An Adaptive Fuzzy Logic Controller: Its VLSI Architecture and Applications

Jer Min Jou, Pei-Yin Chen, and Sheng-Fu Yang

***Abstract*—Most previous work about the hardware design of a fuzzy logic controller (FLC) intended to either improve its inference performance for real-time applications or to reduce its hardware cost. To our knowledge, there has been no attempt to design a hardware FLC that can perform an adaptive fuzzy inference for the applications of on-line adaptation. The purpose of this paper is to present such an adaptive memory-efficient FLC and its applications. Taking advantage of the adaptability provided by a symbolic fuzzy rule format and the dynamic membership function generator, as well as the high-speed integration capability afforded by VLSI, the proposed adaptive fuzzy logic controller (AFLC) can perform an adaptive fuzzy inference process using various inference parameters, such as the shape and location of a membership function, dynamically and quickly. Three examples are used to illustrate its applications, and the experimental results show the excellent adaptability provided by AFLC.**

***Index Terms*—Adaptive fuzzy system, fuzzy logic controller, membership function generator.**

I. Introduction

In THE nearly 30 years since Zadeh first proposed the idea [1], a variety of applications of fuzzy logic have been implemented in various fields ranging from industrial control to financial management. Most notably, fuzzy logic controllers (FLC's) [2]–[5], based on expert systems and fuzzy logic, have been successfully applied to control vague, incomplete, and ill-defined systems.

Physical systems are usually subject to long-term and permanent changes due to wear and tear on the physical plant parts as well as on the sensors [6]. Such changes mean that the underlying process model may begin to fail. Although there are many successful applications about the fuzzy logic control, conventional fuzzy logic systems cannot adapt to the gradual changes in their operating environments. Therefore, it is important to develop a special fuzzy system, called the adaptive fuzzy system [6], that can learn from experience to improve its figure of merit.

A basic adaptive fuzzy system consists of two components: an adaptive fuzzy logic controller (AFLC) and a learning mechanism. Using the tuning information provided by the learning mechanism to adjust the inference parameters used by the AFLC, the adaptive fuzzy system can work well and maintain consistent performances even when external environments are changed. For on-line applications, the AFLC should be able to manipulate various inference parameters quickly and dynamically. This paper discusses the issues involved in the design of such a special-purpose hardware AFLC, and presents some experimental cases of its applications.

The proposed AFLC takes advantage of the adaptability provided by a new symbolic fuzzy rule format and the corresponding dynamic membership function generators, as well as the high-speed integration capability afforded by VLSI. These let the AFLC have the capacity of performing an adaptive fuzzy inference process dynamically and quickly. The storage size of AFLC's knowledge base is compact and occupies 1186 bits for 50 fuzzy rules. With the three-stage pipelined architecture, the AFLC, operating at a clock rate of 33 MHz, works very fast and yields an inference rate of 490- and 220-K inferences/s in the *nonadaptive* mode and the *adaptive* mode, respectively.

This paper is organized as follows. Section II reviews the previous work for the hardware FLC and the adaptive fuzzy system. In Section III, some basic concepts about the FLC and the adaptive fuzzy system are summarized. Section IV focuses on the VLSI architecture of AFLC. Three experiments are described in Section V for illustrating the applications of AFLC. Conclusions and remarks are provided in Section VI.

II. Related Work

In the past few years, many hardware implementations of FLC [7] have been proposed to address the real-time requirements of fuzzy-logic-based process controllers. The digital hardware FLC originated from Togai and Watanabe [8]. Then, Watanabe *et al.* proposed an FLC with dynamically reconfigurable, cascadable architecture in [9]. An optimized architecture for fuzzy inference is presented by Chiueh in [10]. In [11], Sasaki *et al.* proposed a fuzzy processor using SIMD. Lee and Bien designed an expandable fuzzy inference processor consisting of intermediate-frequency (IF) modules and THEN modules in [12]. To our knowledge, most previous work of FLC intended to improve the inference performance for real-time applications, or to expand the capacity for processing more input and output variables. There has been no attempt to design a hardware FLC that can perform an adaptive fuzzy inference process using various parameters dynamically and quickly [7].

The self-organizing fuzzy logic system, one of the earliest adaptive fuzzy systems, provides an ability to change inferring policies with respect to the process and the operating environment [13]. Wu *et al.* proposed a rule self-regulating fuzzy logic system in [14]. Furthermore, Daugherity *et al.* [15] designed a self-tuning fuzzy logic system based on the concept of fuzzy meta rules. In [16], Lee has combined the adaptive heuristic

Manuscript received August 11, 1998; revised December 26, 1998. This work was supported in part by the National Science Council, R.O.C., under Grant NSC-84-2215-E-006-025.

J. M. Jou and S.-F. Yang are with the Department of Electrical Engineering, National Cheng Kung University, Tainan 70101, Taiwan, R.O.C.

P.-Y. Chen is with the Department of Electronic Engineering, Southern Taiwan University of Technology, Tainan 710, Taiwan, R.O.C.

Publisher Item Identifier S 1063-8210(00)00002-0.

critic (AHC) model of Barto *et al.* [17] with fuzzy logic to tune the locations of the consequent membership functions in his system. Berenji *et al.* [18] have proposed a neuro-fuzzy controller architecture that extends the fuzzy-AHC model mentioned above. However, none of them discusses the adaptive fuzzy hardware design.

III. The Adaptive Fuzzy System

In this section, the relevant principles of the FLC and the adaptive fuzzy system are introduced briefly first. Then, an overview of the proposed adaptive fuzzy hardware system is described.

A. Main Principles of the FLC

An FLC is a controller that represents the dynamic behavior of the controlled system by a set of linguistic rules based on expert knowledge and outputs control actions using fuzzy inference process [4]. The rule used in most FLC's usually has the form

> **IF** (a set of conditions is satisfied) **THEN** (a set of consequences can be inferred).

Each rule has an antecedent (or IF) part containing several preconditions, and a consequent (THEN) part that describes the output action. The antecedent and consequent parts are characterized by some membership functions (or fuzzy sets). A membership function S in a universe of discourse $U = \{u_1, \ldots, u_n\}$ can be represented as a set of ordered pairs of an element u_i and its grade of membership function $S = \{(u_i, \mu_S(u_i))|u_i \in U\}$, where the values of μ_S can vary between zero and one.

Like most fuzzy logic controllers, the FLC discussed here applies the concepts of fuzzy implication and the compositional rules of inference for approximate reasoning. Suppose that we have a two-input (X and Y) and one-output (O) fuzzy system with r fuzzy rules of the form

$$\text{Rule } i\text{: if } X \text{ is } A_i \text{ and } Y \text{ is } B_i \text{ then } O \text{ is } C_i \quad \text{for } i = 1, 2, \ldots, r$$

where A_i and B_i are the antecedent membership functions associated with the linguistic input variables X and Y, respectively, and C_i is the consequent membership function associated with the linguistic output variable O. The fuzzy inference process of an FLC can be depicted in several steps. Since the inputs from a plant are crisp values, a fuzzification operator $fuzz$ is used first. Fuzzification could be defined as a mapping from an observed input space to membership functions in certain input universes of discourse. With the operation, an FLC can cover the ignorance of input measures. Suppose that we have two crisp inputs x and y; the fuzzified inputs X and Y are given by

$$X(u) = \text{fuzz}(x, f_x)$$

and

$$Y(u) = \text{fuzz}(y, f_y), \qquad \text{with } u \in U \tag{1}$$

where f_x and f_y mean the degrees of fuzziness.

Then, X and Y are simultaneously broadcasted to all control rules to be compared with the antecedent parts. The matching degrees between (X and A_i) and between (Y and B_i) are given by

$$A\alpha_i = \max_u \left(\min_u(X(u), A_i(u))\right)$$
$$B\alpha_i = \max_u \left(\min_u(Y(u), B_i(u))\right). \tag{2}$$

The firing strength of rule i is calculated by

$$\omega_i = \min(A\alpha_i, B\alpha_i). \tag{3}$$

Hence, rule i recommends a control decision as follows:

$$O_i(u) = \min_u(\omega_i, C_i(u)). \tag{4}$$

Last, the combined fuzzy result of all control rules is given by

$$O(u) = \bigcup_i O_i(u) = \max_u(O_1(u), \ldots, O_r(u)). \tag{5}$$

Since the inference process should output some nonfuzzy (crisp) control results practically, it necessitates the use of a defuzzifier. If the center of gravity (COG) algorithm is employed, the final crisp output o can be calculated as

$$o = \sum_u (u \times O(u)) \Big/ \sum_u O(u). \tag{6}$$

B. Main Principles of the Adaptive Fuzzy System

A basic adaptive fuzzy system can be composed of two major components: the AFLC and the learning mechanism. The former performs the adaptive fuzzy inference process defined below. The latter executes a specified learning algorithm and makes changes to the control surface of AFLC to retain nearly optimal performance even the external environment is changed. In general, the learning algorithms can be classified into three classes: supervised learning, unsupervised learning, and reinforcement learning [5], [18]. Regardless of the learning algorithms used, there are five main interconnected means of allowing a fuzzy system to adapt:

1) dynamic adjustment of fuzzy regions;
2) management of contribution weights attached to the fuzzy rules;
3) structural modification of membership function;
4) redefinition of the truth in the fuzzy model;
5) methods of defuzzification.

The first two are the most important [6] and supported in our design.

To implement the above concept of adaptation, we modify (1)–(6) with some adaptive operators. The fuzzy inference process of an adaptive fuzzy system can be described as follows. First, let Γ be an operator applied to tune the degree of fuzziness of a crisp input. Then the adaptive fuzzified inputs can be given by

$$X'(u) = fuzz(x, \Gamma(f_x))$$

and

$$Y'(u) = fuzz(y, \Gamma(f_y)), \qquad \text{with } u \in U. \tag{1'}$$

Also let $\Im_{\text{ant}}$ and $\Im_{\text{con}}$ be operators used to tune the shapes and locations of the antecedent and consequent membership functions. Then the adaptive membership functions are represented by

$$A_i'(u) = \Im_{\text{ant}}(A_i(u))$$

and

$$B_i'(u) = \Im_{\text{ant}}(B_i(u)) \tag{7}$$
$$C_i'(u) = \Im_{\text{con}}(C_i(u)). \tag{8}$$

Thus, we can determine the two matching degrees of the antecedent part as

$$A\alpha_i' = \max_u \left(\min_u(X'(u),\, A_i'(u))\right)$$
$$B\alpha_i' = \max_u \left(\min_u(Y'(u),\, B_i'(u))\right). \tag{2'}$$

The firing strength in each rule is given by

$$\omega_i = \min(A\alpha_i',\, B\alpha_i'). \tag{3'}$$

Then, the operator $\Re$ is applied to increase the weights of the rules that contributed significantly to the output action and to decrease the weights of those that contributed less. Thus, the weighted firing strength is given by

$$\omega_i' = \Re(\omega_i). \tag{9}$$

The final crisp output o' is computed by the following:

$$O_i'(u) = \min_u(\omega_i',\, C_i'(u)) \tag{4'}$$
$$O'(u) = \bigcup_i O_i'(u)$$
$$= \max_u(O_1'(u), \ldots, O_r'(u)) \tag{5'}$$
$$o' = \sum_u (u \times O'(u)) \Big/ \sum_u O'(u). \tag{6'}$$

Obviously, the adaptive operators Γ, $\Im_{\text{ant}}$, $\Im_{\text{con}}$, and $\Re$ should use the tuning information provided by the learning mechanism. How to accommodate the hardware implementation of (1′)–(6′) and (7)–(9) to the functions of the adaptive operators gently is discussed in Section IV.

C. The Proposed Adaptive Fuzzy Hardware System

Based on the concepts discussed above, we proposed an adaptive fuzzy hardware system shown in Fig. 1. It consists of two major components: an AFLC and a learning mechanism with the ability to generate the tuning information used by the AFLC. The AFLC, which can perform adaptive fuzzy inference process, consists of six parts:

1) knowledge base;
2) input fuzzifier;
3) dynamic membership function generator;
4) inference-processing unit;

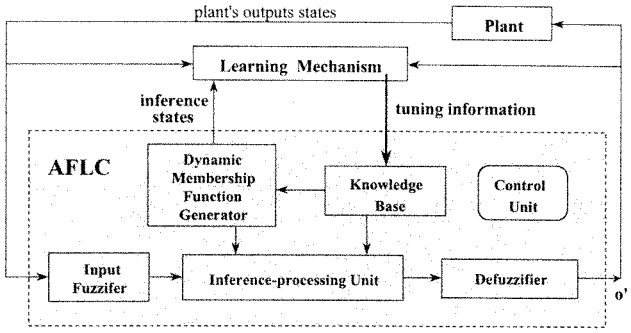

Fig. 1. The proposed adaptive fuzzy hardware system.

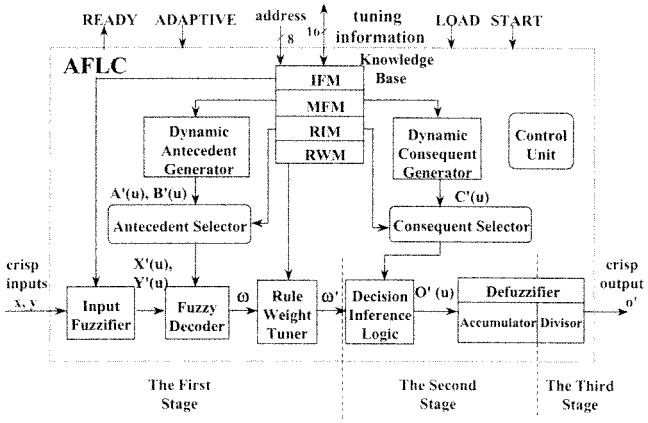

Fig. 2. The basic architecture of AFLC.

5) defuzzifier;
6) control unit.

The detail will be described in the Section IV.

The learning mechanism executes its own learning scheme, modifies the inference parameters stored in the AFLC, and makes the system adapt to the changes of external world. Because there are many diversified learning schemes and most of them are under development [16]–[18], we omit the details about them here.

IV. VLSI Architecture of AFLC

Using CCL 0.8-μm standard cells [19], a prototype realization of AFLC was designed with 50 rules, two input variables, and one output variable. Up to 8 and 16 membership functions are used to specify each input and output variable, respectively. The universe of discourse of a membership function is 64 elements, and the grades of a membership function are discretized into 16 levels. Fig. 2 illustrates the AFLC in more details. Each component of the AFLC is introduced in the following sections.

A. Knowledge Base

The knowledge base of the AFLC consists of a set of fuzzy control rules and the related data used to define the control rules. Since an adaptive fuzzy system may modify its inference parameters frequently, how to store or adjust all tunable inference parameters quickly is a problem. Hence, a memory-efficient symbolic rule format suitable for dynamic adjustment is developed. Four inference parameters—the fuzziness degrees of

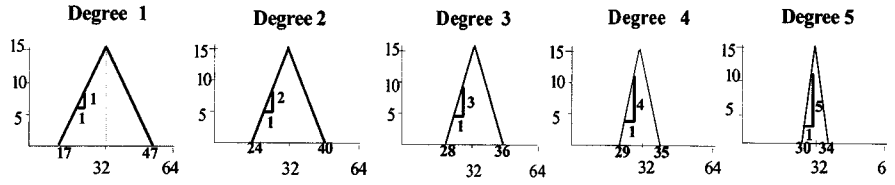

Fig. 3. Five different kinds of fuzziness degrees for a crisp value 32.

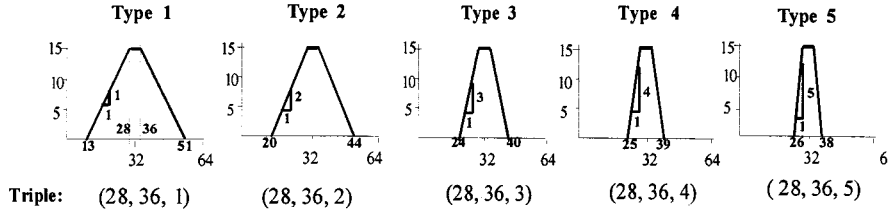

Fig. 4. Five different types of membership functions.

crisp inputs, the shapes and locations of the antecedent and consequent membership functions, the membership functions associated with each fuzzy control rule, and the contribution weight attached to each fuzzy control rule—are tunable. The parameters are represented by the symbolic format and are stored in the different parts of the knowledge base. Each of them is described as follows.

1) *Input fuzziness memory (IFM):* In the design, a fuzziness degree can have a value from one to five. A crisp input can be fuzzified into a triangle-shaped membership function with one of the five fuzziness degrees. Fig. 3 shows the five fuzziness degrees for a crisp input value 32. Hence, 3 bits of binary code are enough to represent the degree of fuzziness for one input.
2) *Membership function memory (MFM):* Each membership function is characterized by a symbolic triple (L_k, R_k, Sel_k), in which L_k and R_k are the leftmost and rightmost points in the upper side of the trapezoid-shaped membership function k, respectively, and Sel_k is a tuning parameter used to select the shape of membership function k. For simpler digital implementation, we implement five different types of symmetrical membership functions with slopes 1, 2, 3, 4, and 5, as shown in Fig. 4. Obviously, it needs 6 bits to represent L_k or R_k and 3 bits for Sel_k.
3) *Rule index memory (RIM):* In the design, each rule is defined by three unsigned values (or indexes), which are used to indicate the associated antecedent and consequent membership functions for that rule. Since each input and output variable can be characterized by using up to eight and 16 membership functions, we need 3 and 4 bits to specify the values of input and output, respectively.
4) *Rule weight memory (RWM):* The contribution weight of a rule determines how much the rule affects the final outcome of fuzzy inference. The contribution weight of each rule is stored in the RWM. For simplification and implementation, we consider only 15 updating ways to tune the weighted firing strength. The 15 possible values of contribution weight are increasing or decreasing the weight by 1, 2, . . ., 7 grades or zero, which means the weighted firing strength is equal to the firing strength. Thus, 4 bits are enough to encode each weight.

B. Dynamic Membership Function Generator

The input of a dynamic membership function generator (DMFG) is a triple, (L_k, R_k, Sel_k), representing the symmetrical membership function k. Its outputs are the grades with respect to the membership function. The DMFG generates 64 grades since the universe of discourse for a membership function is from zero to 63. Let the operator gen represent the function implemented by the DMFG; then antecedent and consequent membership functions can be given by

$$\begin{aligned} A_i(u) &= \text{gen}(L_{A_i}, R_{A_i}, \text{Sel}_{A_i}) \\ B_i(u) &= \text{gen}(L_{B_i}, R_{B_i}, \text{Sel}_{B_i}) \\ C_i(u) &= \text{gen}(L_{C_i}, R_{C_i}, \text{Sel}_{C_i}). \end{aligned} \tag{10}$$

Using (10), (7) and (8) can be modified and implemented in the AFLC as follows:

$$\begin{aligned} A_i'(u) &= \Im_{\text{ant}}(\text{gen}(L_{A_i}, R_{A_i}, \text{Sel}_{A_i})) \\ &= \text{gen}(\Im_{\text{ant}}(L_{A_i}), \Im_{\text{ant}}(R_{A_i}), \Im_{\text{ant}}(\text{Sel}_{A_i})) \\ B_i'(u) &= \Im_{\text{ant}}(\text{gen}(L_{B_i}, R_{B_i}, \text{Sel}_{B_i})) \\ &= \text{gen}(\Im_{\text{ant}}(L_{B_i}), \Im_{\text{ant}}(R_{B_i}), \Im_{\text{ant}}(\text{Sel}_{B_i})) \end{aligned} \tag{11}$$

$$\begin{aligned} C_i'(u) &= \Im_{\text{con}}(\text{gen}(L_{C_i}, R_{C_i}, \text{Sel}_{C_i})) \\ &= \text{gen}(\Im_{\text{con}}(L_{C_i}), \Im_{\text{con}}(R_{C_i}), \Im_{\text{con}}(\text{Sel}_{C_i})). \end{aligned} \tag{12}$$

In other words, an DMFG uses the L_k, R_k, and Sel_k, tuned by the external learning mechanism, to generate the corresponding adaptive membership function.

Fig. 5 shows the block diagram of an DMFG. When constructing the grades of a membership function, the 6-bit counter in the diagram will count down from 63. Using the input triple and the value of the countdown counter, the checking circuit checks the working status and generates some proper signals. The control circuit uses Sel and the left MUX to select an incremental or decremental unit, denoted as Idu. Then, the Idu is sent to the 4-bit adder/subtractor to regenerate the grades of five types of membership functions. When the rising edge of a

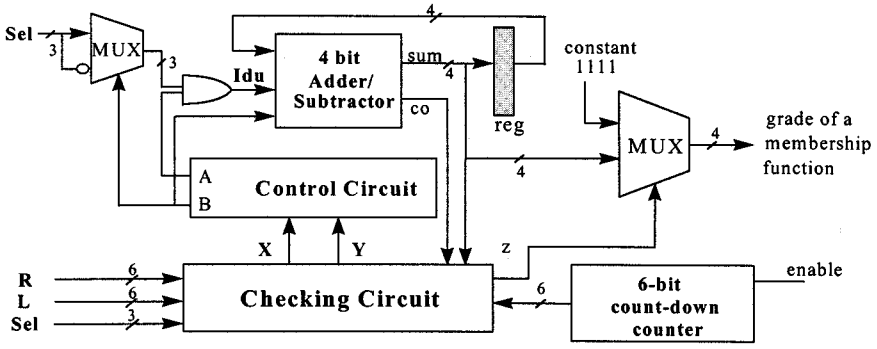

Fig. 5. The dynamic membership function generator.

membership function is generated, the 4-bit adder performs the addition; when the falling edge is generated, it performs the subtraction.

For parallel inference, the generation of each membership function needs a dedicated DMFG. Thus, the dynamic antecedent generator shown in Fig. 2 consists of $8^2 = 16$ identical DMFG's. Additionally, in the AFLC, an input fuzzifier is used to implement the fuzzification operator $fuzz$ on all crisp inputs. Not surprisingly, we find that the function of $fuzz$ is very similar to that of an DMFG. By setting the input triple (L, R, Sel) as $(x, x, \Gamma(f_x))$, an DMFG can implement the $fuzz$ function. Thus, we use the following equations to replace (1′) in the design:

$$X'(u) = \text{fuzz}(x, \Gamma(f_x)) = \text{gen}(x, x, \Gamma(f_x))$$
$$Y'(u) = \text{fuzz}(y, \Gamma(f_y)) = \text{gen}(y, y, \Gamma(f_y)). \quad (13)$$

C. Inference-Processing Unit

The main function of the inference-processing unit is to infer control actions employing fuzzy implication. Since the AFLC can infer all rules in parallel, each rule needs a dedicated inference hardware module. Fig. 6 shows the inference-processing hardware module used for one rule i. Each of the five components in the inference-processing unit is described in the following. The antecedent and the consequent selector are composed of some multiplexers. Using the rule indexes stored in the RIM, the antecedent (consequent) selector properly selects the antecedent (consequent) membership functions attached to each rule and sends them to the fuzzy decoder (decision inference logic).

The fuzzy decoder is applied to find the firing strength of each rule. Using the membership functions, selected by the antecedent selector, and two adaptive fuzzified inputs X' and Y', the antecedent part is compared by the minimum units in the first column. Then, each of the two maximum units determines one matching degree by calculating the peak of the premise comparison. Last, the two matching degrees are used by the minimum unit in the third column to find the firing strength of the rule.

The rule weight tuner is applied to perform the modification operator $\Re$, as in (9). We implement (9) and calculate the weighted firing strength ω_i' using the following:

$$\omega_i' = \Re(\omega_i) = \omega_i + \Delta w_i \quad (14)$$

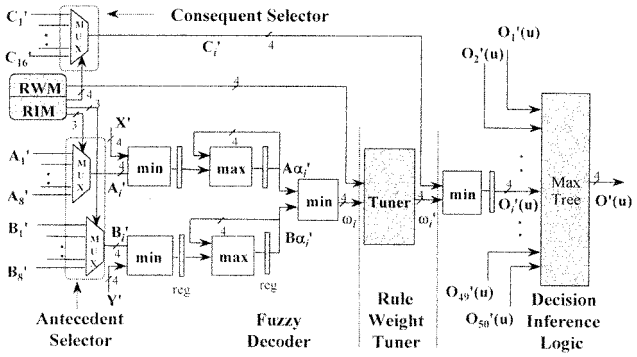

Fig. 6. The inference-processing hardware module for one rule i.

where Δw_i stored in the RWM is the contribution weight, representing the change of weighted firing strength. We consider 15 cases to reflect the weight adjusting: increasing the weight by 1, 2, ..., 7 units, decreasing the weight by 1, 2, ..., 7 units, or zero. Last, the decision inference logic is applied to implement (5′) by using a minimum unit and a maximum tree unit.

D. Defuzzifier

The defuzzifier is applied to compute the COG of the final fuzzy output $O'(u)$. Using (6′), the final crisp output o' can be computed by

$$o' = \sum_{u=63}^{0} u \times O'(u) \Big/ \sum_{u=63}^{0} O'(u)\,. \quad (15)$$

Let P and Q represent the numerator and the denominator of(15), respectively. Thus, P and Q are given by

$$P = 63 \times O'(63) + 62 \times O'(62) + \cdots + 2 \times O'(2) + 1 \times O'(1)$$
$$Q = O'(63) + O'(62) + O'(61) + \cdots + O'(2) + O'(1) + O'(0).$$

Suppose that we define Q_j as follows:

$$Q_j = O'(63) + O'(62) + O'(61) + \cdots + O'(j+1) + O'(j).$$

Then the numerator P can be computed by repeatedly adding the denominator and expressed by

$$P = Q_{63} + Q_{62} + \cdots + Q_j + \cdots + Q_2 + Q_1.$$

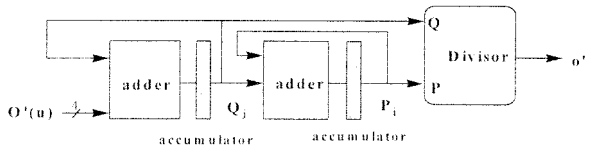

Fig. 7. The defuzzifier.

A two-stage addition/accumulation shown in Fig. 7 can easily perform these computing steps. Last, a divisor is applied to calculate the final crisp output o'.

E. Control Unit

The control unit controls the data flow among the components of AFLC. It also generates the global clock cycle and the stage clock cycle. The global clock cycle is the ordinary clock cycle used to synchronize the whole system. The stage clock cycle is used to control the three-stage pipeline. In addition, the design has three working modes: *load* mode, *adaptive* mode, and *nonadaptive* mode. Three input signals, LOAD, ADAPTIVE, and START, and one output signal READY, shown in Fig. 2, are used to set up the working mode of AFLC.

When the LOAD signal is high, the AFLC enters the *load* mode. In this mode, the AFLC suspends its normal work to load data from learning mechanism. When the LOAD signal is set to low, bringing the START signal high will activate the AFLC to work in the *adaptive* mode or in the *nonadaptive* mode, selected by the ADAPTIVE signal. In the *nonadaptive* mode, the AFLC acts like a conventional FLC. As shown in Fig. 2, the antecedent portion (first stage), the consequent portion (second stage), and the division (third stage) will be performed simultaneously. Each stage needs one stage clock cycle to perform, and the first inference output will come out three stage clock cycles later after the START signal. In the *adaptive* mode, the AFLC works slower to achieve the adaptability. Since the inference parameters of AFLC can be updated by the learning mechanism every inference, no pipelining function is provided in this mode.

F. Implementation and Performance

We have developed the architecture of AFLC using Cadence's Verilog simulator run on a SunSPARC-10 station. Under Verilog simulation, the design can work at 33-MHz clock frequency. This means a clock period of 30 ns. In total, it contains about 164 000 transistors that occupy an area of approximately 6223 × 6046 μm2. Table I summarizes some statistics of AFLC.

The DMFG takes one clock cycle to do its operations. The fuzzy decoder needs 66 clock cycles to find all rules' firing strengths. The rule weight tuner takes another cycle to modify the firing strengths. The decision inference logic takes two clock cycles to find the overall fuzzy output. The accumulator needs 65 cycles to finish its work. Last, the divisor takes 16 clock cycles to output the final crisp result. Hence, the inference rates of the *nonadaptive* mode and the *adaptive* mode are calculated as follows.

In the *nonadaptive* mode, the first stage of pipeline begins to produce the temporary result used by the second stage at the sixty-eighth cycle. The second and third stages need 67 and 16 clock cycles, respectively. Thus, one-stage clock cycle mentioned above is equal to 68 clock cycles. The AFLC can yield an inference rate of nearly 490-K inferences/s in this mode. In the *adaptive* mode, the AFLC works slower, and an inference result will come out every 151-clock cycle. Here, the learning time and tuning time used for adaptation are not considered. Without pipelining, it can yield an inference rate of about 220-K inferences/s.

TABLE I
LAYOUT DATA

Technology	CMOS 0.8μm SPDM
Core size	6223μm x 6046μm
Die size	6478μm x 6478μm
# pin	68
# transistors	164,000
Maximum speed	33MHz
Inf. speed	490 K/220K* inferences/sec
Package type	CLCC
Interface	TTL compatible
Power supply	4.5-5.5 v

*** :non-adaptive/adaptive mode**

G. Comparisons

In the section, we compare both the knowledge base and the DMFG adopted in the AFLC with the previous work. The total storage size S_{KB} for knowledge base in AFLC, can be given by

$$S_{\mathrm{KB}} = S_{\mathrm{IFM}} + S_{\mathrm{MFM}} + S_{\mathrm{RIM}} + S_{\mathrm{RWM}}$$

where S_{IFM}, S_{MFM}, S_{RIM}, and S_{RWM} are the storage sizes for the IFM, MFM, RIM, and RWM, respectively. Given a fuzzy system with m input variables and n output variables using r rules, let the numbers of membership functions specifying each input variable and output variable be a and b, respectively, the universe of discourse be u elements, and the grades of a membership function be discretized into g levels. Assume that the fuzziness degree of each input and the contribution weight of each rule are encoded with k and l bits, respectively, we know $S_{\mathrm{MFM}} = S_L \times (m \times a + n \times b)$, $S_{\mathrm{RIM}} = (m \times \lceil \log_2 a \rceil + n \times \lceil \log_2 b \rceil) \times r$, $S_{\mathrm{IFM}} = k \times m$, and $S_{\mathrm{RWM}} = l \times r$, where S_L is the storage size for digitization of a membership function.

In the AFLC realization, $m = 2$, $n = 1$, $r = 50$, $a = 8$, $b = 16$ and $u = 64$, $g = 16$, $k = 3$, and $l = 4$. A membership function is represented by a triple, so $S_L = 15$ bits. Thus, the AFLC's $S_{\mathrm{IFM}} = 6$ bits, $S_{\mathrm{MFM}} = 480$ bits, $S_{\mathrm{RIM}} = 500$ bits, and $S_{\mathrm{RWM}} = 200$ bits. Thus, $S_{\mathrm{KB}} = 1186$ bits. In [8] and [9], $S_L = u \times \log_2 g = 256$ bits. Thus, their $S_{\mathrm{MFM}} = 8192$ bits. Chiueh [10] and Eichfeld [20] use an optimized memory organization in which the overlapped membership functions are stored in different memories. Then, their $S_L = 256$ bits and $S_{\mathrm{MFM}} = 1536$ bits. Using different numbers of input and output variables, the three designs' S_L and S_{MFM} are listed in Table II. Compared with the previous work, a large storage space is saved in our design.

Each membership function is represented by a triple in the AFLC. We need to update only 15 bits of data for the change of

TABLE II
S_L AND S_{MFM} FOR THE THREE DESIGNS

	4 input variables 1 output variables		6 input variables 1 output variables		6 input variables 3 output variables	
	S_L	S_{MFM}	S_L	S_{MFM}	S_L	S_{MFM}
Togai [8] & Watanabe [9]	32	1536	32	2048	32	3072
Chiueh [10] & Eichfeld [24]	32	320	32	448	32	576
Our design	2	96	2	128	2	192

Unit: Bytes (1 byte=8 bits)

a membership function and 480 bits of data if all membership functions are changed. Previous designs [8]–[10] need to update 256 bits of data for each changed membership function and 8192 bits of data if all membership functions are changed. Obviously, such long updating time makes their system unsuitable for the applications of on-line adaptation.

Due to the reduction in memory size and updating time, our design must use the DMFG to dynamically generate a membership function. In previous FLC's [7]–[10], [20], the on-line membership function generating approach was not used, so the 64 grades used for representing a membership function are directly stored in the memory. The gate count of DMFG is about 470, and the gate count of the memory used for storing a triple is about 70, so the total gate count used for a membership function is about 540 in the AFLC. As compared with the previous designs [8]–[10], whose gate count of the memory used for storing a membership function is estimated as about 1150, the DMFG saves a lot of area.

V. APPLICATIONS OF THE AFLC

A. Example 1—Truck Backing-Up Control

Backing a truck to a loading dock is a nonlinear control problem, which is difficult to solved by using traditional control methods. It has been the object of several studies in the literature of fuzzy control and neural networks [5] and is suitably applied as a typical high-speed application of FLC. We first demonstrate the practicability of AFLC by the example.

The three state variables ϕ, x, and y precisely determine the truck position, where ϕ specifies the angle of the truck with the horizontal and the coordinate pair (x, y) specifies the position of the rear center of the truck. According to the steering angle θ, the truck moves backward by a fixed unit distance every step. The goal of AFLC is to control the truck back up to the loading dock, $(x, \phi) = (300\,90)$. The input variables are the x-position coordinate x and the truck angle ϕ. The output variable is the steering angle. According to our experiments, the simulated truck controlled by the AFLC can back up to the loading dock from any initial position and any angle in the loading zone. Fig. 8 shows the truck trajectories from six initial positions chosen randomly.

Additionally, a behavior model of AFLC using floating-point arithmetic (the software AFLC) is constructed and run on the same platform. For the software AFLC and the AFLC, the universe of discourse is set to 64 and the COG algorithm is used as the defuzzification method. The difference is that the software AFLC uses the floating-point arithmetic and the AFLC uses discrete hardware computations, respectively, to perform the arithmetic operations. Chosen randomly, the three initial positions, $(x, \phi) = (80, 210), (380, 70),$ and $(500, -70)$, are applied to both to get the truck trajectories shown in Fig. 9. We see that both of them successfully control the truck back to the desired position, although the AFLC does not produce smooth trajectories as well as the software AFLC does. The main reason that the AFLC has a larger deviation near the loading line $x = 300$ might be the roundoff errors caused by discrete integer operations of the digital hardware. By using 50 initial positions chosen randomly, we find the average roundoff error between the software AFLC and the AFLC is about 1.7%. With the results, we demonstrate that the AFLC produces the desired outputs just as a well-performed FLC does.

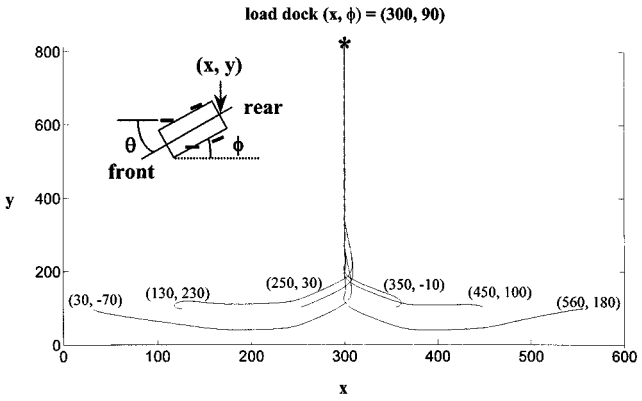

Fig. 8. The trajectories of the truck controlled by the AFLC.

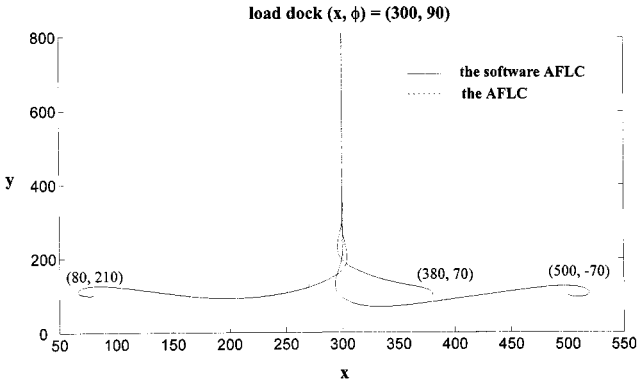

Fig. 9. The truck trajectories for the AFLC and the software AFLC.

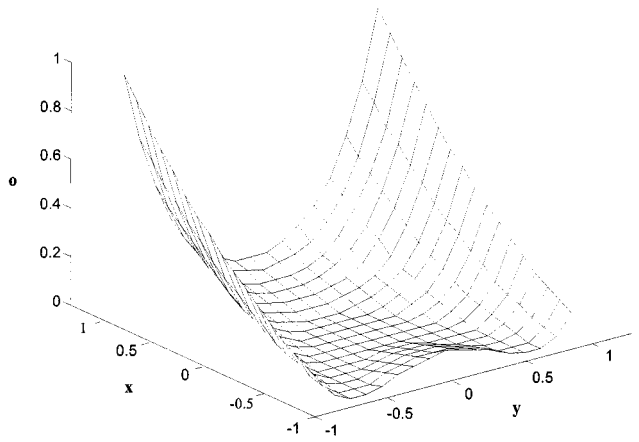

Fig. 10 The control surface of Example 2 using the training data pairs.

B. Example 2—Nonlinear Control

This example is used to demonstrate the off-line adaptive application of AFLC. In the case, the nonlinear example to be controlled is represented by

$$o = (2 \cdot x + 4 \cdot y^2 + 0.1)^2.$$

It has two inputs and one output. Assuming that all fuzzy control rules are unknown, a novel supervised learning algorithm [21] specialized for the AFLC is applied to generate the rules in advance. With the help of computer simulation, we calculate the source output values according to the various combinations of the input values in advance for the equation. Totally, 300 pairs of source input–output values are collected as the training data. Fig. 10 shows the three-dimensional (3-D) control surface according to the training (correct) data pairs.

The specialized supervised learning algorithm gradually generates all control rules in accordance with the 300 training data pairs. After ten learning epochs, the control surface generated by the AFLC is shown in Fig. 11. Basically, both Figs. 10 and 11 have similar frames. Because of such digital hardware limitations as the roundoff errors caused by integer operations and coarse resolutions of the membership function, the control surface generated by the AFLC is not as smooth as that generated by the training data pairs.

C. Example 3—Cart-Pole Balancing Control

In the last example, we apply our AFLC to another interesting control problem—cart-pole balancing. A pole is hinged to a motor-driven cart, which moves on rail tracks to its right or its left. The dynamics of the cart-pole system are characterized by four input state variables $(x, \dot{x}, \theta, \dot{\theta})$ and one output variable (f), where x is the position of the cart on the track, $\dot{x}$ is the velocity of the cart, θ is the angle of the pole with respect to the vertical axis, $\dot{\theta}$ is the angular velocity of the pole, and f is the force applied to the cart.

The primary control task is to keep the pole vertically balanced, by which we mean the pole never deviates more than 12° from the vertical. When $|\theta| > 12°$, a failure happens and the controller receives the failure signal. Then the cart-pole system is reset to its initial state and a new attempt begins. The failure

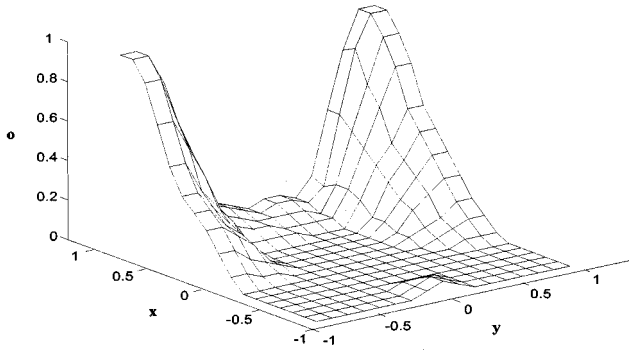

Fig. 11. The control surface generated by the AFLC for Example 2.

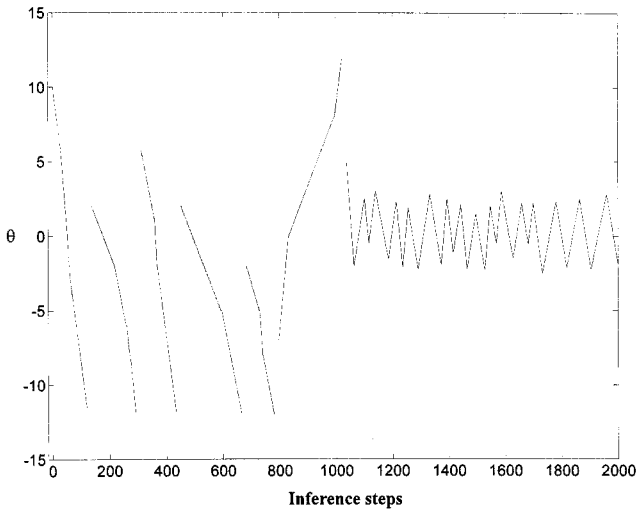

Fig. 12. The performance of Example 3 using random initial states after each failure.

signal occurs only after a long sequence of control decisions. Therefore, the controller must develop its own strategy and learn to balance the pole as long as possible. Obviously, it is a reinforcement learning problem. We have applied a learning mechanism, using the reinforcement learning algorithm proposed by Lee [16], and combined it with our AFLC. The learning mechanism will tune the locations of the consequent membership functions at every inference before the failure signal comes. As shown in Fig. 12, the system can learn to balance the pole after a couple of failures.

VI. Conclusions

In this paper, an adaptive fuzzy logic controller, which is suitable for on-line adaptation, is presented. Taking advantage of the adaptability provided by a symbolic fuzzy rule format and a dynamic membership function generator, as well as the high-speed integration capability afforded by VLSI, the proposed AFLC can perform an adaptive fuzzy inference process using various inference parameters, such as the shape and location of a membership function, dynamically and quickly. Three typical examples of nonlinear systems are used to illustrate the fundamental applications of the proposed AFLC, and the experimental outcomes show that the AFLC achieves excellent adaptability.

ACKNOWLEDGMENT

The authors would like to thank editor D. Bouldin and the anonymous reviewers for their valuable suggestions.

REFERENCES

[1] L. A. Zadeh, "Fuzzy sets," *Inf. Control*, vol. 8, pp. 338–353, 1965.
[2] M. Sugeno, "An introductory survey of fuzzy control," *Inf. Sci.*, vol. 36, pp. 59–83, 1985.
[3] E. H. Mamdani, "Applications of fuzzy algorithms for simple dynamic plant," *Proc. Inst. Elect. Eng.*, vol. 121, pp. 1585–1588, 1974.
[4] C. C. Lee, "Fuzzy logic in control systems: Fuzzy logic controller—Part I and Part II," *IEEE Trans. Syst., Man, Cybern.*, vol. 20, pp. 404–435, 1990.
[5] B. Kosko, *Neural Networks and Fuzzy Systems*. Englewood Cliffs, NJ: Prentice-Hall, 1992.
[6] E. Cox, "Adaptive fuzzy systems," *IEEE Spectrum*, pp. 27–31, Feb. 1993.
[7] T. Yamakawa, "A survey on fuzzy information processing hardware systems," in *Proc. IEEE Int. Conf. Circuits and Systems*, 1995, pp. 1310–1314.
[8] M. Togai and H. Watanabe, "Expert system on a chip: An engine for real-time approximate reasoning," *IEEE Expert Mag.*, vol. 1, pp. 55–62, Aug. 1986.
[9] 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, pp. 376–381, Apr. 1990.
[10] T. C. Chiueh, "Optimization of fuzzy logic inference architecture," *Computer*, pp. 67–71, May 1992.
[11] M. Sasaki, F. Ueno, and T. Inoue, "7.5 MFLIPS fuzzy microprocessor using SIMD and logic-in-memory structure," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1993, pp. 527–534.
[12] S. Lee and Z. Bien, "Design of expandable fuzzy inference processor," *IEEE Trans. Consumer Electron.*, vol. 40, pp. 171–175, May 1994.
[13] H. R. Berenji, "Fuzzy logic controllers," in *An Introduction to Fuzzy Logic Applications in Intelligent Systems*, R. R. Yager and L. A. Zadeh, Eds. Boston, MA: Kluwer, 1992, pp. 69–96.
[14] Z. Q. Wu, P. Z. Wang, H. H. Teh, and S. S. Song, "A rule self-regulating fuzzy controller," *Fuzzy Sets Syst.*, vol. 47, pp. 13–21, 1992.
[15] W. C. Daugherity, B. Rathakrishnan, and J. Yen, "Performance evaluation of a self-tuning fuzzy controller," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1992, pp. 389–397.
[16] C. C. Lee, "A self-learning rule-based controller employing approximate reasoning and neural net concepts," *Int. J. Intell. Syst.*, vol. 6, pp. 71–93, Jan. 1991.
[17] A. G. Barto, R. S. Sutton, and C. W. Anderson, "Neuronlike adaptive elements that can solve difficult learning control problems," *IEEE Trans. Syst., Man, Cybern.*, vol. SMC-13, pp. 834–846, Sept. 1983.
[18] H. R. Berenji and P. Khedkar, "Learning and tuning fuzzy logic controllers through reinforcements," *IEEE Trans. Neural Networks*, vol. 3, pp. 724–740, Sept. 1992.
[19] "0.8 μm CMOS standard cell library," Computer & Communication Lab., Industry Technology Res. Instit., Taiwan, R.O.C., 1993.
[20] H. Eichfeld, M. Löhner, and M. Müller, "Architecture of a CMOS fuzzy logic controller with optimized memory organization and operator design," in *Proc. IEEE Int. Conf. Fuzzy Systems*, 1992, pp. 317–323.
[21] J. M. Jou and P.-Y. Chen, "A hybrid adaptive fuzzy system using error backpropagation learning," in *Proc. CFSA/IFIS/SOFT'95 Fuzzy Theory and Applications*, 1995, pp. 537–542.

Jer Min Jou received the Ph.D. degree in electrical engineering and computer science from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1987.

Since 1989, he has been an Associate Professor in the Department of Electrical Engineering and Computer Science, National Cheng Kung University. His current research interests include application-specific integrated circuit design/synthesis, hardware-software codesign, VLSI CAD, and asynchronous circuit design.

Dr. Jou received a Distinguished Paper Citation at the 1987 IEEE ICCAD Conference in Santa Clara, CA. He received the Longterm Best Paper Award from the Acer Foundation in 1998. He has been a Reviewer for many journals, including the IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS.

Pei-Yin Chen received the B.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1986, the M.S. degree in electrical engineering from Pennsylvania State University, University Park, in 1990, and the Ph.D. degree in electrical engineering from National Cheng Kung University in 1999.

He is currently an Associate Professor in the Electronic Engineering Department at Southern Taiwan University of Technology, Tainan. His research interests include fuzzy logic control, gray prediction, data compression, and VLSI chip design.

Sheng-Fu Yang received the M.S. degree in electrical engineering from National Cheng Kung University, Tainan, Taiwan, R.O.C., in 1995.

His research interests include fuzzy logic and VLSI chip design.