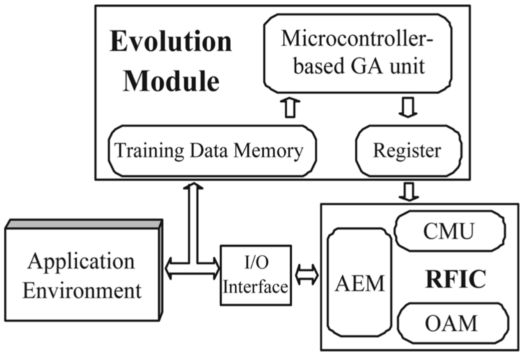


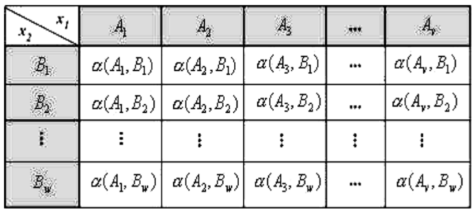
Fig. 7. Functional block diagram for complete system.

genetic algorithm (GA) unit. Functionally, the role of the GA is to evolve a suitable context which is then configured into the CMU of the RFIC. The RFIC-based system is capable of maintaining suitable level of performance in a dynamically changing operating environment.

In Fig. 7, data from 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 earlier, the address to access the FIM encoded by the AEM, 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 RFIC computes the control output. The whole setup allows for online context switching, a critical feature of intrinsic evolution for evolvable hardware systems.

IV. CONFIGURING FIM

The application developmental framework for an RFIC-based system requires a FIM configuration program to compute data set for configuring the FIM. Once programmed, the RFIC 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 section, we illustrate in details the whole process of configuring the FIM.



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

Fig. 8. Linguistic matrix of maximum rule set for two input variables.

A. 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 [34]. As a general illustration, Fig. 8 shows the linguistic matrix for a system of two input variables x_1 and x_2 with the linguistic sets $\{A_1, A_2, \ldots, A_v\}$ and $\{B_1, B_2, \ldots, B_w\}$, respectively. 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 $< x_1 = A_i >$ and $< x_2 = B_j >$ then $< y = \alpha(A_i, B_j) >$" where $\alpha(A_i, B_j)$ is a linguistic value of the variable y. If we consider the linguistic set of the output variable y to be $\{D_1, D_2, \ldots, D_u\}$, this will imply that $\alpha(A_i, B_j) \in \{D_1, D_2, \ldots, D_u\}$.

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, we can write κ as follows:

$$\kappa = \bigcup_{i=1\ldots v} \bigcup_{j=1\ldots w} \{\alpha(A_i, B_j)\} \tag{2}$$

The size of κ is $(v \times w)$.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. 9, the cells can be conveniently classified into groupings. For example, the Grouping $< 3, 5 >$ is representative of the following rules:

$$\begin{aligned}
&\text{If } < x_1 = A_3 > \text{ and } < x_2 = B_5 > \text{ then } < y = D_1 > \\
&\text{If } < x_1 = A_3 > \text{ and } < x_2 = B_5 > \text{ then } < y = D_2 > \\
&\text{If } < x_1 = A_3 > \text{ and } < x_2 = B_5 > \text{ then } < y = D_3 > \\
&\vdots \\
&\text{If } < x_1 = A_3 > \text{ and } < x_2 = B_5 > \text{ then } < y = D_u > .
\end{aligned}$$

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

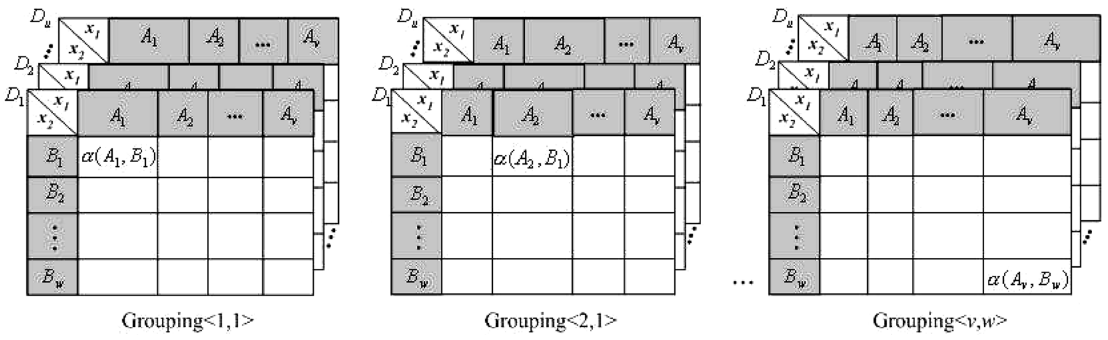


Fig. 9. Linguistic matrices of the $(v \times w)$ groupings.

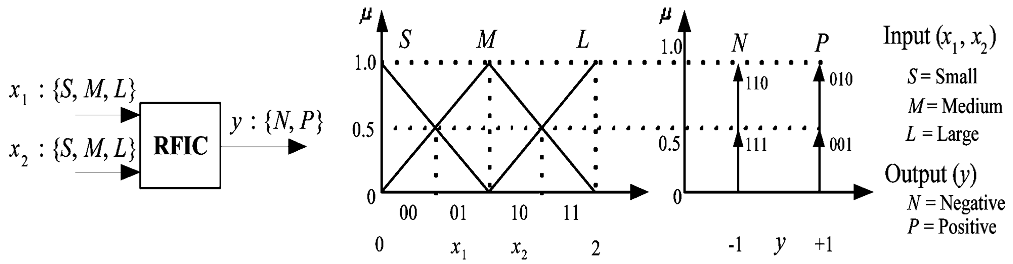


Fig. 10. Membership functions of a simplified system.

same *antecedent part*. The overall situation is depicted in Fig. 9, showing the various groupings.

In general, for each grouping$< i, j >$, only the cell corresponding to "If $< x_1 = A_i >$ and $< x_2 = B_j >$ then $< y = \alpha(A_i, B_j) >$" is included, while the other cells are treated as NULL entries. With the $(v \times w)$ groupings, we 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. 9 . 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 RFIC 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 pulse functions. The number of m bits is chosen to indicate the output variables. Hence, the width of the address bus to the FIM is $(k_1 + k_2 + m)$. In our RFIC, the width of data bus is p bits. The complete FIM configuration procedure is demonstrated by means of a simple illustration. Following this illustration, we present two practical examples in the next section. The functional simulation of the practical example is also incorporated to verify the operational functionality of the RFIC.

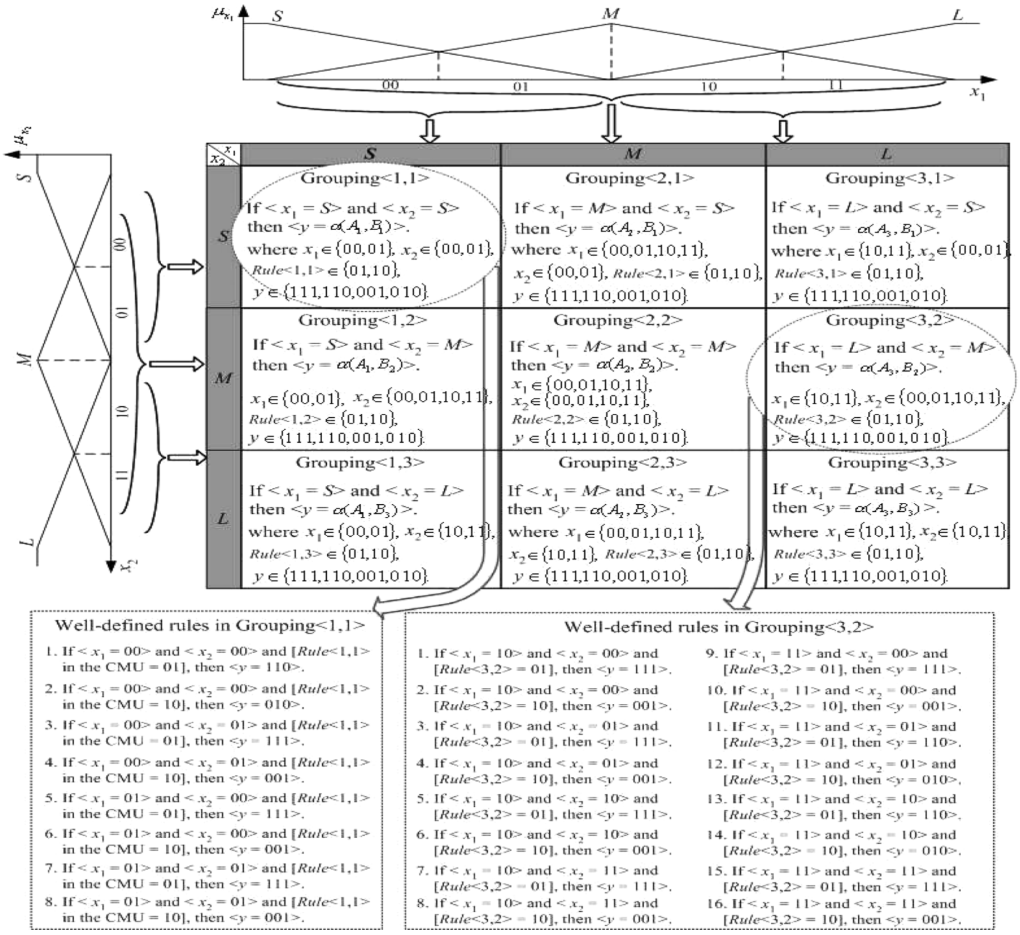
B. Simple Illustration

To describe the process of configuring the FIM in the RFIC, we use a simple example 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. 10

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

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

1) Coverage Map: The digitization of the membership functions for variables x_1, x_2 and y is as shown in Fig. 10. 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. 11, 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. 11. 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 $< x_1 = S >$ and $< x_2 = S >$ then $< y = \alpha(A_1, B_1) >$". 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



x_2 \ x_1	S	M	L
S	Grouping<1,1> If $<x_1 = S>$ and $<x_2 = S>$ then $<y = \alpha(A_1, B_1)>$. where $x_1 \in \{00,01\}$, $x_2 \in \{00,01\}$, $Rule<1,1> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<2,1> If $<x_1 = M>$ and $<x_2 = S>$ then $<y = \alpha(A_2, B_1)>$. where $x_1 \in \{00,01,10,11\}$, $x_2 \in \{00,01\}$, $Rule<2,1> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<3,1> If $<x_1 = L>$ and $<x_2 = S>$ then $<y = \alpha(A_3, B_1)>$. where $x_1 \in \{10,11\}$, $x_2 \in \{00,01\}$ $Rule<3,1> \in \{01,10\}$, $y \in \{111,110,001,010\}$
M	Grouping<1,2> If $<x_1 = S>$ and $<x_2 = M>$ then $<y = \alpha(A_1, B_2)>$. $x_1 \in \{00,01\}$, $x_2 \in \{00,01,10,11\}$, $Rule<1,2> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<2,2> If $<x_1 = M>$ and $<x_2 = M>$ then $<y = \alpha(A_2, B_2)>$. $x_1 \in \{00,01,10,11\}$, $x_2 \in \{00,01,10,11\}$, $Rule<2,2> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<3,2> If $<x_1 = L>$ and $<x_2 = M>$ then $<y = \alpha(A_3, B_2)>$. $x_1 \in \{10,11\}$, $x_2 \in \{00,01,10,11\}$, $Rule<3,2> \in \{01,10\}$, $y \in \{111,110,001,010\}$
L	Grouping<1,3> If $<x_1 = S>$ and $<x_2 = L>$ then $<y = \alpha(A_1, B_3)>$. where $x_1 \in \{00,01\}$, $x_2 \in \{10,11\}$, $Rule<1,3> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<2,3> If $<x_1 = M>$ and $<x_2 = L>$ then $<y = \alpha(A_2, B_3)>$. where $x_1 \in \{00,01,10,11\}$, $x_2 \in \{10,11\}$, $Rule<2,3> \in \{01,10\}$, $y \in \{111,110,001,010\}$	Grouping<3,3> If $<x_1 = L>$ and $<x_2 = L>$ then $<y = \alpha(A_3, B_3)>$. where $x_1 \in \{10,11\}$, $x_2 \in \{10,11\}$ $Rule<3,3> \in \{01,10\}$, $y \in \{111,110,001,010\}$

Well-defined rules in Grouping<1,1>

1. If $<x_1 = 00>$ and $<x_2 = 00>$ and [$Rule<1,1>$ in the CMU = 01], then $<y = 110>$.
2. If $<x_1 = 00>$ and $<x_2 = 00>$ and [$Rule<1,1>$ in the CMU = 10], then $<y = 010>$.
3. If $<x_1 = 00>$ and $<x_2 = 01>$ and [$Rule<1,1>$ in the CMU = 01], then $<y = 111>$.
4. If $<x_1 = 00>$ and $<x_2 = 01>$ and [$Rule<1,1>$ in the CMU = 10], then $<y = 001>$.
5. If $<x_1 = 01>$ and $<x_2 = 00>$ and [$Rule<1,1>$ in the CMU = 01], then $<y = 111>$.
6. If $<x_1 = 01>$ and $<x_2 = 00>$ and [$Rule<1,1>$ in the CMU = 10], then $<y = 001>$.
7. If $<x_1 = 01>$ and $<x_2 = 01>$ and [$Rule<1,1>$ in the CMU = 01], then $<y = 111>$.
8. If $<x_1 = 01>$ and $<x_2 = 01>$ and [$Rule<1,1>$ in the CMU = 10], then $<y = 001>$.

Well-defined rules in Grouping<3,2>

1. If $<x_1 = 10>$ and $<x_2 = 00>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
2. If $<x_1 = 10>$ and $<x_2 = 00>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.
3. If $<x_1 = 10>$ and $<x_2 = 01>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
4. If $<x_1 = 10>$ and $<x_2 = 01>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.
5. If $<x_1 = 10>$ and $<x_2 = 10>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
6. If $<x_1 = 10>$ and $<x_2 = 10>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.
7. If $<x_1 = 10>$ and $<x_2 = 11>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
8. If $<x_1 = 10>$ and $<x_2 = 11>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.
9. If $<x_1 = 11>$ and $<x_2 = 00>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
10. If $<x_1 = 11>$ and $<x_2 = 00>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.
11. If $<x_1 = 11>$ and $<x_2 = 01>$ and [$Rule<3,2>$ = 01], then $<y = 110>$.
12. If $<x_1 = 11>$ and $<x_2 = 01>$ and [$Rule<3,2>$ = 10], then $<y = 010>$.
13. If $<x_1 = 11>$ and $<x_2 = 10>$ and [$Rule<3,2>$ = 01], then $<y = 110>$.
14. If $<x_1 = 11>$ and $<x_2 = 10>$ and [$Rule<3,2>$ = 10], then $<y = 010>$.
15. If $<x_1 = 11>$ and $<x_2 = 11>$ and [$Rule<3,2>$ = 01], then $<y = 111>$.
16. If $<x_1 = 11>$ and $<x_2 = 11>$ and [$Rule<3,2>$ = 10], then $<y = 001>$.

Fig. 11. Coverage map of the nine groupings for configuring the FIM.

following series of well-defined rules:

$$\begin{aligned}
&\text{If } <x_1 = 00 \text{ and } <x_2 = 00> \text{ and}\\
&[\text{ Rule } <1,1>= 01], \text{ then } <y = 110>\\
&\text{If } <x_1 = 00 \text{ and } <x_2 = 00> \text{ and}\\
&[\text{ Rule } <1,1>= 10], \text{ then } <y = 010>\\
&\text{If } <x_1 = 00 \text{ and } <x_2 = 01> \text{ and}\\
&[\text{ Rule } <1,1>= 01], \text{ then } <y = 111>\\
&\text{If } <x_1 = 00 \text{ and } <x_2 = 01> \text{ and}\\
&[\text{ Rule } <1,1>= 10], \text{ then } <y = 001>\\
&\text{If } <x_1 = 01 \text{ and } <x_2 = 00> \text{ and}\\
&[\text{ Rule } <1,1>= 01], \text{ then } <y = 111>\\
&\text{If } <x_1 = 01 \text{ and } <x_2 = 00> \text{ and}\\
&[\text{ Rule } <1,1>= 10], \text{ then } <y = 001>\\
&\text{If } <x_1 = 01 \text{ and } <x_2 = 01> \text{ and}\\
&[\text{ Rule } <1,1>= 01], \text{ then } <y = 111>\\
&\text{If } <x_1 = 01 \text{ and } <x_2 = 01> \text{ and}\\
&[\text{ Rule } <1,1>= 10], \text{ then } <y = 001> .
\end{aligned}$$

The data set for the grouping$<1,1>$ is as shown in Table II-A. 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 $\mathrm{PB}_{<1,1>}$ of the FIM via the FIM configuration interface during the configuration stage. The coverage map for all the nine groupings is as shown in Fig. 11 with detailed illustration of the well-defined rules for Grouping$<1,1>$ and Grouping$<3,2>$ 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$<2,1>$, the data set of $\mathrm{PB}_{<2,1>}$ is configured as shown in Table II-B. Similarly for grouping $<3,1>$, the data set of $\mathrm{PB}_{<3,1>}$ is configured as shown in Table II-C. This is repeated for all the other groupings as shown in Table II-D–I.

2) FIM Configuration Data: All the data sets for the 9 partition blocks are as shown in Table II. These data sets are used to program the partition blocks of the RFIC through the FIM configuration interface during the configuration stage.

V. Application Examples

Two examples are considered to illustrate the practical functionality of the RFIC. The first application selected is that of a water bath temperature control, for demonstrating the practical functionality of the RFIC. A scheme for controlling the temperature of the water bath based on evolutionary learning of fuzzy system has been described in [35]. In our work, the model

TABLE II
DATA SETS FOR CONFIGURING FIM PARTITION BLOCKS

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.

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.

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.

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.

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 00 10:001

e.

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.

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.

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.

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.

presented in [36] is adopted to simulate the functionality of the plant controlled by an RFIC-based fuzzy system. The simulation results and a comparison between a typical fuzzy solution and the RFIC implementation of the control system are presented.

The second example is a real-time application of the RFIC in an evolvable fuzzy hardware system. The RFIC serves as the core-processor in an *evolvable fuzzy system* (EFS) for ATM cell scheduling. An EFS for ATM cell scheduling combining GA and fuzzy system was proposed in [37]. A GA is used to search for good fuzzy rule sets online. If the EFS is to be realized in hardware at the system-on-chip level, the scheme based on RFIC has the capability to support intrinsic evolution. There are certain inherent advantages of the proposed intrinsic evolvable and online adaptive EFS. One of the main advantages is that the system can adapt to the changes of the cell flow traffic to maintain good system performance. When the working fuzzy rule set is not suitable for the current cell flow scenario, it is replaced by another fuzzy rule set derived by an evolutionary search algorithm. Another main advantage is that the *quality of service* (QoS) performance satisfied by the system can be tuned by adjusting the appropriate parameters in the fitness function. We present the simulation results, and compare the results with those of first-in–first-out (FIFO), static priority (SPR), and dynamically weighted priority scheduling (DWPS) schemes.

A. Water Bath Temperature Control

The objective is to develop a fuzzy system for water bath temperature control. The membership functions for the input

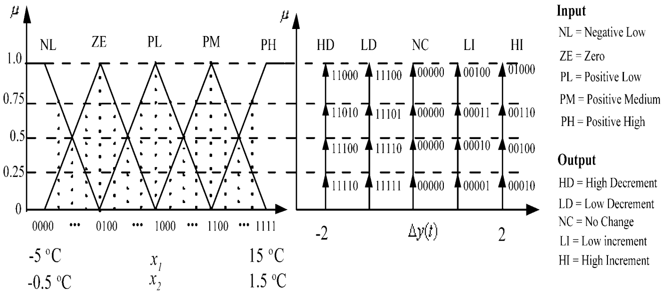


Fig. 12. Membership functions for the water bath temperature controller.

variables x_1 and x_2, output variable $\Delta y(t)$ are as shown in Fig. 12. Both input variables are described by the linguistic set $\{\mathrm{NL}, \mathrm{ZE}, \mathrm{PL}, \mathrm{PM}, \mathrm{PH}\}$ while the output variable is described by $\{\mathrm{HD}, \mathrm{LD}, \mathrm{NC}, \mathrm{LI}, \mathrm{HI}\}$. We choose 4 bits $(k_1, k_2 = 4)$ 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. 12. 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.

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 IV-B).

TABLE III
CONFIGURATION DATA SET FOR GROUPING $<1, 1>$ OF THE FIM

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
•••	•••	•••	•••	•••
•••	•••	•••	•••	•••

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

a.

x_2 \ 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.

Inputs:

NL=Negative Low
ZE = Zero
PL=Positive Low
PM=Positive Medium
PH=Positive High

x_2 \ 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.

x_2 \ 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.

Output:

Null Entry - 0
HD - 1
LD - 2
NC - 3
LI - 4
HI - 5

Fig. 13. Four examples of contexts for the RFIC.

For example, the 1-rule fuzzy sub-system with the active rule "If $< x_1 = \mathrm{NL} >$ and $<x_2 = \mathrm{NL} >$ then $<y = \alpha(A_1, B_1) >$" 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 III. The data set is programmed into the $\mathrm{PB}_{<1,1>}$ 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 RFIC can function efficiently at potentially high inference speed.

An example of a 12-rule fuzzy system as linguistic rule matrix is as shown in Fig. 13(a). The coding for the fuzzy rule set is "0405513055100500130500000". This represents the context of the domain problem loaded into the context register (See Fig. 3). 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 RFIC are activated during inferencing. Three other contexts (14-rule, 16-rule, and 17-rule) are also as shown in Fig. 13. From the viewpoint of the RFIC, any of the contexts specified in Fig. 13(b), –(d) can be effected by loading the corresponding codes "3033503000010503140502240", "0030303545015353023400350", and "40044235003225521 15001003", respectively into the context register. The RFIC functions accordingly, regardless of the context presented to the system via the context register.

1) Functional Simulation: We simulate the function of the RFIC using these four different contexts and compare our results with the fuzzy control system in Lim and Ng [35]. We show four cases of simulation based on the RFIC being used as the plant controller. The results are presented side by side with the control response of a typical fuzzy control implementation [35]. The plots of the control response for four different contexts at the same temperature settings (from 25° to 80°) are presented in Fig. 14(a)–(d). 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.

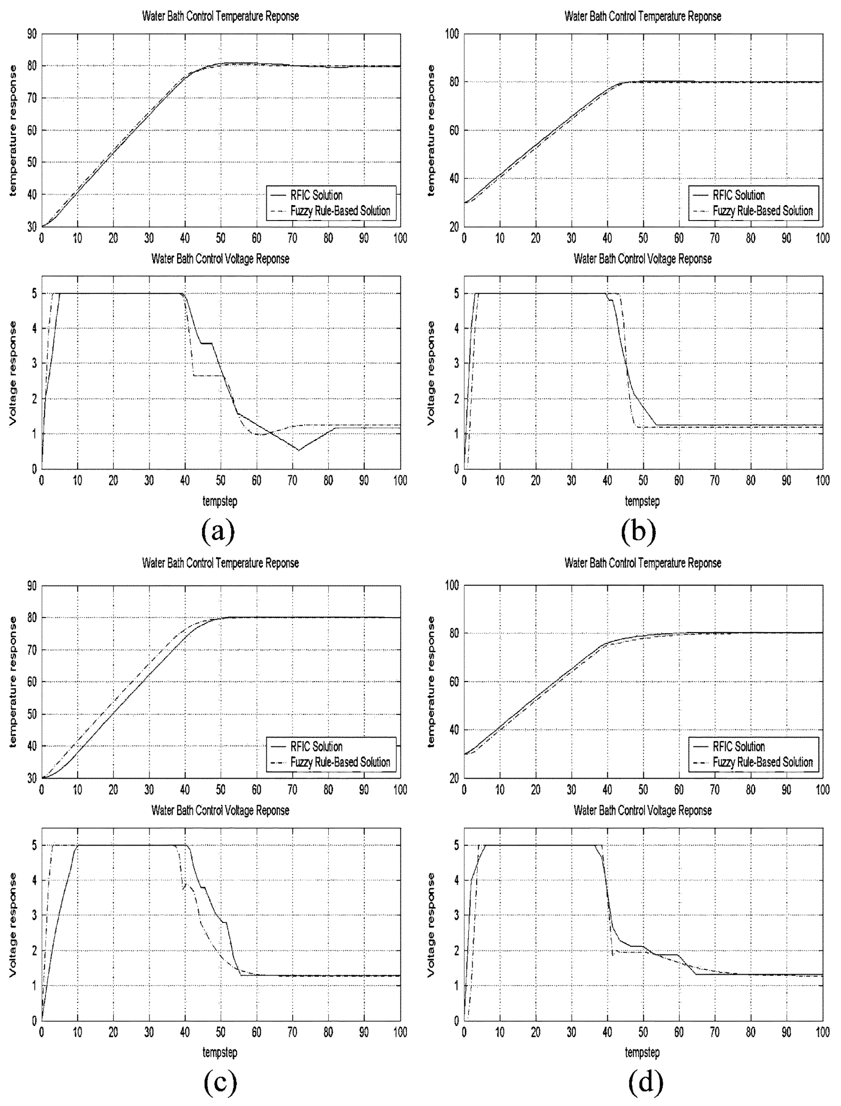


Fig. 14. Control responses for heating from 25° to 80° (a) Response for 12-rule context. (b) Response for 14-rule context. (c) Response for 16-rule context. (d) Response for 17-rule context.

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

B. ATM Cell Scheduling

In order to demonstrate the overall developmental framework of EFS using the RFIC, we consider the case of an evolvable fuzzy hardware system for ATM cell scheduling. It is envisaged that the pattern of cell flow traffic into an ATM cell scheduling system may change dynamically. To maintain a desired level of QoS performance and to satisfy the real-time requirement, ATM cell scheduling model requires online evolution in order for evolvable fuzzy hardware to be practical. This poses the requirement for evolvable fuzzy hardware that conveniently supports hardware reconfiguration. There are also many algorithms designed to solve the problem of cell scheduling. The common switching schemes are FIFO, SPR, and DWPS schemes [38]. The performance of these algorithms is described by the QoS. The major parameters of QoS are cell loss and cell delay. FIFO is easy to implement in hardware. However, it is not very good in terms of QoS performance. SPR is also an easily realizable approach. However, there is a tendency for priority bias whereby *class1* cell always has a higher priority than other cell classes. DWPS is a significant improvement over the SPR scheme. It adjusts the priority according to the cell flow scenarios [38]. However, its adaptation scheme is simple and may not be efficient if the cell flow changes dramatically. DWPS is also difficult to implement in hardware.

The architectural framework of the EFS for ATM cell scheduling is as shown in Fig. 15. 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

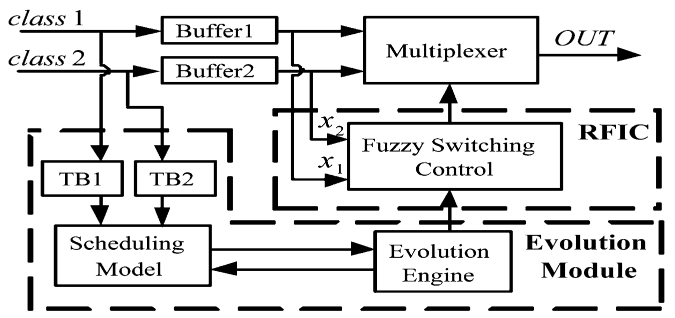


Fig. 15. Architectural framework of EFS for ATM cell scheduling.

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 RFIC, we simulate the ATM cell scheduling system using the RFIC to emulate the FSC function. The simulation results of the EFS using the RFIC as the core-processor are compared with the results of using the FIFO, SPR, and DWPS schemes.

Simulation of the EFS scheduling for ATM cell scheduling has been reported in [37], with detailed QoS performance analysis and evaluation presented in [39]. For each scenario, an optimal fuzzy rule set is derived by a GA. The fitness function used in the evolution module is described by (3).

$$F = \tau - |\,\text{AveDelay} - \lambda \times\ \text{MaxDelay}|. \tag{3}$$

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 our simulation, we choose the value of λ to be 0.38.

1) Simulation Results and Comparisons: In order to test the system's performance, two traffic scenarios lasting for 2 seconds are adopted for simulation [37]. Scenario1 is a statistical model to describe the aspect of burstiness of cell flow traffic. It satisfies the Poisson distribution with a mean value of 2 ms. The *class1* and *class2* cell traffic are cell flows of 155.52 MHz, shown in Fig. 16(a). The difference between them is that *class1* is CBR (constant bit rate) while *class2* is a bursty traffic flow with 2 ms ON period and 2 ms OFF period.

The second traffic scenario which is referred to as Scenario 2 is more complicated than the first. The cell flow is as shown in Fig. 16(b). For *class1*, the bit rate is 100 MHz while the bit rate of *class2* can change randomly every 2 ms. The bit rate varies from 55.52 to 155.52 MHz and the total bit rate of the input channels is larger than the output channel capacity of 155.52 MHz [37]. With these two traffic scenarios, we test the functionality and robustness of the RFIC-based EFS.

Scenario 1: The simulation results are as shown in Fig. 17. In each figure, we compare the QoS performance of the RFIC-based EFS with the FIFO, SPR, and DWPS scheduling schemes.

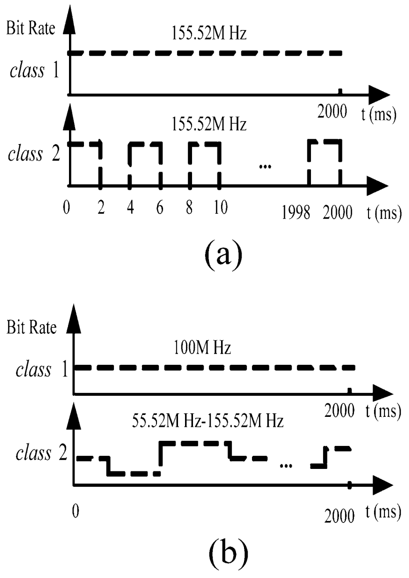


Fig. 16. Two scenarios of cell flow. (a) Scenario 1. (b) Scenario 2.

The QoS is described in terms of cell loss and cell delay in ATM network. In general, *class1* communication should be assigned a wide bandwidth, while *class2* communication is assigned a narrow bandwidth. In Fig. 17(a), the SPR scheduling scheme shows a minimum *class1* cell loss, while cell loss for *class2* communication as shown in Fig. 17(b) is maximal. For the cell delay shown in Fig. 17(c) and (d), it appears that the SPR scheme which produces zero delay for both *class1* and *class2* has the best performance. This however is misleading. Since the SPR scheme accords priority to *class1* cell, and the capacity of the *OUT* channel is just enough to accommodate *class1* communication, all the *class2* cells are actually not serviced and therefore lost. The plot of Fig. 17(d) shows a zero cell delay for *class2* because the calculation of cell delay only account for cells being serviced. In general, the SPR scheme exhibits a form of extreme bias that results in too good a service for *class1*. The cell loss and cell delay of *class1* are kept to zero, while the cell loss and cell delay of *class2* are sacrificed. Even when the 2-s simulation is completed, *Buffer2* continues to be filled with *class2* cells waiting to be serviced. For the other three scheduling schemes, the cell loss is spread out over *class1* and *class2* cell flow. The total cell losses for the other three schemes are about the same. From Fig. 17(a)–(c), the DWPS scheme is more capable in scheduling the *class1* and *class2* cell flow than the FIFO scheme. It achieves a better balance of cell loss between *class1* and *class2* communications, as well as a smaller *class1* cell delay. Compared to the DWPS scheme, the RFIC-based EFS scheme achieves smaller *class1* cell loss and *class1* cell delay in Fig. 17(a) and (b). It is evident from the plots in Fig. 17 that the RFIC-based EFS scheme performs just as well as the DWPS scheme. The simulation results show that the RFIC-based EFS scheme works efficiently, and it achieves good QoS performance based on Scenario 1 cell flow traffic.

Scenario 2: In this scenario, the pattern of cell flow changes dynamically. From a practical point of view, Scenario 2 is more

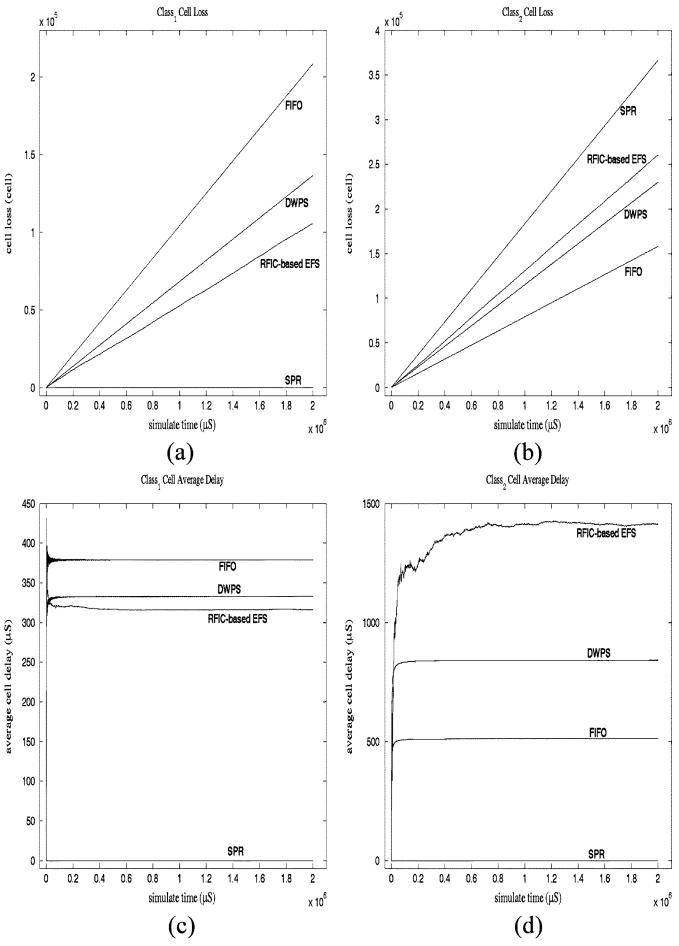


Fig. 17. QoS performance for *class1* and *class2* in Scenario 1. (a) Cell loss of *class1*. (b) Cell loss of *class2*. (c) Cell delay of *class1*. (d) Cell delay of *class2*.

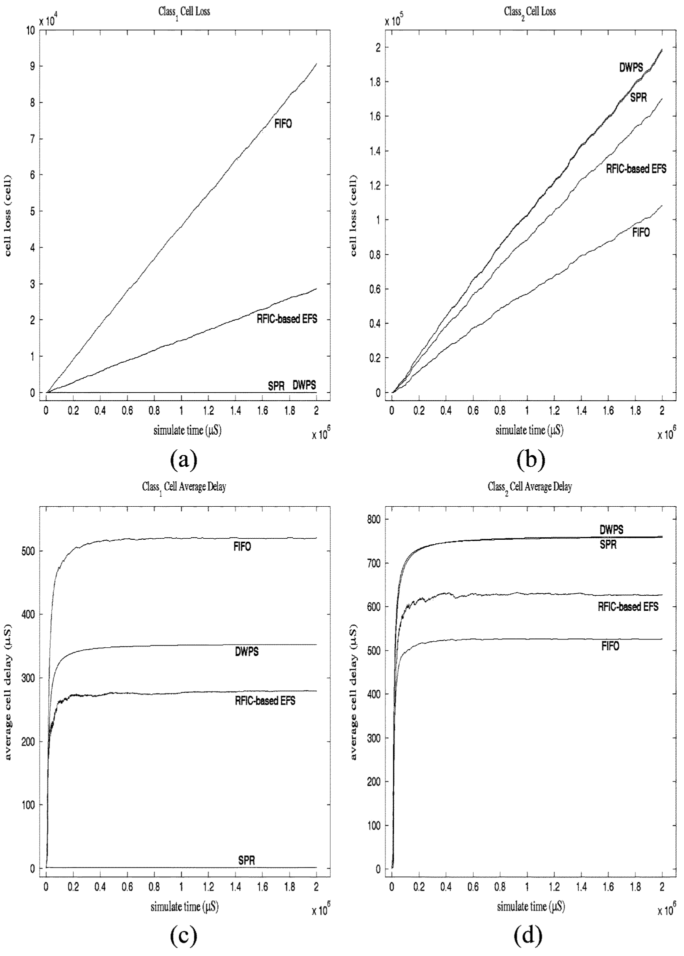


Fig. 18. QoS performance for *class1* and *class2* in Scenario 2. (a) Cell loss of *class1*. (b) Cell loss of *class2*. (c) Cell delay of *class1* Cell delay of *class2*.

likely to occur than Scenario 1. It is necessary to adopt an adaptive scheduling scheme to maintain a desirable level of QoS performance. The simulation results using the four schemes are as shown in Fig. 18.

For the SPR scheme, *class1* cell loss and cell delay are kept at zero again [see Fig. 18(a) and (c)]. However, according to Fig. 18(b) and (d), the SPR scheme suffers the largest *class2* cell loss and cell delay due to the low priority assigned for *class2* communication. Comparing the SPR and DWPS schemes, all the plots show the same performance level except for Fig. 18(c) which clearly shows the SPR scheme delivering a better *class1* cell delay. In retrospect, the observation for Scenario 1 (Fig. 17) showed a comparable performance level between the SPR and DWPS schemes. This suggests that the performance of the DWPS scheme has degraded from Scenarios 1 to 2. In this respect, the DWPS scheme is not efficient when cell flow changes unpredictably. The sum of the cell losses of the two classes for the four schemes is about the same. The simulation results show that the FIFO scheme performs better than the SPR and DWPS schemes in terms of maintaining balance between *class1* cell loss and *class2* cell loss. By the same token, as shown in Fig. 18(a) and (b), the RFIC-based EFS scheme is able to achieve a similar balance compared to the FIFO scheme. On top of that, the RFIC-based EFS scheme shows much smaller *class1* cell delay than the FIFO scheme in Fig. 18(c). Considering cell delay and the ability to maintain desirable balance between the *class1* cell loss and *class2* cell loss, the RFIC-based EFS scheme is generally more advantageous over the FIFO, SPR and DWPS schemes. It is a more versatile approach compared to all the other schemes.

The simulation results have shown that our proposed EFS using the RFIC as the core-processor works efficiently. It is capable of dealing with the ATM cell scheduling, a potential real-time application of evolvable fuzzy hardware. It is a viable alternative to handle the unpredictable scenarios in an ATM network. For the RFIC-based EFS, the QoS performance can be tuned easily by setting different values of λ for the fitness function. This is a significant advantage compared to the other three schemes.

VI. Conclusion

The real-time requirement of some applications necessitates the implementation of the fuzzy inference as embedded hardware. In this respect, the main challenge is the implementation of hardware that supports in-system reconfiguration. In this paper, we overcome this challenge by means of a RFIC which supports online context switching. A context refers to an instance of fuzzy control rule set and membership functions. Since a switch in context as a result of changes in membership functions has limited applicability, we design our chip to handle context updating brought about by changes in the domain rules. More importantly, the RFIC can offer very high inference throughput to support time-critical applications.

The RFIC consists of FIM partition blocks which implicitly encode the fuzzy relational knowledge and the inference model. The key to the context independence of the RFIC is the novel structuring of the FIM partition blocks. Any changes in the domain rule set will not affect the FIM configuration. It is only necessary to update the context in the RFIC via a loadable register. In this respect, the bottleneck of having to reconfigure the whole FIM is avoided.

The context switchable RFIC is an ideal platform to develop evolvable fuzzy hardware. The RFIC incorporates the desirable characteristic of real-time reconfigurability, adaptation and high inference speed, making it a viable alternative for realizing intrinsically evolvable fuzzy hardware systems. We can design an intrinsic evolvable fuzzy hardware system using the RFIC as the core fuzzy processing unit. A GA can be deployed to derive the appropriate context for the applications. The most appropriate context is applied to the RFIC online, via the CMU. In this way, intrinsic evolution can be realized smoothly and expeditiously without interrupting the normal operation of the fuzzy system. The data set to configure the FIM for handling all the contexts in the GA search space is achieved through supporting software utility programs.

Two examples to illustrate the functionality of the RFIC were presented. The first example pertains to a water temperature control. It clearly illustrates the RFIC's versatility in handling context-independent fuzzy inferencing. The second example of ATM cell scheduling is a real-time switching control application. In this example, the viability of the RFIC as a device to support evolvable fuzzy hardware is demonstrated. More importantly, it demonstrates that the RFIC is capable of supporting intrinsic hardware evolution.

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," *Inform. Control*, vol. 8, no. 3, pp. 338–353, June 1965.

[2] ——, "Fuzzy logic," *IEEE Comput.*, vol. 21, no. 4, pp. 83–93, Apr. 1988.

[3] M. H. Lim, J. Y. Leong, and K. T. Lau, "A general framework for the development of PLD-based fuzzy controllers," *J. Inst. Eng.*, vol. 35, no. 1, pp. 7–12, Feb. 1995.

[4] N. E. Evmorforpoulos and J. N. Avaritsitis, "Adaptive digital fuzzy hardware in application-specific integrated circuits," in *Proc. 6th IEEE Int. Conf. Electronics, Circuits, and Systems*, Sept. 1999, pp. 1635–1638.

[5] 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 Trans. Fuzzy Syst.*, vol. 10, no. 2, pp. 234–246, Apr. 2002.

[6] 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 Trans. Fuzzy Syst.*, vol. 12, no. 4, pp. 436–446, Aug. 2004.

[7] 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.

[8] S. Mollov, R. Babuska, J. Abonyi, and H. B. Verbruggen, "Effective optimization for fuzzy model predictive control," *IEEE Trans. Fuzzy Syst.*, vol. 12, no. 5, pp. 661–675, Oct. 2004.

[9] G. Louverdis and I. Andreadis, "Design and implementation of a fuzzy hardware structure for morphological color image processing," *IEEE Trans. Circuits Syst. Video Technol.*, vol. 13, no. 3, pp. 277–288, Mar. 2003.

[10] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller - Part I," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, no. 2, pp. 404–418, Mar. 1990.

[11] A. Costa, A. D. Gloria, P. Faraboschi, A. Pagni, and G. Rizzotto, "Hardware solutions for fuzzy control," in *Proc. IEEE*, Mar. 1995, vol. 83, no. 3, pp. 422–434.

[12] G. Ascia, V. Catania, and M. Russo, "VLSI hardware architecture for complex fuzzy systems," *IEEE Trans. Fuzzy Syst.*, vol. 7, no. 5, pp. 553–570, Oct. 1999.

[13] 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.

[14] P. Thrift, "Fuzzy logic synthesis with genetic algorithms," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 509–513.

[15] M. H. Lim, S. Rahardja, and B. H. Gwee, "A GA paradigm for learning fuzzy rules," *Fuzzy Sets Syst.*, vol. 82, no. 2, pp. 177–186, Sept. 1996.

[16] H. Eichfeld, T. Künemund, and M. Menke, "A 12 b general-purpose fuzzy logic controller chip," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 4, pp. 460–475, Nov. 1996.

[17] K. Shimizu, M. Osumi, and F. Imae, "The digital fuzzy processor FP-5000," in *Proc. 2nd Int. Conf. Fuzzy Logic and Neural Networks*, Dec. 1992, pp. 539–542.

[18] H. Watanabe, W. D. Dettloff, and K. E. Yount, "A VLSI fuzzy logic controller with reconfigurable, cascadable architecture," *IEEE J. Solid-State Circuits*, vol. 25, no. 2, pp. 376–382, Apr. 1990.

[19] A. Gabrielli and E. Gandolfi, "A fast digital fuzzy processor," *IEEE Micro.*, vol. 19, no. 1, pp. 68–79, Jan. 1999.

[20] M. Sasaki, F. Ueno, and T. Inoue, "7.5 MFLIPS fuzzy microprocessor using SIMD and logic-in-memory structure," in *Proc. 2nd IEEE Int. Conf. Fuzzy Systems*, Sep. 1993, pp. 527–534.

[21] T. Yamakawa and T. Miki, "The current mode fuzzy logic integrated circuits fabricated by the standard CMOS process," *IEEE Trans. Comput.*, vol. C-35, no. 2, pp. 161–167, Feb. 1986.

[22] M. A. Manzoul and D. Jayabharathi, "Fuzzy controller on FPGA chip," in *Proc. IEEE Int. Conf. Fuzzy Systems (FUZZ-IEEE '92)*, Mar. 1992, pp. 1309–1316.

[23] K. Nakamura, N. Sakashita, Y. Nitta, K. Shimomura, and T. Tokuda, "A 12-bit resolution 200 k FLIPS fuzzy inference processor," *IEICE Trans. Electron.*, vol. E76-C, no. 7, pp. 1102–1110, Jul. 1993.

[24] T. Yamakawa, "A fuzzy inference engine in nonlinear analog mode and its application to a fuzzy logic control," *IEEE Trans. Neural Netw.*, vol. 4, no. 3, pp. 496–522, May 1993.

[25] J. M. Jou, P. Y. Chen, and S. F. Yang, "An adaptive fuzzy logic controller: Its VLSI architecture and applications," *IEEE Trans. Very Large Scale Integration (VLSI) Syst.*, vol. 8, no. 1, pp. 52–60, Feb. 2000.

[26] D. L. Hung, "Dedicated digital fuzzy hardware," *IEEE Micro.*, vol. 15, no. 4, pp. 31–39, Aug. 1995.

[27] M. McKenna and B. M. Wilamowshi, "Implementing a fuzzy system on a field programmable gate array," in *Proc. Int. Joint Conf. Neural Networks*, Jul. 2001, pp. 189–194.

[28] V. Catania, A. Puliafito, M. Russo, and L. Vita, "A VLSI fuzzy inference processor based on a discrete analog approach," *IEEE Trans. Fuzzy Syst.*, vol. 2, no. 2, pp. 93–106, May 1994.

[29] S. Guo and L. Peters, "A high-speed fuzzy co-processor implemented in analogue/digital technique," *Comput. Elect. Eng.*, vol. 24, no. 1–2, pp. 89–98, Jan. 1998.

[30] M. A. Manzoul and D. Jayabharathi, "FPGA for fuzzy controllers," *IEEE Trans. Syst., Man, Cybern.*, vol. 25, no. 1, pp. 213–216, Jan. 1995.

[31] D. Kim, "An implementation of fuzzy logic controller on the reconfigurable FPGA system," *IEEE Trans. Ind. Electron.*, vol. 47, no. 3, pp. 703–715, Jun. 2000.

[32] M. J. Patyra, J. L. Grantner, and K. Koster, "Digital fuzzy logic controller: Design and implementation," *IEEE Trans. Fuzzy Syst.*, vol. 4, no. 4, pp. 439–459, Nov. 1996.

[33] Q. Cao, M. H. Lim, and J. H. Li, "A novel aggregation circuit for reconfigurable fuzzy inference chip," in *Proc. 2nd Int. Conf. Artificial Intelligence in Engineering and Technology*, Aug. 2004, pp. 835–839.

[34] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.

[35] M. H. Lim and W. L. Ng, "Iterative genetic algorithm for learning efficient fuzzy rule set," *Art. Intell. Eng. Design, Anal., Manuf.*, vol. 17, no. 4, pp. 335–347, Sep. 2003.

[36] J. Tanomaru and S. Omatu, "Process control by online trained neural controllers," *IEEE Trans. Ind. Electron.*, vol. 39, no. 6, pp. 511–521, Dec. 1992.

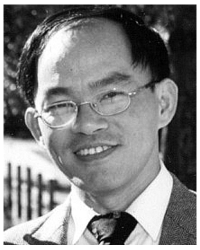
[37] J. H. Li and M. H. Lim, "Evolvable fuzzy system for ATM cell scheduling," in *Proc. 5th Int. Conf. Evolvable Systems, LNCS 2606*, 2003, pp. 208–217.

[38] T. Lizambri, F. Duran, and S. Wakid, "Priority scheduling and buffer management for ATM traffic shaping," in *Proc. 7th IEEE Workshop on Future Trends of Distributed Computing Systems*, 1999, pp. 36–43.

[39] J. H. Li, M. H. Lim, and Q. Cao, "A QoS-tunable scheme for ATM cell scheduling using evolutionary fuzzy system," *Appl. Intell.*, vol. 23, no. 3, pp. 207–218, Dec. 2005.

Qi Cao received the B.Eng. degree in control science and engineering from Huazhong University of Science and Technology, Wuhan, P.R. China, in 2000. He is currently working toward the Ph.D. degree at the School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

From 2000 to 2002, he was a Hardware Design Engineer with ALi, Shenzhen Branch, P.R.C. His research interests focus on GPGA design and in-system reconfigurable fuzzy inference processor design.



Meng Hiot Lim received the B.Sc., M.Sc., and Ph.D. degrees from the University of South Carolina, Columbia, in 1982, 1984, and 1988, respectively.

He joined the Faculty of the School of Electrical and Electronics Engineering, Nanyang Technological University, Singapore, in 1989, and is currently an Associate Professor. His research interests include computational intelligence, combinatorial optimization, web-based applications, evolvable hardware systems, reconfigurable circuits and architecture, computational finance, and graph theory.

He is an Associate Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART C, and has co-edited a volume on *Recent Advances in Simulated Evolution and Learning* (Singapore: World Scientific).

Ju Hui Li received the B.Eng. degree in computer communication from Hunan University, Changsha, P.R. China, in 1994, and the M.Eng. degree in computer application from Northwestern Polytechnical University, Xi'an, P.R. China, in 2001.

He is currently with Semiconductor-APIC, Philips Electronics Singapore Pte, Ltd. His research interests are evolvable hardware, evolvable fuzzy hardware, digital IC design, and mix-signal IC design.

Yew Soon Ong (M'99) received the B.S. and M.S. degrees in electrical and electronics engineering from Nanyang Technological University, Singapore, in 1998 and 1999, respectively, and subsequently the Ph.D. degree from the University of Southampton, Southampton, U.K., in 2003.

He is currently an Assistant Professor with the School of Computer Engineering, Nanyang Technological University. His research interests lie in computational intelligence, spanning: evolutionary optimization, surrogate-assisted evolutionary algorithms, memetic algorithms, evolutionary fuzzy systems, and grid-based computing.

He has been a Guest Editor of the IEEE TRANSACTIONS ON SYSTEMS, MAN, AND CYBERNETICS, PART B, and has co-edited a volume on *Advances in Natural Computation* (New York: Springer-Verlag).

Wil Lie Ng received the Honours degree in electrical and electronics engineering from the University of Edinburgh, Edinburgh, U.K., in 2001, and the M.S. degree by research in the School of EEE, Nanyang Technological University (NTU), Singapore, in 2004.

Currently, he is a member of the Centre for Advanced Information Systems at NTU. His research interest is in computational intelligence involving neural network, evolutionary programming, and fuzzy systems.