9

Evolvable Fuzzy Hardware for Real-time Embedded Control in Packet Switching

Ju Hui Li, Meng Hiot Lim, and Qi Cao

School of EEE, Block S1, Nanyang Technological University, Singapore 639798,
(pg01896341 | emhlim | pg04780942)@ntu.edu.sg

In this chapter, we describe a scheme to realize an *Evolvable Fuzzy Hardware* (EFH) for real-time Packet Switching problem. The common challenges of *Evolvable hardware* (EHW) implementation are issues pertaining to *online adaptation*, *scalability* and *termination of evolution* [1]. The proposed EFH addresses these issues effectively. A very interesting advantage of the proposed EFH is that the system performance can be tuned intuitively through parametric adjustment of the fitness function. This advantage gives the EFH system a very special property that conventional scheduling methods cannot fulfill easily. For the hardware implementation of the EFH, real-time fuzzy inference with high-speed context switching capability is necessary. We address this aspect through implementation based on a context independent *reconfigurable fuzzy inference chip* (RFIC).

9.1 Introduction to EHW and EFH

Evolvable hardware (EHW) is a new type of hardware whose architecture can be evolved to suit the operating environment. In recent years, it has been attracting greater attention from researchers. The idea behind EHW is based on evolutionary algorithm, a methodology to search the solution space to derive the appropriate hardware architecture. EHW can be classified into *extrinsic* and *intrinsic* EHW based on the scheme of evolution used. Extrinsic EHW relies on a simulated evolutionary process independent of the hardware. It may rely on hardware description languages (HDL), C or other programming languages to represent the circuit and then rely on an evolutionary algorithm to evolve the hardware configuration. Only the elite design is downloaded into the reconfigurable device. Intrinsic evolvability means that the evolution and evaluation of solutions are carried out at the hardware level of the EHW system. In principle, intrinsic EHW can modify its own hardware configuration and behavior autonomously. If the environment changes, the behavior or architecture will also change to maintain an acceptable level of system performance. Currently, there has been great progress made for extrinsic type of EHW [2, 3, 4, 5, 6, 7].

There are also research works that focused on intrinsic EHW. In some reported works, the researchers rely on a semi-intrinsic approach. They use software to realize the evolution part and hardware to carry out evaluation of the derived architecture. After the evolution process, the best chromosome is implemented in hardware. This scheme can be called *offline adaptive intrinsic* EHW. Most of the works on intrinsic EHW up to now can be found in [8, 9, 10]. This type of EHW generally has some advantages over extrinsic EHW. Since it carries out the evaluation in hardware, the evaluation process is very fast, and the performance of the elite is not affected by error in the simulation model. Intrinsic EHW is useful for applications that require online and real-time system reconfiguration. However, the implementation of intrinsic EHW still poses significant challenges for such promising areas.

From the perspective of evolution granularity, current EHW can be classified into three types: transistor level, gate level and function level. Among the three, the transistor level represents the lowest level of evolution granularity. This gives the greatest flexibility because transistors are the smallest components of any circuit. Gate level EHW means that logic gates are the smallest configurable components of the EHW [11, 12, 13, 14, 15, 16, 17]. Functional level EHW carries out the evolution of macro units (adder, multiplier, sine, cosine, etc.) implemented on a special type of FPGA [2, 18, 19]. There are many *functional processing units* (FPU) in the FPGA chip. Each FPU can be configured to perform one of the high-level functions such as addition, subtraction, multiplication, division, sine and cosine. The functions and connections of FPUs are configured based on the elite chromosome. Most of the EHW reported can be categorized into one of these three levels. The limitations of these forms of EHW imply that evolutions can only be done extrinsically or in some instances, intrinsically but in an offline adaptive manner.

For the implementation of intrinsic Evolvable and online adaptive EHW, there are three main open issues that need to be addressed [1]. These issues are briefly outlined below.

Online adaptation: This means that the system hardware is required to adapt during the normal operation. Online adaptation is very hard to realize because the system has to reconfigure the hardware for every chromosome in order to carry out the evaluation. Some chromosomes may inevitably result in very poor performance. If these chromosomes are evaluated by reconfiguring the hardware, they may potentially result in some damages or disastrous outcome.

Scalability refers to the extensibility of the scheme to handle more complex architecture or configurations. For a typical EHW, the chromosome length may be hundreds or even thousands of genes for a complicated system. The search space represented by a chromosome may be very big. Hence the search by the *genetic algorithms* (GA) for a good solution in such a big solution space may take a very long time.

Termination of evolution pertains to criteria or conditions for stopping the evolution process. For example, one commonly used criterion is the number of runs. With a GA scheme, there is no guarantee as to the number of runs required before a desirable solution can be found. This can be a significant drawback for real-time operation.

In order to perform online adaptive and intrinsic Evolvable hardware, we propose a new form of EHW that is referred to as *Evolvable Fuzzy Hardware* (EFH). EFH can be viewed as a form of *Evolvable fuzzy system* (EFS) whereby the fuzzy inference system is implemented in hardware to deliver real-time inference throughput. Furthermore, the domain knowledge of the fuzzy system should be able to support online real-time reconfiguration. EFH can overcome the disadvantages of the other three EHWs described earlier and is amenable to intrinsic evolution and online adaptation. Earlier in [20], we proposed EFS for ATM cell scheduling. In that system, the EFS searches for an appropriate fuzzy rule set to carry out the scheduling task on dynamically changing cell flows. The evolutionary search process does not cause any interruption in the system operation. After a good fuzzy rule set is found, the old one is replaced immediately. From simulation results, it was shown that EFS is capable of dynamic real-time adaptation to deliver robust performance. To further support our work, we have also proposed a *reconfigurable fuzzy inference chip* (RFIC) whereby the context can be changed or reconfigured online [21]. By combining the advantages of the EFS and RFIC, we demonstrate in this work how intrinsic Evolvable and online adaptive EFH can be implemented.

In Section 2, we introduce the real-time Packet Switching problem, an application for demonstrating the viability of the EFH. In Section 3, we describe specifically how the implementation challenges of the intrinsic EFH are addressed. In Section 4, we describe the detailed formulation of the fitness function adopted in our EFH. In Section 5, we present the simulation results of applying EFH to solve the real-time problem. Certain desirable properties of the EFH in dealing with the real-time problem are also discussed in this section. In Section 6, we outline details on how the EFH can be implemented from a system's perspective. Finally, we offer some concluding remarks for our work on EFH.

9.2 Packet Switching

Packet Switching is a backbone of modern communication networks. Because of the characteristics of the various services supported by the network, the management of the bandwidth resources is very critical. The multiplexer is an important component used to administer the sharing of bandwidth among different cell flows. It is mainly employed to provide a means of sharing high-speed link for network terminations or network inter-nodes. *Time division scheme* is adopted in the multiplexer. The output link can be divided into different time slots. At anytime, only one input flow is accorded the priority of sending packets through the output channel. The simplified block architecture of the multiplexer is as shown in Fig. 9.1. For illustration, we classify the services into two types, $class_1$ and $class_2$. In the block diagram, BUF_1 and BUF_2 refer to buffers for $class_1$ and $class_2$ respectively. MP represents the time division multiplexing system for transmitting packets through the OUT channel. The *switching control* block is a part of the hardware that handles cell scheduling. When the OUT channel is available, the *switching control* block decides on which cell flow to be sent.

For Packet Switching, $class_1$ can be a form of CBR (*Constant Bit Rate*) traffic, rt-VBR (*real-time Variable Bit Rate*) or both. The $class_2$ traffic type may refer to nrt-VBR (*non-real-time Variable Bit Rate*), UBR (*Unspecified Bit Rate*) or ABR (*Available Bit Rate*) [22]. While $class_1$ type is delay sensitive, $class_2$ is considered to be not sensitive to delay. These two sources of cell flow must be multiplexed on the output channel (OUT) by the MP unit through time division. The capacities of OUT and the input channels are fixed. In this problem, the QoS (*Quality of Service*) of the system can be evaluated by $class_1$ cell delay, $class_2$ cell loss and the balance between $class_1$ cell loss and $class_2$ cell loss. The ideal case is that $class_1$ cell delay and $class_2$ cell loss are very small and there is also a good balance established between $class_1$ cell loss and $class_2$ cell loss.

The application of EHW in ATM cell scheduling has been reported in Liu *et. al* [2, 3]. In their works, the authors presented schemes of functional EHW to solve the problem of cell scheduling. The functional EHW system successfully achieved a circuit that had service performance similar to that of traditional scheduling schemes. However, the scheme has some significant limitations, hence not suitable for practical applications. The main limitation of the system is its inability to evolve intrinsically. Another limitation is that the system had to rely on an external computation platform to carry out evolutionary process due to its large search space. Finally, the system faces the limitation of being trained and tested only on fixed cell flow patterns. In a practical system, the cell flows can change dramatically. There was no effective scheme in this system to adjust the system along with the changing cell flows.

9.3 Solutions for Open Issues

In order to solve the packet scheduling problem, we design the system architecture, incorporating evolutionary mechanisms as in Fig. 9.2. In this system, the training buffers TB_1 and TB_2 are used to store $class_1$ and $class_2$ cells respectively. The size of TB_1 and TB_2 is at least 2 or 3 times that of BUF_1

and BUF_2. When either TB_1 or TB_2 is full, the evolutionary process is triggered. Fitness evaluation is carried out by subjecting each chromosome to the *scheduling model* according to the cell flow stored in TB_1 and TB_2. The purpose of the *scheduling model* is to emulate the function of the multiplexer as in Fig. 9.1. After a specified number of cycles and generations, if a chromosome

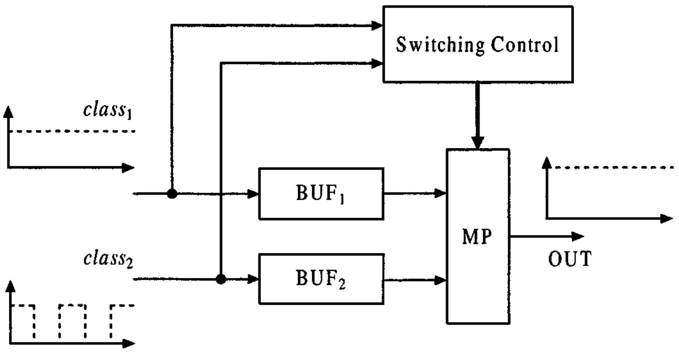

Fig. 9.1. Multiplexer scheme

that corresponds to a system rule set is better than the working chromosome, the working chromosome is replaced immediately. In order to prevent the search procedure from being trapped in a local region, after a pre-specified number of generations, the whole evolutionary process is restarted, from the point where the initial population is generated. This is essentially the start of a new *evolution cycle*. Functionally, the *scheduling model* emulates the packet switching to derive the cell delay and cell loss parameters. This is achieved by a multiplexer model within the *scheduling model* block. The derived parameters enable the fitness value to be calculated using the fitness function. Basically, the *evolution module* evolves the appropriate rule set by interacting with the *scheduling model* to evaluate the fitness of each evolved fuzzy rule set. When evolution is triggered, it works in the background while the MP unit is in operation. With EFH, the fuzzy inference circuit is a very important component and it directly affects the speed of the system's response to the changes in cell flow. Two high-speed fuzzy inference components are required. One is in the *scheduling model* and another is the RFIC block performing cell scheduling control.

During evolution, it is inevitable that poor quality chromosomes i.e., chromosomes that result in poor switching performance, are also evaluated. To avoid the possibility of detrimental effects on the system performance by these chromosomes, the *scheduling model* is incorporated in Fig. 9.2 to emulate the cell scheduling process. This allows for evaluation of the evolved chromosomes in the background. After the evolution process, only the final fuzzy rule set

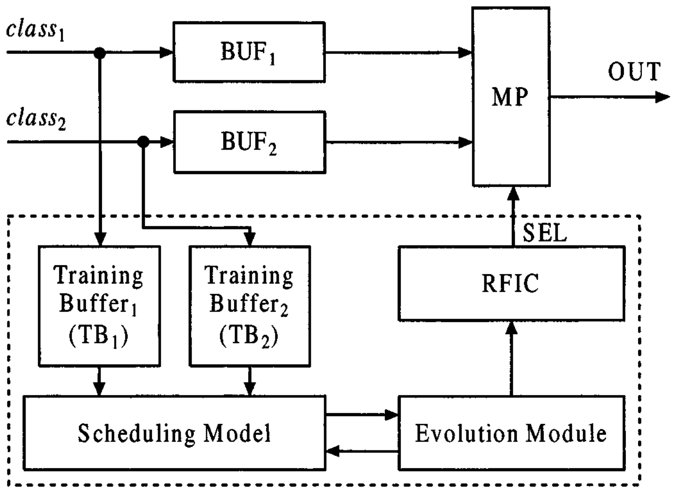

Fig. 9.2. Adaptation framework for EFH

will be configured in the RFIC block. In this way, we address the first major open issue of the intrinsic EHW.

In order to achieve online adaptation and intrinsic evolution for real-time control, another issue that can be regarded as a sub-problem of online adaptation and intrinsic evolution, must also be addressed. During evolution, training data are required. In [2], the EHW system uses the same data for training and testing. This scheme can work well in applications when the real time data do not change dramatically. But if the application scenario is significantly different from the training situation, the system may not perform very well. This indicates that extensive data samples are necessary for such an evolution scheme. If the real-time data change dramatically, it is not practical to incorporate diversely representative real data samples to train the system. For many real-time control areas, we believe that there is no need to do so. In fact, we can apply the principle of "locality" to substantiate this belief. For example, in computer operating system, the design of the cache memory system is based on this principle. Accordingly in computer operating system, if a program is accessing a certain part of the memory, then there is a great likelihood that the program will also access the part of the memory within the same locality in the next time period. In our EFH, we contend that there is a very high probability that the data model within a small time window is the same as the model of data samples in the previous time window. The locality proposition is valid if we assume that the time window is small enough. For the CBR flow, since the cell rate is constant [22, 23], the cell rate at any particular time period is the same as that of the preceding time period. For VBR flow, which can be described by a *two-phase burst/silence* model [2, 24, 25, 26, 27], cells can be sent equidistantly during the burst period and no cells are transmitted during the silence period. The cell rate during the burst period can be approximated based on the principle of "locality". But at the edge of the burst period and the silence period or vice versa, significant error may occur. This kind of prediction error can be tolerated if the time window is sufficiently small. Based on this justification, we can train the system using the previous data flow to approximate the expected data model of the subsequent time period. The smaller the time window, the more flexible the EFH adapts to the cell flow. The best chromosome after an evolution process will be used to do scheduling in the next time period.

To address the scalability issue, we adopt an evolutionary granularity at the fuzzy rule level. In the EFH for Packet Switching, a chromosome can be represented as a string of 25 integers. Each gene of a chromosome represents a fuzzy rule. For this scheme, the search space is not too big compared to the search space in [2, 9], in which each chromosome is represented by a string comprising of hundreds of integers or thousands of bits. The evolution time in the EFH is thus manageable. The third issue to address is the termination of evolution. In many EHW systems, the evolution system may require thousands of generations to get close to an optimal chromosome. The extent of evolution time may limit the applicability of the system for real-time application. In [2], in order to get a good functional EHW to do ATM cell scheduling, the system evolved for 2500 generations with a population size of 400. In [9], in order to derive a circuit with Gaussian output voltage characteristic, the Evolvable hardware system has to evolve 10000 generations. The time scale for evolution in these reported works is not appropriate if used in real-time intrinsic EHW control system. For comparison, in the proposed EFH, a very small population size and small number of the generations are important features of the evolutionary process. In order to prevent the system from adopting a very poor performing fuzzy rule set, we defined a core rule set in the system derived based on the analysis of the problem through human intuition. The core rule set is also used as the startup rule set. If the EFH system is not able to find a chromosome that is better than the core rule set within a fixed number of generations, the core rule set is adopted. The appropriate number of generations for each evolutionary cycle is determined through experimentation. The objective of the evolution is to get a fuzzy rule set better than the working chromosome for the cell flow of the following time period. Even if the derived fuzzy rule set is not optimal, it is deemed to be sufficient. By adopting this idea, the criterion for the termination of evolution can be satisfactorily managed.

9.4 Evolution Scheme

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

9.4.1 Genetic Coding

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

Suppose a fuzzy system has q input variables x_1, x_2, ..., x_q and single output control variable y, a typical rule for the fuzzy system will be "*if* $< x_1$ is $A_1 >$ and $< x_2$ is $A_2 >$ ... *and* $< x_q$ is $A_q >$ *then* $< y$ is $D >$". A_1, A_2, ..., A_q and D are fuzzy concepts or linguistic values. Usually, the development of a fuzzy system involves specifying a finite set of labels to represent the linguistic values for describing each of the variables. If the number of labels for the input variables x_1, x_2, ..., x_q are ξ_1, ξ_2, ..., ξ_q respectively, then the number of rules that one can declare will be $\xi_1 \times \xi_2 \times \ldots \times \xi_q$. We refer to this as the maximum or exhaustive rule set. An n-rule fuzzy system would therefore refer to a system with n being less than or equal to $\xi_1 \times \xi_2 \times \ldots \times \xi_q$. This is refered to as an n-rule constrained fuzzy system or simply an n-rule fuzzy system [28, 29, 30].

To begin with, we define two symbols for the inputs, c_1 and c_2. The symbol c_1 refers to the status of *class*$_1$ cell flow, which is a function of V_1 and V_{max}. V_1 is the current cell rate of *class*$_1$ cell flow while V_{max} is the line capacity. The symbol c_2 refers to the buffer status of BUF$_2$. It is a function of L_2 and L_{max}. L_2 is the number of empty units in BUF$_2$ while L_{max} is the length of BUF$_2$. For c_1 and c_2, the memberships are characterized by the term set $\{VS, S, M, L, VL\}$ as depicted in Fig. 9.3. These are standard triangular membership functions. The output SEL of the *fuzzy switching control* block (see Fig. 9.2) is characterized by the term set $\{T, F\}$. Both T and F are singletons, or fuzzy sets with impluse membership functions as shown in Fig. 9.4. Functionally, a T or *true* means that the MP unit allocates time slots to cater for the *class*$_1$ cell flow in BUF$_1$. An output F or *false* implies that switching is reverted to cells in BUF$_2$.

Based on the above characterization of the switching network, it is possible to define the n-rule heuristics to control the switching behavior. With the fuzzy memberships defined, one can rely on intuitive logic to define the necessary input-output mappings as shown in Table 9.1. The 25-rule system serves as the default cell scheduling algorithm on system startup. We refer to this rule set as the core rule set.

A fuzzy rule set can be represented as a string of integers. For example, the genetic code for the 25-rule system in Table 9.1 can be described by the string "2221122111221112111111111". The allelic code 1 and 2 correspond to the labels *true* and *false* respectively. The position of the gene in the string

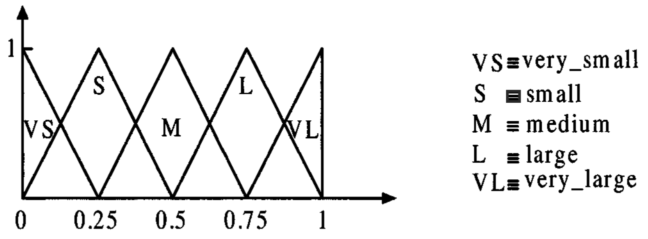

Fig. 9.3. Membership functions for c_1 and c_2

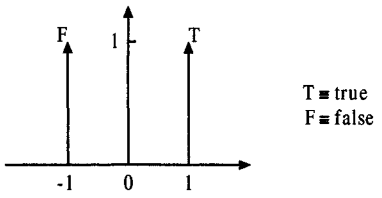

Fig. 9.4. Membership functions for T and F

identifies a specific rule in Table 9.1 when interpreted accordingly in a row wise manner. If the value of a gene is 0, it means that there is no specific fuzzy rule defined for the corresponding input condition. The core rule set not only serves as the startup rule set, but also provides a means to benchmark the performance during the evolution of chromosomes. This scheme guarantees that the performance of the system is better than or at least comparable to that of the core rule set.

Table 9.1. A 25-rule fuzzy system for ATM cell scheduling

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

9.4.2 Inference Scheme

Each entry in Table 9.1 can be interpreted as a statement of the form "if *antecedent*$_1$ and *antecedent*$_2$ then *conclusion*". The *antecedent*$_{\#}$ represents the fuzzy conditions for c_1 or c_2, characterized over the term set $\{VS$, S, M,

L, $VL\}$. The *conclusion* can be T or F. The degree of firing for each fuzzy rule is taken as the minimum of the degrees of matching between the inputs c_1 and c_2 and the antecedents. The aggregation is carried out by averaging the fuzzy conclusions derived from all the rules.

Although we have shown a 25-rule system, for this Evolvable system, the number of the fuzzy rules can vary between 0 and 25. In order to manage the evolution time and reduce the search space, we can fix the size of the rule set to be less than 25 as in [29], so that the evolution time can be managed. This is because the search space for a reduced rule set is more manageable and hence the evolution efficiency can be significantly improved.

9.4.3 Fitness Function

According to the specifications of the problem, the capacity of the output channel is fixed. This implies that no further adjustment on the output capacity can be made to cater for fluctuations in demand. If the bandwidth is not big enough to meet the demand of the two cell flows, servicing $class_1$ cell would mean filling up the $class_2$ buffer and eventually resulting in cell loss for $class_2$. Hence for a specified requirement on the level of cell delay for $class_1$, a certain expected level of $class_2$ cell loss is inevitable. In other words, the $class_2$ cell loss is constrained by the desired level of $class_1$ cell delay that the system is trying to achieve.

There is one main consideration in formulating the fitness function for the EFH. This pertains to the $class_1$ average cell delay. From the above discussion, it is apparent that the level of $class_2$ cell loss is negatively correlated to the average $class_1$ cell delay. Adjusting $class_1$ cell delay will adversely affect the $class_2$ cell loss. Based on these justifications, the fitness function can be described explicitly as in Eq.9.1.

$$F = \kappa - |AveDelay - \lambda \times DelayFactor| \tag{9.1}$$

In Eq.9.1, κ is a very large numerical constant. It is used to adjust the range of fitness values such that F is proportional to the fitness measure of the chromosome. The larger the fitness value, the fitter the chromosome. *AveDelay* is the average delay of $class_1$ cell units after all the cells in TB$_1$ have been processed. *DelayFactor* is a constant used as a reference for scaling the value of λ based on the desired $class_1$ cell delay. λ is an adjustable coefficient to denote the desired level of average cell delay for $class_1$ cell units stored in TB$_1$. In general, the system tries to search for a chromosome with minimum $|AveDelay - \lambda \times DelayFactor|$. Both the *AveDelay* and *DelayFactor* in Eq.9.1 can be determined from Eq.9.2 and Eq.9.3 respectively.

$$AveDelay = \frac{1}{\tau} \times \sum_{i=1}^{\tau} m(i) \tag{9.2}$$

$$DelayFactor = \rho \times v \tag{9.3}$$

In Eq.9.2, $m(\mathrm{i})$ is the waiting time of the ith cell in TB_1 before being sent out. τ is a variable denoting the number of $class_1$ cell units in TB_1 sent during evaluation. $\sum_{i=1}^{\tau} m(i)$ is the sum of the cell delay of the cell units in TB_1. In Eq.9.3, ρ is a constant corresponding to the time required to send a cell through the output channel. The value of ρ depends on the bandwidth capacity of the output channel. The symbol v denotes the size of $\mathrm{TB}_{\#}$. With Eq.9.3, a reference value for the possible delay of $class_1$ cell units can be determined.

9.5 Simulation

In order to demonstrate the viability of the EFH scheme, we carried out simulations of EFH in cell scheduling on two different scenarios. In the simulation, we assume the capacity of the output channel (OUT) and the input channels to be 155.52MHz. The two cell flows are as shown in Fig. 9.5.

For $scenario_1$, $class_1$ is the CBR cell flow with cell bit rate of 155.52MHz. $class_2$ is VBR cell flow, also with a cell bit rate of 155.52MHz. The difference is that the VBR specified has a 2ms ON time period and a 2ms OFF time period. This scenario is a very extreme case used to test the system's controllability. In order to simulate the system performance on a more realistic cell flow, we can adopt $scenario_2$. For $scenario_2$, $class_1$ refers to CBR cell flow with a cell bit rate of 100MHz. $class_2$ is VBR cell flow with unknown random cell bit rate. The minimum cell bit rate for VBR is 55.52MHz while the maximum is 155.52MHz. In these two scenarios, since the sum of the CBR and VBR cell rate is larger than the OUT channel's capacity, cell loss is unavoidable. From

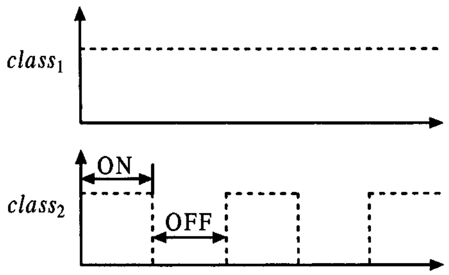

Fig. 9.5. Two classes of cell flows

a practical point of view, the second scenario is more likely compared to the first scenario.

The simulation results are compared with the results of *first-in first-out* (FIFO) and *dynamically weighted priority scheduling* (DWPS) [24]. FIFO is a very traditional scheduling method. It schedules the cell flows based on the arrival time of the packets. FIFO can achieve very good balance between *class*$_1$ cell loss and *class*$_2$ loss, but it is very bad in terms of *class*$_1$ cell delay performance. DWPS is a very good algorithm for cell scheduling. It adjusts the priority according to the cell flow scenarios. But the adaptation scheme of DWPS is not very efficient if the cell flow changes dramatically. DWPS can be described by Eq.9.4. In Eq.9.4, v_i is the fixed priority for different cell flow inputs, a lower value indicates a higher priority. $T_i(t)$ is the waiting time of the oldest packet in the buffer of the ith channel. Q_i is the priority index associated with each cell. The lower the value, the higher the priority. γ is an emphasis parameter and the recommended value is 0.9.

$$Q_i = \frac{v_i}{[T_i(t)]^\gamma} \tag{9.4}$$

9.5.1 Simulation Results

For the simulation, the size of BUF$_1$ and BUF$_2$ is 100 cells, and the size of TB$_1$ and TB$_2$ is 300 cells. In the fitness function, λ is 0.35. All the simulations are carried out by using a C++ program. The setting for the parameters of the evolutionary algorithm is as follows:

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

We simulated each scheduling scheme for cell flows lasting 2 seconds. Fig. 9.6 and 9.7 are the simulation results of FIFO, DWPS and EFH schemes on *scenario*$_1$. Fig. 9.8 and 9.9 are simulation results for FIFO, DWPS and EFH schemes on *scenario*$_2$. The simulation results demonstrate the viability of the evolution scheme and that EFH can fulfill the cell scheduling task. For *scenario*$_1$, EFH can achieve lower *class*$_1$ cell delay than FIFO and DWPS. The balance of *class*$_1$ and *class*$_2$ cell loss by using these three methods is acceptable. None of the schemes show significant bias towards any of the two cell flows. For *scenario*$_2$, the situation is quite different. EFH can still achieve lower *class*$_1$ cell delay with an acceptable balance between *class*$_1$ cell loss and *class*$_2$ cell loss. The *class*$_1$ cell delay by using DWPS is higher than that of EFH and the balance between the *class*$_1$ cell loss and the *class*$_2$ cell loss is not good. So according to the quality factors as discussed in Section 9.2, EFH can control the cell scheduling better than FIFO and DWPS when the cell flow changes dramatically.

9.5.2 Tunability of EFH

One advantageous property of EFH is that the system performance can be adjusted very intuitively by decreasing or increasing the value of λ in Eq.9.1. The

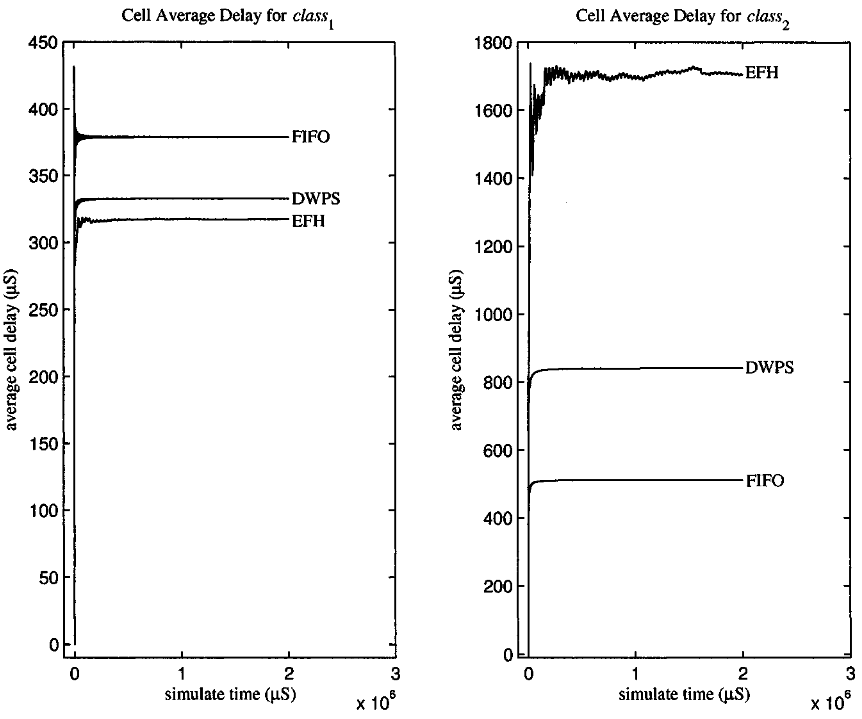

Fig. 9.6. Cell delay for $class_1$ and $class_2$ in $scenario_1$

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

The results of the simulation with different values of λ for $scenario_1$ and $scenario_2$ are as shown in Fig. 9.10, 9.11, 9.12 and 9.13. In Fig. 9.10 and 9.11, when λ is 0.4, the $class_1$ cell delay and $class_1$ cell loss are very small. Accordingly, the $class_2$ cell delay and cell loss are significant. If good balance of $class_1$ cell loss and $class_2$ cell loss is desired, a bigger value can be assigned to λ. In Fig. 9.10 and 9.11, both the $class_1$ cell loss and $class_2$ cell loss are moderate when λ is 0.6. For situations where QoS for $class_2$ needs to be significantly emphasized, the value of λ can be increased. The larger the value for λ, the better the QoS for $class_2$. For example, it is clear from the plots in Fig. 9.10 and 9.11 that λ=0.8 offers good QoS for $class_2$.

For the simulation results in Fig. 9.12 and 9.13 on $scenario_2$, the same conclusion can also be derived. In principle, $class_1$ cell delay can be adjusted in the range from 0 to $\rho \times v$ if λ is between 0 and 1. This means that $class_1$ cell delay has a very wide range of tunability. It further implies that $class_1$ cell loss and $class_2$ cell loss are also tunable to a wide range. According to the

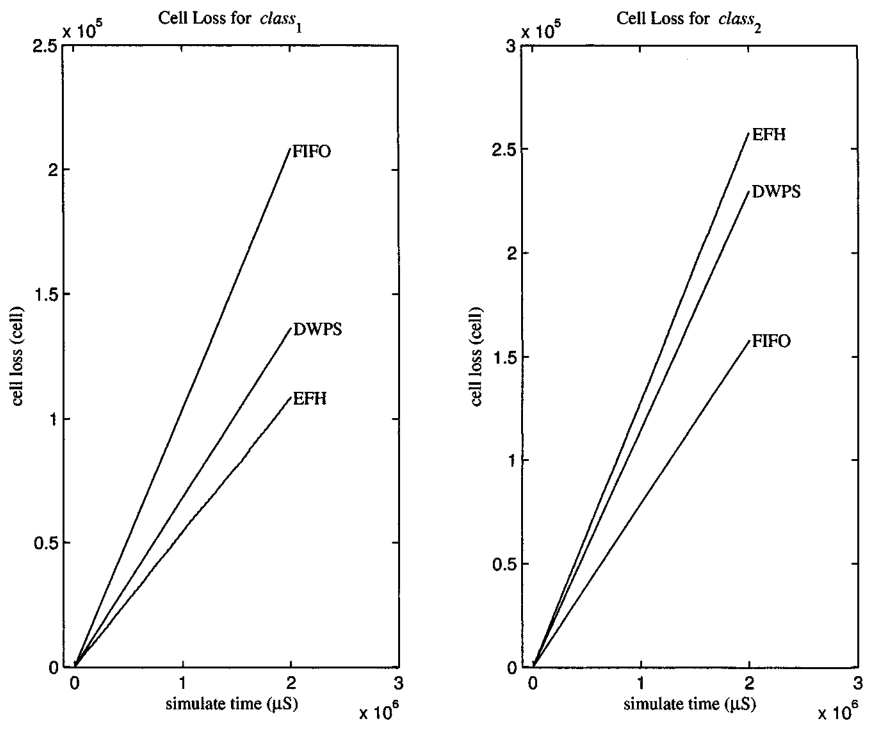

Fig. 9.7. Cell loss for $class_1$ and $class_2$ in $scenario_1$

fitness function, the acceptable level of $class_1$ cell delay can be decided based on the value of λ. On the other hand, if one can decide on the satisfactory $class_1$ cell delay to be achieved, the value of λ can also be approximated.

9.6 Hardware Implementation

According to the evolution scheme described by Fig. 9.2 in Section 9.4, the chromosomes need to be evaluated within a very short time period for each evolution. If the whole evolution process can be completed within the time it takes to send one cell packet through the OUT channel, and a good fuzzy rule set can be found during this time period, the system will enjoy the greatest flexibility in adapting to the changing environment. On the whole, the performance of the system is very much dictated by the quality of the rule set being applied. Each rule set instance is referred to as a context, and is applicable to the current scenario of the operating environment. As context changes, the fuzzy inference circuit is required to accommodate the new context without incurring significant overhead for setup. This implies that a reconfigurable high-speed fuzzy inference circuit is very critical in EFH. In order to achieve

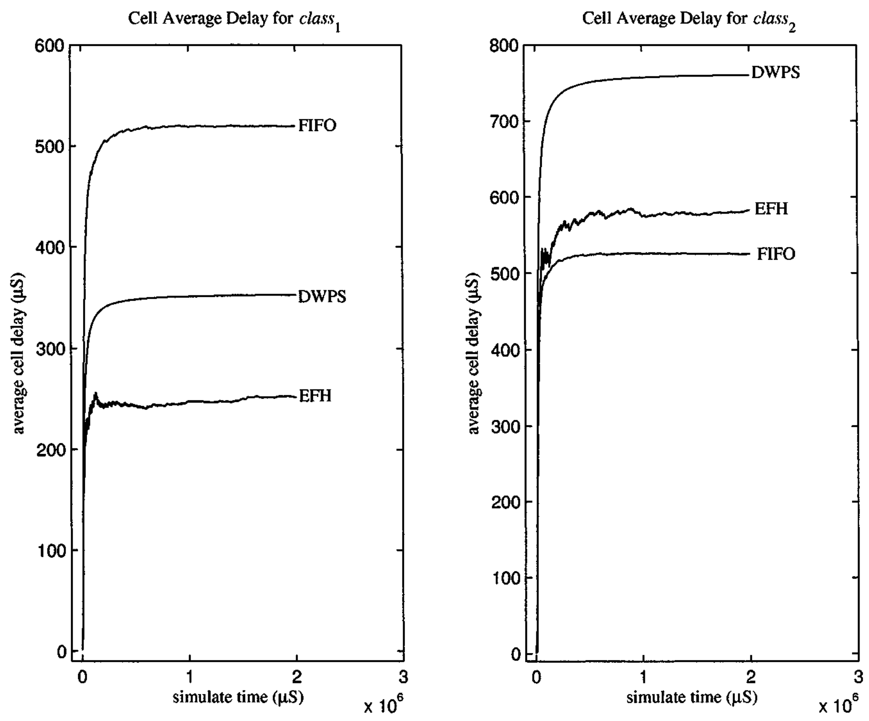

Fig. 9.8. Cell delay for $class_1$ and $class_2$ in $scenario_2$

fast fuzzy inference and at the same time accommodate real-time online context updating, we have proposed a hardware scheme for fuzzy inference called *reconfigurable fuzzy inference chip* (RFIC) [21].

The novelty of the RFIC lies in its ability to accommodate an online context change without interrupting the system operation. The block architecture of RFIC is as shown in Fig. 9.14. The main component is the FIM (*fuzzy inference map*) block. It adopts an implicit inference approach to deliver high inference speed for applications with dynamically changing contexts. The current applicable context is managed by the CMU (*context management unit*). It stores the working fuzzy context and generates control signals such as $Ena_{<x,y>}$ and $Sel_{<x,y>}$ for the FIM. AEM (*address encoding mechanism*) is the module that generates the address to access the FIM partition blocks activated by the $Ena_{<x,y>}$ signals. The OAM (*output aggregation mechanism*) is the dedicated circuit for fuzzy inference aggregation.

The proposed EFH system for cell scheduling is able to accommodate fuzzy rule sets of up to 25 fuzzy rules. Hence, the FIM block incorporates 25 PBs (*partition blocks*); $PB_{<1,1>}$, $PB_{<1,2>}$ ... $PB_{<5,5>}$. Each PB is a mapping that accommodates all the input situations with specific outputs. The mapping for each PB is created based on a software fuzzy inference model. To illustrate

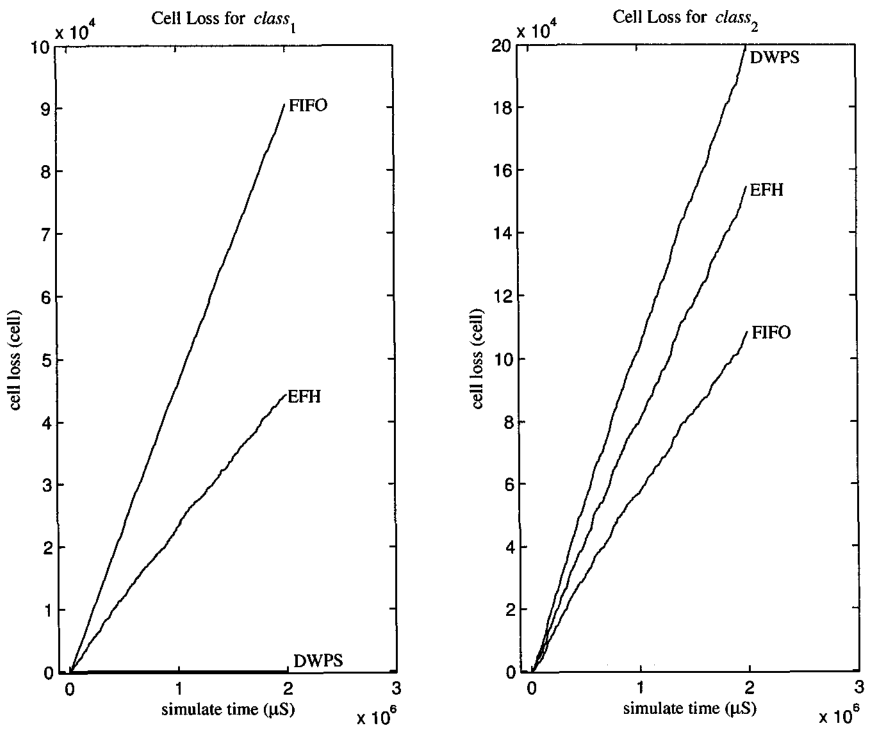

Fig. 9.9. Cell loss for *class*$_1$ and *class*$_2$ in *scenario*$_2$

the basic structure and format of each PB, we can assume that the inputs and the membership functions are digitized to 5 bits. A sample of the mapping data for $PB_{<1,1>}$ is presented in Table 9.2 for illustration. The left column of the table lists the addresses. The whole address string is composed of three parts, i.e., the digitized values of $Input_1$, $Input_2$ and $Sel_{<1,1>}$. The data are made up of two parts. The most significant bit is the fuzzy conclusion bit indicating T or F. The other bits represent the degree of firing for the corresponding fuzzy rule. For example, refering to the first memory unit in Table 9.2, where both $Input_1$ and $Input_2$ equal to "00000", the degree of matching to the membership function *VS* is "11111". So the corresponding datum in the location is "0,11111". Its first bit "0" represents the fuzzy conclusion T and the other bits "11111" is the firing strength.

CMU stores the current application context and generates $Ena_{<x,y>}$ and $Sel_{<x,y>}$ signals. For the application described, the size of the context register required is 50 bits. Each two-bit datum in the register represents a fuzzy rule. The position of each two-bit datum in the 50-bit string identifies the specific rule of the context. A "01" means the fuzzy conclusion is T and "10" indicates the fuzzy conclusion is F. A "00" means that there is no fuzzy rule for the corresponding input situation. Each $Ena_{<x,y>}$ signal can be generated

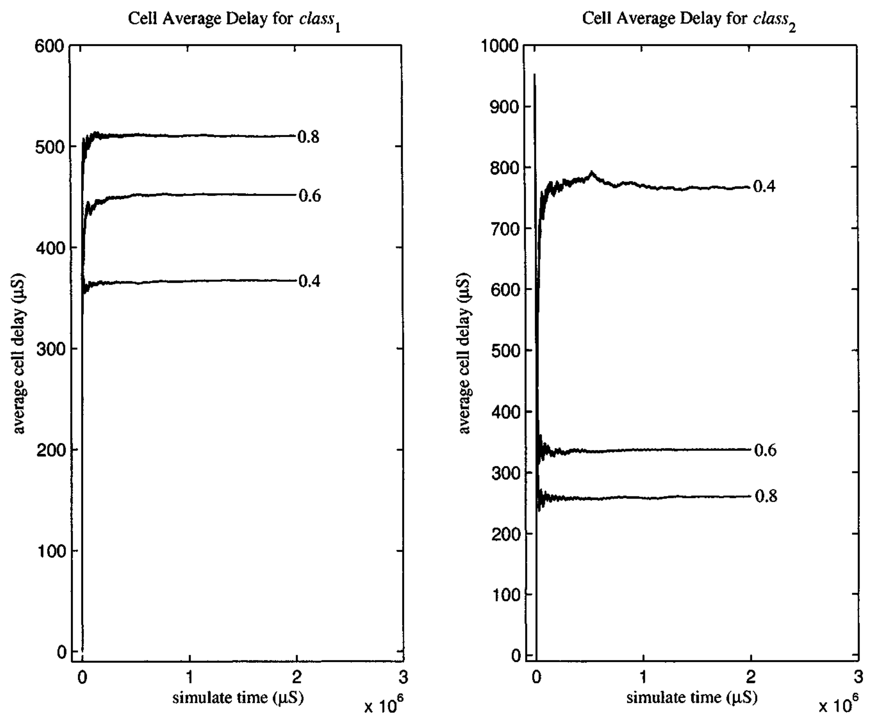

Fig. 9.10. Cell delay for *scenario*1

by applying the logical *OR* operation to the corresponding two bits. A value of "1" for $\text{Ena}_{<x,y>}$ indicates that $\text{PB}_{<x,y>}$ is enabled, which otherwise is disabled. $\text{Sel}_{<x,y>}$ also depends on the specific two bits and is connected to $\text{PB}_{<x,y>}$ separately. A "01" generates a "0" for $\text{Sel}_{<x,y>}$ and "10" produces a "1". The circuit for OAM is as shown in Fig. 9.15. It is made up of Ave_2 blocks and Ave_3 block. In this circuit, the most significant bit of each datum shown in Table 9.2 involve in the aggregration operation is a sign bit. The output has 5 more bits than the input data in order to preserve calculation precision. The control output is derived from the sign bit, i.e, the most significant bit of the OAM output. A positive value indicates that the inference conclusion is T and a negative means the conclusion is F.

9.7 Conclusions

There are several challenges to the application of Evolvable hardware for solving time critical problems. We highlighted three issues, namely *online adaptation*, *scalability* as well as *termination of evolution*. To realize EHW capable of intrinsic online evolution, these issues have to be considered. In this chapter,

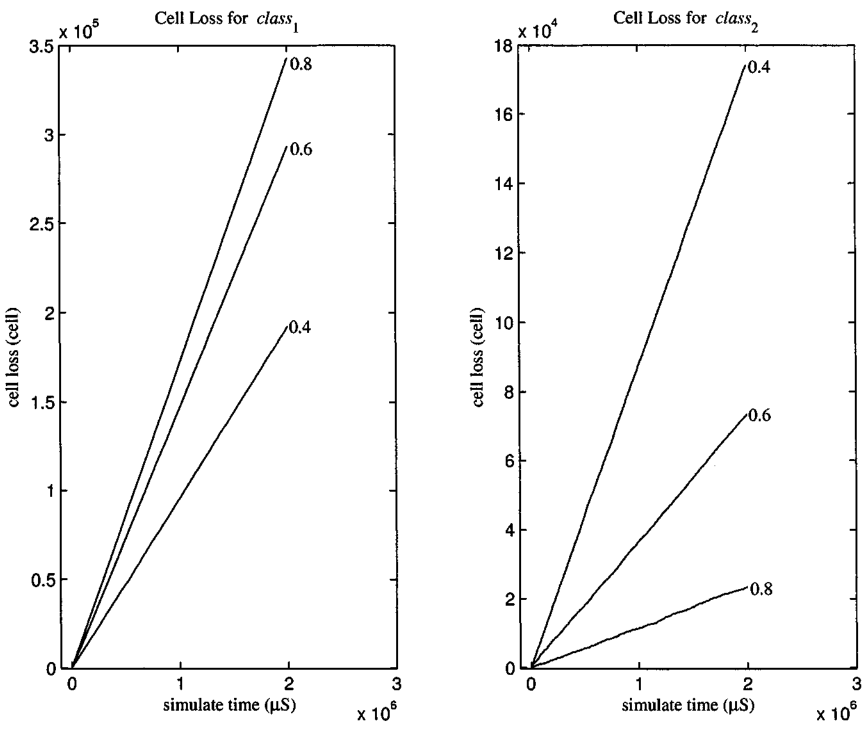

Fig. 9.11. Cell loss for *scenario*$_1$

Table 9.2. FIM content in $PB_{<1,1>}$

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

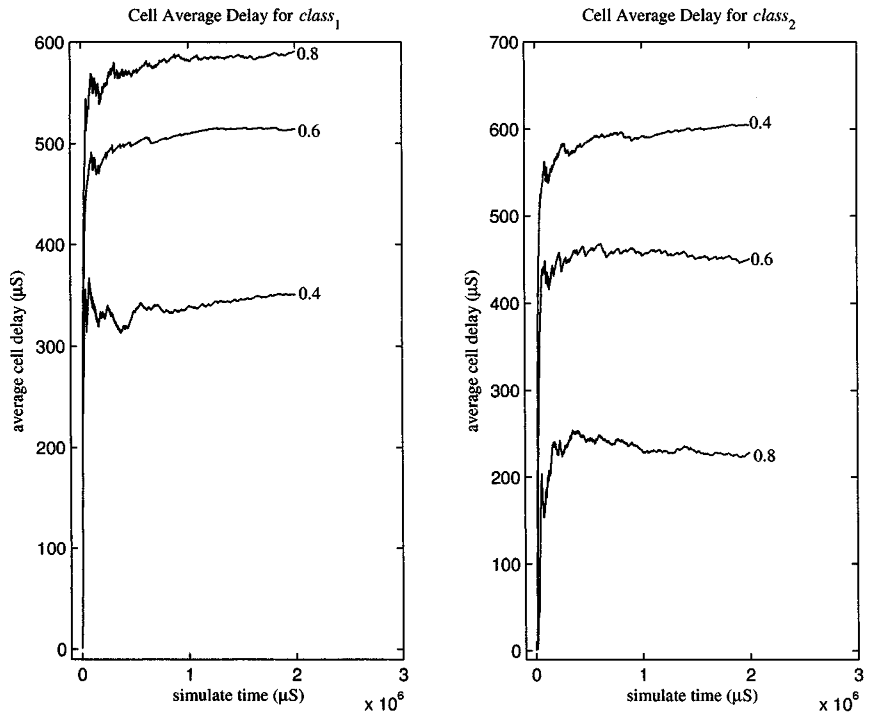

Fig. 9.12. Cell delay for *scenario*$_2$

we proposed the EFH scheme, a form of EHW whereby the fuzzy inference scheme is carried out in hardware to achieve real-time operation. The scheme allows for updating of online context and domain rules and further incorporating mechanisms to evolve a context appropriate for the application scenario. In order to demonstrate the viability of our proposed EFH, we simulated the control performance of the EFH in cell scheduling and compared the results with some traditional scheduling methods. From the simulation results, it can be seen that the EFH is capable of dealing with changing cell flows much better than the traditional methods. Another significant advantage of the EFH is tunability. This was also analyzed based on the simulation results. Based on analysis of the simulation results, the EFH possesses significant advantages over conventional scheduling methods. To implement the EFH, we described the hardware implementation based on a context switchable RFIC to achieve real-time high-speed fuzzy inferencing and high-speed context updating. By combining this hardware scheme and the evolution scheme, an online adaptive and intrinsic Evolvable EFH can be potentially realized using system-on-chip technology. Although we demonstrated the application of EFH on Packet Switching, the application of EFH is not limited to this. Some real-time control problems such as packet control in parallel computer, token control in

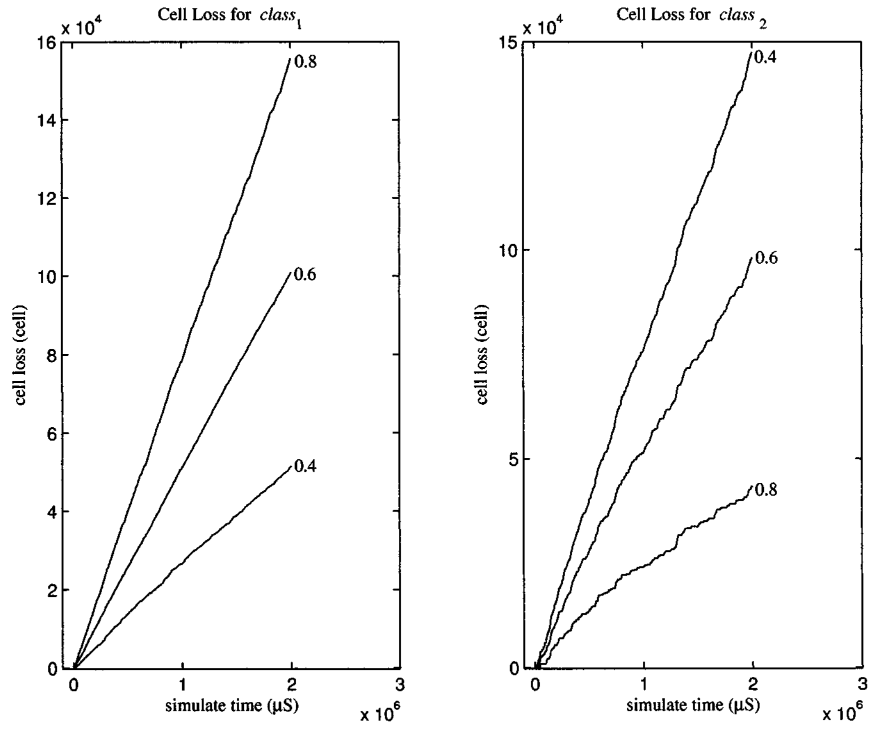

Fig. 9.13. Cell loss for *scenario*$_2$

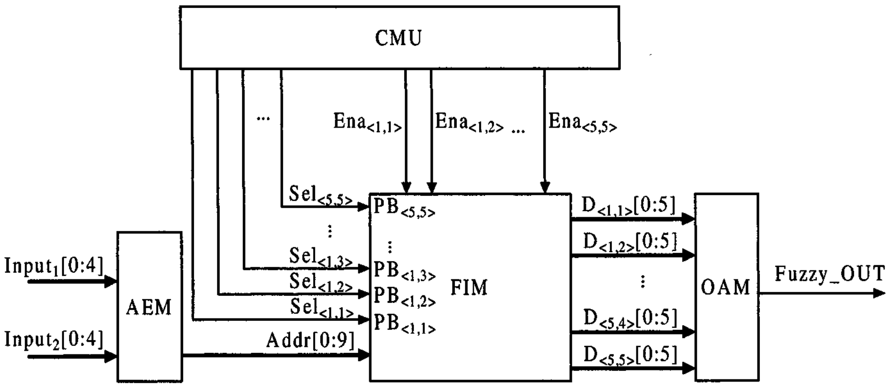

Fig. 9.14. Block architecture of RFIC

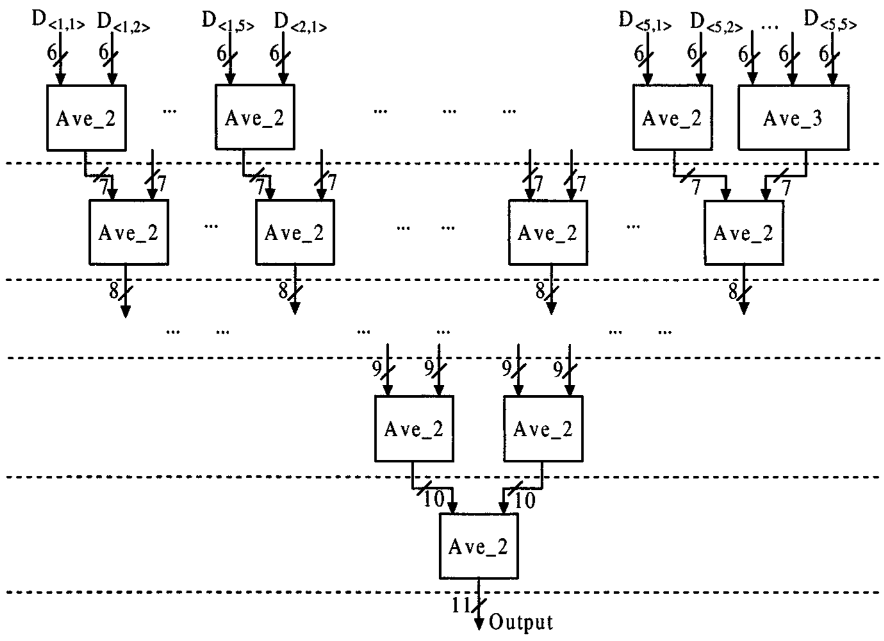

Fig. 9.15. The hardware architecture of OAM

data flow machine, cell flow control in future communication networks are potentially suitable application areas.

References

1. X. Yao and T. Higuchi, "Promises and Challenges of Evolvable Hardware", *IEEE Trans on Syst., Man and Cybern.*, Part C, Applications and Reviews, vol.29, no.1, pp: 87–97, Feb. 1999.
2. W.X. Liu, M. Murakawa and T. Higuchi, "ATM Cell Scheduling by Function Level Evolvable Hardware", LNCS 1259 (ICES1996): pp. 180-192
3. W.X. Liu, M. Murakawa and T. Higuchi, "Evolvable Hardware for On-line Adaptive Traffc Control in ATM Networks", *Genetic Programming 1997, Proc. of the Second Annual Conference*, pp.504–509, Morgan Kaufmann Publishers, 1997.
4. T.G.W. Gordon and P.J. Bentley, "On Evolvable Hardware", In Ovaska, S. and Sztandera, L. (Ed.) *Soft Computing in Industrial Electronics.* Physica-Verlag, Heidelberg, Germany, 2002, pp. 279-323
5. H. D. Garis, "Evolvable Hardware: Principles and Practice", http://www.cs.usu.edu/~degaris/papers/CACM-E-Hard.html
6. M. Iwata, I. Kajitani, H. Yamada, H. Iba and T. Higuchi, "A pattern recognition system using Evolvable hardware", in *Proc. Int. Conf. Parallel Probl. Solving Nature* (PPSN'96).

7. E. Sanchez, *Towards Evolvable hardware: the evolutionary engineering approach,* Berlin; New York: Springer, c1996.
8. K.C. Tan, C.M. Chew, K.K. Tan, L.F Wang and Y.J. Chen, "Autonomous Robot Navigation via Intrinsic Evolution", *Proc. of the 2002 Congress on Evolutionary Computation,* 2002 (CEC'02). vol.2, 2002 pp.1272–1277
9. J. Langeheine, K. Meier and J. Schemmel, "Intrinsic Evolution of Quasi DC Solutions for Transistor Level Analog Electronic Circuits Using a CMOS FPTA Chip". *Proc. NASA/DoD Conference on Evolvable Hardware,* 2002, pp.75–84
10. F.H. Bennett, J.R. Koza, M.A. Keane, J. Yu, W. Mydlowee and O. Stiffelman, "Evolution by Means of Genetic Programming of Analog Circuits that Perform Digital Functions", *Proc. of the Genetic and Evolutionary Computation Conference,* July 13-17, 1999, Orlando, Florida.
11. T. Higuchi, M. Iwata, I. Kajitani, M. Murakawa, S. Yoshizawa and T. Furuya, "Hardware Evolution at Gate and Function Levels," *Proc. Biologically Inspired Autonomous Systems: Computation, Cognition and Action,* Durham, North Carolina, March, 1996.
12. D. Keymeulen, K. Konada, M. Iwata, Y. Kuniyoshi and T. Higuchi, "Robot Learning using Gate-Level Evolvable Hardware", In A. Birk and J. Demiris, (ed.), *Proc. of the Sixth European Workshop on Learning Robots,* Lecture Notes in Artificial Intelligence, Springer-Verlag, 1998.
13. M. Iwata, I. Kajitani, Y. Liu, N. Kajihara and T. Higuchi, "Implementation of a Gate-Level Evolvable Hardware Chip", LNCS 2210 (ICES2001), pp. 38-49, Springer Verlag, 2001.
14. D. Keymeulen, M. Durantez, K. Konaka, Y. Kuniyoshi and T. Higuchi, "An Evolutionary Robot Navigation System using a Gate-Level Evolvable Hardware", LNCS 1259 (ICES1996), pp.195-209, Springer Verlag, 1996.
15. I. Kajitani, T. Hoshino, D. Nishikawa, H. Yokoi, S. Nakaya, T. Yamauchi, T. Inuo, N. Kajihara, M. Iwata, D. Keymeulen and T. Higuchi, "A gate-level EHW chip: Implementing GA operations and reconfigurable hardware on a single LSI", *Evolvable Systems: From Biology to Hardware* (ICES1998), LNCS 1478, pp.1-12, Springer Verlag, 1998.
16. H. Iba, M. Iwata and T. Higuchi, "Gate-Level Evolvable Hardware: Empirical Study and Application", In D. Dasgupta and Z. Michalewicz, editors, *Evolutionary Algorithms in Engineering Applications,* pp.260-275, Springer-Verlag, Berline,1997.
17. H. Iba, M. Iwata and T. Higuchi, "Machine Learning Approach to Gate-Level Evolvable Hardware", *Evolvable Systems: From Biology to Hardware* (ICES1996), LNCS 1259, pp.327-343, Springer-Verlag, 1997.
18. T. Higuchi, M. Murakawa, M. Iwata, I. Kajitani, W. Liu and M. Salami, "Evolvable Hardware at Function Level", *Proc. of 1997 IEEE Int. Conf. on Evolutionary Computation* (ICEC97), pp. 187-192, 1997.
19. M. Murakawa, S. Yoshizawa, I. Kajitani, T. Furuya, M. Iwata and T. Higuchi, "Hardware Evolution at Function Level", *Parallel Problem Solving from Nature–PPSN IV,* LNCS 1141, pp.62-71, Springer-Verlag, 1996.
20. J.H. Li and M.H. Lim, "Evolvable fuzzy system for ATM cell scheduling", *Proc. of 5th Int. Conf. Evolvable Syst.: From Biology to Hardware* (ICES 2003) LNCS 2606, pp. 208-217, Springer-Verlag, 2003.
21. Q. Cao, M.H. Lim and J.H. Li, "A context switchable fuzzy inference chip," submitted to *IEEE Trans. on Fuzzy Syst.*.

22. ATM Forum, "ATM Traffic Management Specification 4.0", April 1996, ftp://ftp.atmforum.com/pub/approved-specs/af-tm-0056.000.pdf
23. R. Jain, "Congestion Control and Traffic Management in ATM Networks: Recent Advances and A Survey," *Computer Networks and ISDN Systems*, vol.28, no.13, October 1996, pp. 1723-1738.
24. T. Lizambri, F. Duran and S. Wakid, "Priority Scheduling and Buffer Management for ATM Traffic Shaping", *Proc. of 7th IEEE Workshop on Future Trends of Distributed Computing Systems, FTDCS'99*, pp.36-43, Dec.20-22, 1999, Cape Town, South Africa.
25. E.P. Rathgeb, "Modeling and Performance Comparison of Policing Mechanisms for ATM Networks", *IEEE J. Select. Areas Commun.*, vol.9, no.3, April 1991.
26. B. Maglaris, D. Anastassiou, P. Sen, G. Karlsson and J.D. Robbins, "Performance models of statistical multiplexing in packet video communications," *IEEE Trans. Commun.*, vol.36, no.7, pp.834-844, July 1988.
27. R. Guerin, H. Ahmadi, M. Naghshineh, "Equivalent Capacity and Its Application to Bandwidth Allocation in High-Speed Networks," *IEEE J. Select. Areas Commun.*, vol.9, no.7, pp968-981, Sept. 1991.
28. M.H. Lim, S. Rahardja and B.H. Gwee, "A GA paradigm for learning fuzzy rules", *Fuzzy Sets and Systems* 82(1996), pp.177-186.
29. M.H. Lim and W.L. Ng, "Iterative Genetic Algorithm for Learning Efficient fuzzy rule Set", to appear in *AIEDAM*, 2004.
30. B. Kosko, *Neural Networks and Fuzzy Systems*, Prentice Hall, Englewood Cliffs, NJ, 1992.